An Open Pipeline for Generating Executable Neural Circuits from Fruit Fly Brain Data

Lev E. Givon

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

ABSTRACT

An Open Pipeline for Generating Executable Neural Circuits from Fruit Fly Brain Data

Lev E. Givon

Despite considerable progress in mapping the fly's connectome and elucidating the patterns of information flow in its brain, the complexity of the fly brain's structure and the still-incomplete state of knowledge regarding its neural circuitry pose significant challenges beyond satisfying the computational resource requirements of current fly brain models that must be addressed to successfully reverse the information processing capabilities of the fly brain. These include the need to explicitly facilitate collaborative development of brain models by combining the efforts of multiple researchers, and the need to enable programmatic generation of brain models that effectively utilize the burgeoning amount of increasingly detailed publicly available fly connectome data.

This thesis presents an open pipeline for modular construction of executable models of the fruit fly brain from incomplete biological brain data that addresses both of the above requirements. This pipeline consists of two major open-source components respectively called Neurokernel and NeuroArch.

Neurokernel is a framework for collaborative construction of executable connectome-based fly brain models by integration of independently developed models of different functional units in the brain into a single emulation that can be executed upon multiple Graphics Processing Units (GPUs). Neurokernel enforces a programming model that enables functional unit models that comply with its interface requirements to communicate during execution regardless of their internal design. We demonstrate the power of this programming model by using it to integrate independently developed models of the fly retina and lamina

into a single vision processing system. We also show how Neurokernel's communication performance can scale over multiple GPUs, number of functional units in a brain emulation, and over the number of communication ports exposed by a functional unit model.

Although the increasing amount of experimentally obtained biological data regarding the fruit fly brain affords brain modelers a potentially valuable resource for model development, the actual use of this data to construct executable neural circuit models is currently challenging because of the disparate nature of different data sources, the range of storage formats they use, and the limited query features of those formats complicates the process of inferring executable circuit designs from biological data. To overcome these limitations, we created a software package called NeuroArch that defines a data model for concurrent representation of both biological data and model structure and the relationships between them within a single graph database. Coupled with a powerful interface for querying both types of data within the database in a uniform high-level manner, this representation enables construction and dispatching of executable neural circuits to Neurokernel for execution and evaluation.

We demonstrate the utility of the NeuroArch/Neurokernel pipeline by using the packages to generate an executable model of the central complex of the fruit fly brain from both published and hypothetical data regarding overlapping neuron arborizations in different regions of the central complex neuropils. We also show how the pipeline empowers circuit model designers to devise computational analogues to biological experiments such as parallel concurrent recording from multiple neurons and emulation of genetic mutations that alter the fly's neural circuitry.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to extend special thanks to my adviser Prof. Aurel A. Lazar for his novel insights into studying the information processing properties of the brain that led me to become involved in the exciting field of computational neuroscience. His incisive feedback on my ideas, willingness to discuss ongoing research at all hours, and securing of cutting-edge parallel computing equipment were instrumental in making the work presented in this dissertation possible. Parts of this dissertation were developed in collaboration with my current Bionet Group colleagues Nikul H. Ukani, Chung-Heng Yeh, and Yiyin Zhou, whom I thank for their tireless work and excellent ideas. I would like to commend them and Bionet Group alumni Anmo Kim, Wenze Li, Eftychios A. Pnevmatikakis, Konstantinos Psychas, Yevgeniy B. Slutskiy, and Robert J. Turetsky for maintaining a supportive lab atmosphere that fosters genuine collaboration and mutual assistance when tackling the challenges that arise in the course of research. I would also like to thank Prof. Brian McCabe for useful feedback regarding part of the work presented in this thesis.

I would like to thank Juergen Berger for kindly permitting reuse of his fruit fly photograph and thank Nacho Vizcaíno, Richard Benton, Bertram Gerber, and Matthieu Louis for permitting reuse of the robot fly image they composed for the ESF-EMBO 2010 Conference on Functional Neurobiology in Minibrains.

I would like to thank Profs. Paul Sajda, Christine Hendon, Nima Mesgarani, and Matei Ciocarlie for taking the time to serve on my dissertation committee and provide useful feedback regarding my research. I also would like to thank Prof. Sajda and Prof. Roxana Geambasu for having served on my thesis proposal committee.

I would like to thank the host of developers in the open-source scientific computing

community who were not only instrumental in developing the software that made the work presented in this thesis possible, but took the time to personally address questions that arose in the course of my research work. In particular, I would like to express appreciation to Rolf Vandevaart (NVIDIA) for his assistance in debugging issues regarding GPU support in OpenMPI [42], Dr. Andreas Klöckner (UIUC) for valuable feedback related to the use of Python with GPUs and for his excellent PyCUDA [71] package, and Dr. Lisandro Dalcin (KAUST) for addressing questions regarding the use of Python and MPI via his mpi4py package [28].

I would like to gratefully acknowledge the Electrical Engineering Department of Columbia University, the Center for Neural Engineering and Computation at Columbia University, the Dean's Office of the Fu Foundation School of Engineering and Applied Science at Columbia University, Columbia Business School, Columbia Technology Ventures, CaseRails Inc., the Air Force Office of Scientific Research (grant #FA9550-12-10232), and the National Science Foundation (grant #1544383) for their financial support of my research. Special thanks are due to the Engineering Graduate Student Council of Columbia University for awarding me a Professional Development Scholarship (funded by the Dean's Office) that sponsored part of my research work.

I would like to thank my parents Yokhai and Frieda Givon, my brother Shai, and my late grandmother Gita Simkin for always being there for me throughout graduate school. I would also like to gratefully acknowledge the many friends and relatives in New York City who kindly opened their doors to me and my wife during our years in graduate school when we needed a breather between research projects - in particular, Dr. Perry and Margy-Ruth Davis, Steve and Naomi Wolinsky, David and Ceil Olivestone, R. Arthur and Sarah Marmorstein, R. Robert and Beile Block, and Drs. Heshy and Linda Friedman.

Finally, I would like to thank my dear wife Dr. Yvette Y. Yien for her constant support, wisdom, and love.

To my dear wife Yvette.

רבות בנות עשו חיל ואת עלית על כלנה

Chapter 1

Introduction

1.1 Motivation

Reverse engineering the information processing functions of the brain is an engineering grand challenge of immense interest that has the potential to drive important advances in computer architecture, artificial intelligence, and medicine. The human brain is an obvious and tantalizing target of this effort however, its structural and architectural complexity place severe limitations upon the extent to which models built and executed with currently available computational technology can relate its biological structure to its information processing capabilities. Successful development of human brain models must therefore be preceded by an increased understanding of the structural/architectural complexity of the more tractable brains of simpler organisms and how they implement specific information processing functions and govern behavior [66].

The nervous system of the fruit fly *Drosophila melanogaster* possesses a range of features that recommend it as a model organism of choice for relating brain structure to function. Despite the obvious differences in size and complexity between the mammalian and fruit fly brains, researchers dating back to Cajal have observed common design principles in the structure of their sensory subsystems [124] and striking similarities in circuits underlying regulation of behavioral actions [131]. Many of the genes and proteins expressed in the

mammalian brain are also conserved in the genome of *Drosophila* [3]. These features strongly suggest that valuable insight into the workings of the mammalian brain can be obtained by focusing on that of *Drosophila*.

Remarkably, the fruit fly is capable of a host of complex nonreactive behaviors that are governed by a brain containing only $\sim 10^5$ neurons and $\sim 10^7$ synapses organized into fewer than 50 distinct functional units, many of which are known to be directly involved in functions such as sensory processing, locomotion, and control [22]. The relationship between the fruit fly's brain and its behaviors can be experimentally probed using a powerful toolkit of genetic techniques for manipulation of the fruit fly's neural circuitry such as the GAL4 driver system [35, 119, 129, 137, 85], recent advances in experimental methods for precise recordings of the fruit fly's neuronal responses to stimuli [68, 140, 69], techniques for analyzing the fly's behavioral responses to stimuli [15, 84, 23], and progress in reconstruction of the fly connectome, or neural connectivity map [24, 134]. These techniques have provided access to an immense amount of valuable structural and behavioral data that can be used to model how the fruit fly brain's neural circuitry implements processing of sensory stimuli [41, 93, 22, 61, 94, 125].

Biological experimentation techniques alone are essential but insufficient for determining how the fly brain implements specific functions; to paraphrase Richard Feynman, our understanding of the brain remains incomplete if we fail to create an executable brain model that replicates its functions. Despite considerable progress in mapping the fruit fly's connectome and elucidating the patterns of information flow in its brain, the complexity of the fly brain's structure and the still-incomplete state of knowledge regarding its neural circuitry pose significant challenges to the translation of biological data to executable hypotheses that can be tested *in silico*. Although the highly parallel structure of neural circuits demands computationally efficient means of testing brain models, attempting to optimize the performance of neural circuit simulations given current uncertainty regarding the appropriate computational modeling paradigm to employ is premature and overlooks other requirements that must be

met to successfully emulate the fly brain. When seen in light of the relative tractability of the fly brain, the dramatic increase in power of commodity parallel computing technology at the disposal of neuroscientists over the past decade affords the opportunity to effectively address those requirements without being overly preoccupied by the computational needs of executable brain models.

Its relative tractability nothwithstanding, the complexity of the fly brain is such that effectively combining the efforts of multiple researchers is essential to the successful reverse engineering of the fly brain. The multiplicity of neural simulators and modeling tools currently available complicates collaborative model construction because of the difficulty of combining models of different parts of the fly brain devised by different researchers. While open neural formats for simulator-independent model specification are a step in the right direction, they currently lack a programming model that defines how different neural circuit models may communicate. This inability to combine hypotheses regarding the information processing capabilities of different parts of the brain prevents the neuroscience community from pursuing an effective divide-and-conquer strategy to reverse engineering the fly brain's functions.

The second major requirement of developing a whole brain model is the need to translate biological data into testable hypotheses. The challenges of this requirement continue to mount as ever more detailed data regarding the fly connectome becomes available. Effective utilization of such extensive data sets requires that model construction be at least partially performed via algorithmic means as opposed to explicit manual specification of a neural circuit. Extracting data from large biological data sets for the purposes of algorithmic model generation can only be performed efficiently if the data is hosted in a database with a sufficiently powerful querying mechanism and exposed through an interface that explicitly provides developers with an API for accessing this mechanism from model generation applications. Existing open fly connectome datasets, however, are either hosted in a range of static files formats that cannot be directly queried or exposed through database query

interfaces that require manual interaction by a user. Similarly, the inevitable modification of circuit models that correspond to a significant portion of the fly brain and/or employ a high level of detail requires a means of directly and efficiently querying a model's structure. Although open neuronal model sharing resources do exist, they do not provide any means for directly querying the internal structure of the models that they host.

1.2 Approach and Contributions

Overall Contribution The intersection of an increasing abundance of fruit fly biological data with the availability of commodity computing hardware of previously unavailable power affords neuroscientists with an unprecedented opportunity to advance towards the goal of reverse engineering the fly brain. To enable the neuroscience community to seize this opportunity, this thesis addresses the above challenges by presenting an open brain modeling pipeline explicitly designed for collaborative model development and for algorithmic construction of executable circuits from biological data (Fig. 1.1). To foster the open science approach advanced by this work, the designs of the software and models presented in this thesis were developed in a series of Requests for Comments (RFCs) [12] and Jupyter [111] notebooks made available to the public. These RFCs contain detailed descriptions of the software presented in § 2 and § 3 and the models presented in § 2.3, § 3.4.5, § 3.6 and § 4 that use them. Both the RFCs [80, 73, 46, 44] and Jupyter notebooks are publicly available on the Neurokernel project website `http://neurokernel.github.io/docs.html`.

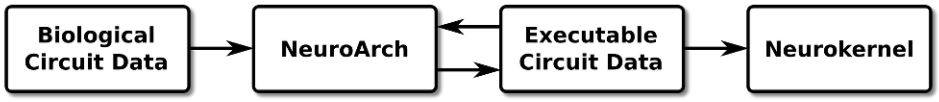


Figure 1.1: Pipeline for generating executable circuits from biological data. NeuroArch's data model and query API enables sophisticated querying of loaded biological data to generate neural circuit models for execution by Neurokernel. Executable circuits designed and constructed by independent researchers are also stored in NeuroArch to enable their subsequent refinement after evaluation by Neurokernel.

Neurokernel The Neurokernel framework enables implementation of models of the constituent functional units in the fly brain that can be executed on multiple Graphics Processing Units (GPUs). In order to achieve scaling over multiple computational resources while providing the programmability required to develop such models, Neurokernel's architecture provides GPU resource management and programming services to brain emulations analogous to those an operating system kernel provides to software applications. A key feature of Neurokernel is its enforcing of a programming model that provides models of functional units of the fly brain called Local Processing Units (LPUs) with a mandatory communication interface. This enables LPU models developed by independent researchers to be combined into a comprehensive brain model even if they possess different internal designs. The Neurokernel core developed by the author[1] provides fly brain researchers with the following key components for developing brain emulations:

- a set of Python classes that LPU developers can subclass to implement new LPUs that utilize NVIDIA GPU hardware via PyCUDA[71] without having to explicitly invoke any communication services;
- support for an XPATH-like identifier syntax to facilitate labeling and management of large numbers of communication ports exposed by each LPU and connectivity pattern;
- a class for construction of inter-LPU connectivity patterns from a variety of inputs such as XPATH-like selectors, tables, or bipartite graphs of the connections between two sets of ports;
- a data structure for mapping XPATH-like identifiers and selectors to arrays of GPU memory that can be used by an LPU's internal logic to read from and write to communication ports using their identifiers rather than integer indexes;
- an inter-LPU communication mechanism that automatically utilizes OpenMPI [42] to

[1] `http://github.com/neurokernel/neurokernel`

exploit peer-to-peer memory transfer features of modern GPU technology to accelerate transmission of data between LPUs executed on different GPUs.

- an emulation manager that distributes LPU classes in a specified emulation to GPUs for instantiation and execution with no user intervention;
- an emulation launcher that sets up the MPI environment required by a Neurokernel emulation with almost no user intervention.

Further details regarding Neurokernel are presented in § 2; a demonstration of Neurokernel's use in integrating multiple LPUs into a model of part of the fruit fly's vision system developed by Konstantinos Psychas, Nikul Ukani, and Yiyin Zhou is presented in § 2.3.1.

Neurodriver Being that the Neurokernel core does not prescribe any specific neuron or synapse models, the author, Nikul Ukani, Chung-Heng Yeh, and Yiyin Zhou jointly implemented a Python package called Neurodriver[2] that provides an LPU implementation with extensible support for a range of point neuron models such as Leaky Integrate-and-Fire, Morris-Lecar, and Hodgkin-Huxley, and conductance-based synapse models. This package enables construction of LPU models without having to write any Python code by providing support for loading LPU circuits composed of supported neurons and synapses and inter-LPU connectivity saved in a format such as the Graph Exchange Format (GEXF) [3]. Since all communication between LPUs is handled by Neurokernel, LPU models based upon Neurodriver can automatically interact with other LPUs without any effort on the part of the model developer. New GPU-based neuron and synapse models can be added by means of a simple plugin architecture. Neurodriver is described in § 2.2.6 and its performance shown to be competitive with that of other GPU-based simulation engines that support Python for the scenarios currently targeted by Neurokernel in § 2.3.3

[2] `http://github.com/neurokernel/neurodriver`

[3] `http://www.gexf.net`

NeuroArch Although the support for loading LPU circuits obviates the need to explicitly implement many LPU designs in Python, manual specification and revision of circuit models becomes increasingly cumbersome as they increase in complexity to account for higher levels of biological detail. The growing number and magnitude of available datasets comprising fruit fly connectome and genetic data also raises significant challenges as to how to effectively utilize large-scale biological data stored in a disparate set of formats and databases to create more accurate brain models. To overcome these impediments to model development, the author and Nikul Ukani designed a graph database platform called NeuroArch for facilitating algorithmic generation of executable neural circuit models from biological data. The current Python implementation of this database by the author

- provides an extensible data model that enables concurrent representation of both biological and executable circuit model data in a linked hierarchy of entities corresponding to different levels of circuit subdivision or modeling abstraction;
- exploits this hierarchy to enable queries on biological and/or executable circuit data designed to extract subgraphs of interest (e.g., circuit motifs, LPUs, connectivity patterns, etc.) for analysis or execution;
- encapsulates queries in a manner that permits them to be composed into more complex queries via set operations without having to explicitly write any query;
- provides an extensible Object Graph Mapping (OGM) that enables entities in NeuroArch's graph database to be accessed as Python class instances whose methods conveniently expose the above query mechanism;
- exposes query results via a multimodal view interface that permits them to be easily accessed either as tabular or graph-based Python data structure using widely used Python data analysis packages;

- utilizes an enterprise open source property graph database platform[4] to support complex internal graph traversals;
- provides a means of exporting stored executable circuits to a form that Neurokernel can execute (provided that the circuit uses neuron and synapse models supported by Neurodriver);
- provides loaders for importing fly connectome data from disparate public sources such as NeuroMorpho [4] and Janelia Research [1] and executable circuit data for LPU models designed by other researchers into NeuroArch's graph database.

Further details regarding NeuroArch are presented in § 3.

An Executable Model Generator for the Fruit Fly Central Complex To show that Neurokernel and NeuroArch enable algorithmic construction of executable models of portions of the fly brain by inferring circuit structure from incomplete biological information, the author designed and implemented a system for generating a model of the fruit fly's central complex that utilizes NeuroArch to facilitate inference of synaptic connections between neurons from incomplete neuron arborization information. The use of NeuroArch also affords the flexibility to generate LPU circuits that utilize different internal components. Details regarding the biological data used by this system, the evaluation of generated circuits by Neurokernel, and a demonstration of how NeuroArch's API enables rapid modification of the generated circuit to evaluate a range of model variations are presented in § 4.

scikit-cuda There exist a range of free software libraries that provide many powerful GPU-based numerical routines that are potentially useful in LPU implementations, but provide no direct access to these routines from Python. To enable their use in Neurokernel, the author developed a package called scikit-cuda[5] [47] that provides high-level numerical

[4] `http://orientdb.com`

[5] `http://github.com/lebedov/scikit-cuda`

Python functions similar to those in other high-level numerical computing packages such as NumPy [136] and SciPy [64] that utilize the aforementioned GPU-based libraries. This package is utilized by the retina/lamina model discussed in § 2.3.1. Since its initial release, the package has attracted numerous contributions from researchers and developers in a range of computational fields.

Chapter 2

Neurokernel: a Framework for Integration of Executable Fruit Fly Brain Models

2.1 Introduction

Neurokernel's design is predicated upon the organization of the fruit fly brain into a fixed number of functional modules characterized by local neural circuitry called Local Processing Units (LPUs); we review the anatomy of the fruit fly brain that motivates this design in § 2.2.1 and describe Neurokernel's architecture, support for GPU resources, and programmability in § 2.2.2. Neurokernel explicitly enforces a programming model for implementing models of these functional modules that separates their internal design from the connectivity patterns that link their external communication interfaces; this modular architecture facilitates collaboration between researchers focusing on different functional modules in the fly brain by enabling models independently developed by different researchers to be integrated into a single whole brain model irrespective of their internal designs. We present

Parts of this chapter appear in [44].

Neurokernel's programming model in § 2.2.3 and detail its API in § 2.2.4. We also present a configurable LPU implementation called Neurodriver that can be used to construct LPUs using several common neuron and synapse models without writing any code; this package is described in § 2.2.6. To illustrate the use of Neurokernel's API, we use it it to integrate independently developed models of the retina and lamina neuropils in the fly's visual system; this integration is described in § 2.3.1. We provide performance benchmarks of Neurokernel's module communication services in § 2.3.2 that demonstrate its ability to exploit technology for accelerated data transmission between multiple GPUs to achieve scalable performance. We also benchmark Neurodriver's performance and compare it with another recently developed GPU-based Python neuronal network simulation engine in § 2.3.3 to demonstrate Neurodriver's suitable performance when executing neural circuits comprising numbers of neurons and synapses similar to those found in actual neuropils. We review existing neural simulation tools and projects in the context of fly brain emulation and compare them with Neurokernel in § 2.4. Finally, we conclude the section with § 2.5.

2.2 Framework Design and Features

2.2.1 Modeling the Fruit Fly Brain

Analysis of the *Drosophila* connectome has revealed that its brain can be decomposed into fewer than 50 distinct neural circuits, most of which correspond to anatomically distinct regions in the fly brain [22]. These regions, or neuropils, include sensory circuits such as the olfactory system's antennal lobe and the visual system's lamina and medulla, as well as control and integration neuropils such as the protocerebral bridge and ellipsoid body (Fig. 2.1). Neuropils range in size from about 6,000 neurons (lamina) to 40,000 neurons (medulla). Most of these modules are referred to as local processing units (LPUs) because they are characterized by unique populations of local neurons whose processes are restricted to specific neuropils.

The axons of an LPU's local neurons and the synaptic connections between them and other neurons in the LPU constitute an internal pattern of connectivity that is distinct from the bundles, or tracts, of projection neuron processes that transmit data to neurons in other LPUs (Fig. 2.1); this suggests that an LPU's local neuron population and synaptic connections largely determine its functional properties. While the connection densities within and between LPUs is not fully known, the total strength of connections between LPUs (defined in terms of total numbers of dendritic and axonal terminals for all projection neurons linking a LPU with other LPUs) has been observed to vary between 600 and 44,000 for a sample of 13,000 projection neurons in the adult *Drosophila* brain [127]. The fruit fly brain also comprises modules referred to as hubs that contain no local neurons; they appear to serve as communication relays between different LPUs.

In contrast to a purely anatomical subdivision, the decomposition of the brain into functional modules casts the problem of reverse engineering the brain as one of discovering the information processing performed by each individual LPU and determining how specific patterns of axonal connectivity between these LPUs integrates them into functional subsystems. Modeling both these functional modules and the connectivity patterns that link them independent of the internal design of each module is a fundamental requirement of Neurokernel's architecture.

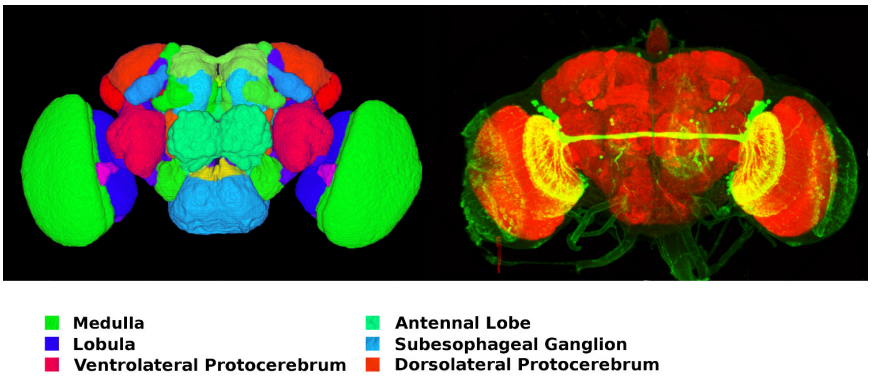


Figure 2.1: Modular structure of fruit fly brain. Individual neuropils are identified by different colors in the left-hand figure, with the names of several major neuropils listed. Most neuropils are paired across the fly's two hemispheres. The right-hand figure depicts a tract of neuronal axons connecting neuropils across hemispheres highlighted in yellow (image created using data and software from [108, 109, 107], reproduced with permission).

2.2.2 Architecture of the Neurokernel

We refer to our software framework for fruit fly brain emulation as a *kernel* because it aims to provide two classes of functions associated with traditional computer operating systems [72]: it must serve as a *resource allocator* that enables the scalable use of parallel computing resources to accelerate the execution of an emulation, and it must serve as an *extended machine* that provides software services and interfaces that can be programmed to emulate and integrate functional modules in the fly brain.

Neurokernel's architectural design consists of three planes that separate between the time scales of a model's representation and its execution on multiple parallel processors (Fig. 2.2). Each plane exposes a vertical API that provides abstractions/services of that plane to higher level planes; this enables development of new features within one plane while minimizing the need to modify code associated with other planes. Services that implement the computational primitives and numerical methods required to execute supported models on parallel processors are provided by the framework's *compute plane*. Translation or mapping of a models' specified components to the methods provided by the compute

plane and management of the parallel hardware and data communication resources required to efficiently execute a model is performed by Neurokernel's *control plane*. Finally, the framework's *application plane* provides support for specification of neural circuit models, connectivity patterns, and interfaces that enable independently developed models of the fly brain's functional subsystems to be interconnected; we describe these interfaces in greater detail in § 2.2.4.

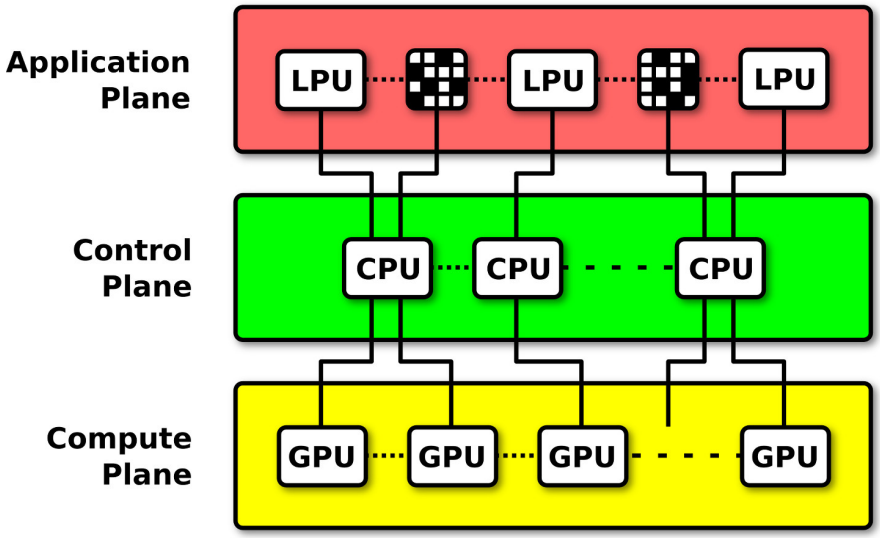


Figure 2.2: The three-plane structure of the Neurokernel architecture is based on the principle of separation of time scales. The application plane provides support for hardware-independent specification of LPUs and their interconnects. Services that implement the neural primitives and computing methods required to execute neural circuit model instantiations on GPUs are provided by the compute plane. Translation or mapping of specified model components to the methods provided by the compute plane and management of multiple GPUs and communication resources is performed by the control plane operating on a cluster of CPUs.

2.2.3 Neurokernel Programming Model

2.2.3.1 Interface Configuration

A key aspect of Neurokernel's design is the separation it imposes between the internal processing performed by an LPU model and how that model communicates with other models (Fig. 2.3). Neurokernel's programming model requires that one specifies how an LPU's interface is configured and connected to those of other LPUs. The interface of an LPU must be described exclusively in terms of communication ports that either transmit data to or receive data from ports exposed by other LPUs after each execution step. Each port must be configured either to receive input or emit output, and must be configured to either accept spike data represented as boolean values or graded potential data represented as floating point values (Fig. 2.4). Both of these settings are mutually exclusive; a single port may not both receive input and emit output, nor may it accept both spike and graded potential data. Ports may be connected to arbitrary internal components of an LPU; a graded potential port, for example, need not be associated with a neuron model's membrane voltage. Ports are uniquely specified relative to other ports within an interface using a path-like identifier syntax to facilitate hierarchical organization of large numbers of ports (Tab. 2.1).

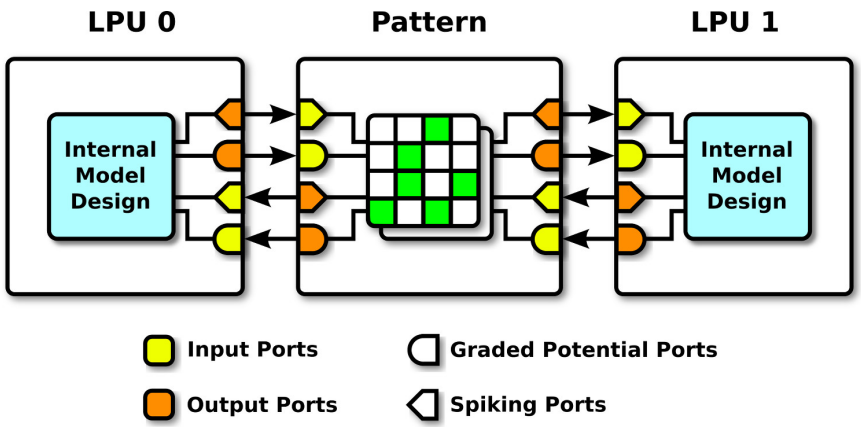


Figure 2.3: Neurokernel programming model. An LPU model's internal components (cyan) are exposed via input and output ports (yellow and orange). Connections between LPUs are described by patterns (green) that link the ports of one LPU to those of another. Connections may only be defined between ports of the same transmission type.

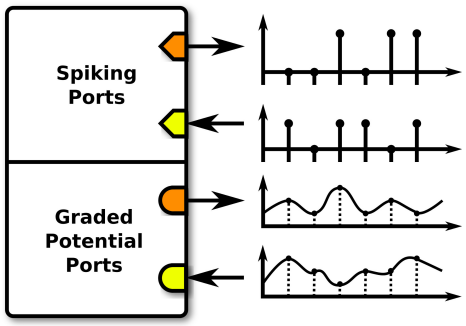


Figure 2.4: LPU interface. Each communication port must either receive input (yellow) or emit output (orange), and must either transmit spikes (diamonds) or graded potentials (circles).

Identifier/Selector	Comments
`/med/L1[0]`	selects a single port
`/med/L1/0`	equivalent to `/med/L1[0]`
`/med+/L1[0]`	equivalent to `/med/L1[0]`
`/med/[L1,L2][0]`	selects two ports
`/med/L1[0,1]`	another example of two ports
`/med/L1[0],/med/L1[1]`	equivalent to `/med/L1[0,1]`
`/med/L1[0:10]`	selects ten ports
`/med/L1/*`	selects all ports starting with `/med/L1`
`(/med/L1,/med/L2)+[0]`	equivalent to `/med/[L1,L2][0]`
`/med/[L1,L2].+[0:2]`	equivalent to `/med/L1[0],/med/L2[1]`

Table 2.1: Path-like port identifier and selector syntax examples. In these examples, the identifier level strings `med` and `L1` are chosen to respectively denote an LPU and a neuron within that LPU. An interface designer may select whichever level strings are deemed suitable to label ports in an interface, however.

2.2.3.2 Pattern Configuration

A single LPU may potentially be connected to many other LPUs; these connections must be expressed as patterns between pairs of LPUs (Fig. 2.3). Each pattern must be expressed in terms of (1) two interfaces - each comprising a set of ports - between which connections may be defined, (2) the actual connections between individual ports in the two interfaces (Tab. 2.2), and (3) the attributes of each port in the pattern's interfaces (Tab. 2.3).

Source Port	Destination Port
`/lam[0]`	`/med[0]`
`/lam[0]`	`/med[1]`
`/lam[1]`	`/med[2]`
`/med[3]`	`/lam[3]`
`/med[4]`	`/lam[4]`
`/med[4]`	`/lam[5]`

Table 2.2: Example of connections between ports in two LPUs respectively denoted `lam` and `med`. An instance of the `Pattern` class comprises these connections and the port attributes in Tab. 2.3.

Port attributes are used by Neurokernel to determine compatibility between LPU and

Port	Interface	I/O	Port Type
/lam[0]	0	in	graded potential
/lam[1]	0	in	graded potential
/lam[2]	0	out	graded potential
/lam[3]	0	out	spiking
/lam[4]	0	out	spiking
/lam[5]	0	out	spiking
/med[0]	1	out	graded potential
/med[1]	1	out	graded potential
/med[2]	1	out	graded potential
/med[3]	1	in	spiking
/med[4]	1	in	spiking

Table 2.3: Attributes of the ports in the connectivity pattern described in Tab. 2.2.

pattern objects. To provide LPU designers with the freedom to determine how to multiplex input data from multiple sources within an LPU, Neurokernel does not permit multiple input ports in a pattern to be connected to a single output port. Input ports in a pattern may be connected to multiple output ports. It should be noted that the connections defined by an inter-LPU connectivity pattern do not represent synaptic models; any synapses comprised by a brain model must be a part of the design of a constituent LPU and connected to the LPU's ports in order to either receive or transmit data from or to modeling components in other LPUs.

2.2.4 Application Programming Interface

In contrast to other currently available GPU-based neural emulation packages [38, 39, 96, 143, 10], Neurokernel is implemented entirely in Python, a high-level language with a rich ecosystem of scientific packages that has enjoyed increasing popularity in neuroscience research. Although GPUs can be directly programmed using frameworks such as NVIDIA CUDA and OpenCL, the difficulty of writing and optimizing code using these frameworks exclusively has led to the development of packages that enable run-time code generation (RTCG) using higher level languages [13]. Neurokernel uses the PyCUDA package to provide

RTCG support for NVIDIA's GPU hardware [71] and scikit-cuda [47] to provide high-level access to GPU-powered numerical libraries without forgoing the development advantages afforded by Python.

To make use of Neurokernel's LPU API, all LPU models must subclass a base Python class called `Module` that provides LPU designers with the freedom to organize the internal structure of their model implementations as they see fit independent of the LPU interface configuration. Implementation of a Neurokernel-compatible LPU requires that (1) the LPU be uniquely identified relative to all other LPUs to which it may be connected in a subsystem or whole-brain emulation, (2) the execution of all operations comprised by a single step of the LPU's emulation be performed by invocation of a single method called `run_step()`, and that (3) the LPU's interface be configured as described in § 2.2.3.1.

An instantiated LPU's graded potential and spiking ports are respectively associated with GPU data arrays that Neurokernel accesses to transmit data between LPUs during emulation execution; LPU designers are responsible for reading the data elements associated with input ports and populating the elements associated with output ports in the `run_step()` method. Modeling components that do not communicate with other LPUs and the internal connectivity patterns defined between them are not made accessible through the LPU's interface (Fig. 2.3).

Inter-LPU connectivity patterns correspond to the connections described by the tracts depicted in Fig. 2.1. These are represented by a tensor-like class called `Pattern` that contains the port and connection data described in § 2.2.3.2. To conserve memory, only existing connections are stored in a `Pattern` instance. In addition to manually constructing inter-LPU connectivity patterns using the configuration methods provided by the `Pattern` class, Neurokernel also supports loading connectivity patterns from CSV, GEXF, or XML files using a schema similar to NeuroML [48] with components that enable the specification of LPUs, connectivity patterns, and the ports they expose. Inter-LPU connections currently remain static throughout an emulation; future versions of Neurokernel will support dynamic

instantiation and removal of connections while a model is being executed.

The designer of an LPU is responsible for associating ports with internal components that either consume input data or emit output data. Neurokernel provides a class called `GPUPortMapper` that maps port identifiers to GPU data arrays; by default, each `Module` instance contains two `GPUPortMapper` instances that respectively associate the LPU's ports with arrays containing graded potential and spike values. After each invocation of the LPU's `run_step()` method, data within these arrays associated with the LPU's output ports is automatically transmitted to the port data arrays of destination LPUs, while input data from source LPUs is automatically inserted into those elements associated with the LPU's input ports (Tab. 2.4).

Graded Potential Ports			Spiking Ports		
Port	Array Index	Array Data	Port	Array Index	Array Data
`/lam[0]`	0	0.71	`/lam[3]`	0	1
`/lam[1]`	1	0.83	`/lam[4]`	1	0
`/lam[2]`	2	0.52	`/lam[5]`	2	1

Table 2.4: Example of input and output data mapped to and from data arrays by the `GPUPortMapper` class for the ports comprised by interface 0 in the pattern described in Tab. 2.2 and Tab. 2.3.

In addition to the classes that represent LPUs and inter-LPU connectivity patterns, Neurokernel provides an emulation manager class called `Manager` that provides services for configuring LPU classes, connecting them with specified connectivity patterns, and determining how to route data between LPUs based upon those patterns. The manager class hides the process and communication management performed by OpenMPI so as to obviate the need for model designers to directly interact with the traditional MPI job launching interface. Once an emulation has been fully configured via the manager class, it may be executed for a specified interval of time or for a specified number of steps.

Apart from the API requirements discussed above, Neurokernel currently places no explicit restrictions upon an LPU model's internal implementation, how it interacts with avail-

able GPUs, how LPUs record their output, or the topology of interconnections between different LPUs. An LPU implementation called Neurodriver that provides built-in support for several commonly used neuron and synapse models is available (§ 2.2.6), however. Compatible LPUs and inter-LPU patterns may be arbitrarily composed to construct subsystems (Fig. 2.5). It should be noted that the current LPU interface is not intended to be final; we anticipate its gradual extension to support communication between models that more accurately account for the range of interactions that occur within the fruit fly's brain.

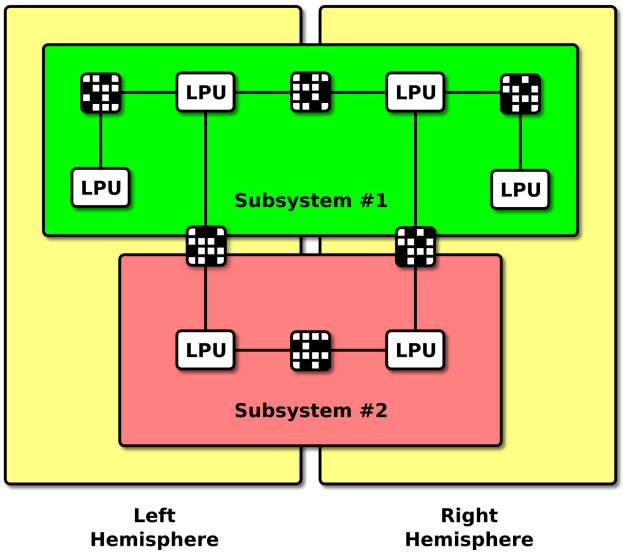


Figure 2.5: Neurokernel brain modeling architectural hierarchy. Independently developed LPUs and connectivity patterns may be composed into subsystems (red, green) which may in turn be connected to other subsystems to construct a model of the whole brain (yellow).

Communication between LPU instances in a running Neurokernel emulation is performed using MPI by means of the mpi4py Python bindings [28] to enable brain emulations to take advantage of multiple GPUs hosted either on single computer or a computer cluster. Neurokernel uses OpenMPI [42] to provide accelerated access between GPUs that support NVIDIA's GPUDirect Peer-to-Peer technology [100, 101] when the source and destination memory locations of an MPI data transfer are both in GPU memory. Neurokernel-based

models are executed in a bulk synchronous fashion; each LPU's execution step is executed asynchronously relative to other LPUs' execution steps, but data associated with the output ports of all connected LPUs must be propagated to their respective destinations before those LPUs can proceed to the next execution step. Since data is transmitted between connected LPUs at every execution step, the output ports of all LPUs are effectively sampled at the same rate. Individual LPUs may perform internal computations at a finer time resolution, provided that they update their output port data arrays at the end of each invocation of their `run_step()` methods.

2.2.5 Using the Neurokernel API

This section illustrates how to use the Neurokernel classes described in § 2.2.4 to construct and execute an emulation consisting of multiple connected LPUs. The section assumes that Neurokernel and its relevant dependencies (including OpenMPI) have already been installed on a system containing multiple GPUs. First, we import several required Python modules; the `mpi_relaunch` module provided by Neurokernel sets up the MPI environment required to enable communication between LPUs.

```
import neurokernel.mpi_relaunch

from mpi4py import MPI
import numpy as np
import pycuda.gpuarray as gpuarray

from neurokernel.mpi import setup_logger
from neurokernel.core_gpu import Module, Manager
from neurokernel.pattern import Pattern
from neurokernel.plsel import Selector, SelectorMethods
```

Next, we create a subclass of `Module` whose `run_step()` method accesses the class instance's port data arrays; the example below generates random graded potential and spiking output port data.

```
class MyModule(Module):
    """
    Example of derived module class.
```

```
    """

    def run_step(self):

        # Call the run_step() method of the parent class (Module):
        super(MyModule, self).run_step()

        # Log input graded potential data:
        self.log_info('input gpot port data: '+\
                      str(self.pm['gpot'][self.in_gpot_ports]))

        # Log input spike data:
        self.log_info('input spike port data: '+\
                      str(self.pm['spike'][self.in_spike_ports]))

        # Output random graded potential data:
        out_gpot_data = \
            gpuarray.to_gpu(np.random.rand(len(self.out_gpot_ports)))
        self.pm['gpot'][self.out_gpot_ports] = out_gpot_data
        self.log_info('output gpot port data: '+str(out_gpot_data))

        # Output spikes to randomly selected output ports:
        out_spike_data = \
            gpuarray.to_gpu(np.random.randint(0, 2,
                                              len(self.out_spike_ports)))
        self.pm['spike'][self.out_spike_ports] = out_spike_data
        self.log_info('output spike port data: '+str(out_spike_data))
```

The data arrays associated with an LPU's ports may be accessed using their path-like identifiers via two instances of the `GPUPortMapper` class stored in the `self.pm` attribute. Updated data associated with output ports is propagated to the relevant destination LPUs by Neurokernel before the next iteration of the emulation's execution.

To connect two LPUs, we specify the ports to be exposed by each LPU using path-like selectors. The example below describes the interfaces for two LPUs that each expose two graded potential input ports, two graded potential output ports, two spiking input ports, and two spiking output ports. `Selector` is a convenience class that provides methods and overloaded operators for combining and manipulating sets of validated port identifiers. For example, `Selector('/a/in/gpot[0:2]')` corresponds to the set of two input graded potential port identifiers `/a/in/gpot[0]` and `/a/in/gpot[1]`. Additional methods for manipulating port identifiers are provided by the `SelectorMethods` class.

```
# Define input graded potential, output graded potential,
```

```
# input spiking, and output spiking ports for LPUS 'a' and 'b':
m1_sel_in_gpot = Selector('/a/in/gpot[0:2]')
m1_sel_out_gpot = Selector('/a/out/gpot[0:2]')
m1_sel_in_spike = Selector('/a/in/spike[0:2]')
m1_sel_out_spike = Selector('/a/out/spike[0:2]')

m2_sel_in_gpot = Selector('/b/in/gpot[0:2]')
m2_sel_out_gpot = Selector('/b/out/gpot[0:2]')
m2_sel_in_spike = Selector('/b/in/spike[0:2]')
m2_sel_out_spike = Selector('/b/out/spike[0:2]')

# Combine selectors to obtain sets of all input, output,
# graded potential, and spiking ports for the two LPUs:
m1_sel = m1_sel_in_gpot+m1_sel_out_gpot+\
         m1_sel_in_spike+m1_sel_out_spike
m1_sel_in = m1_sel_in_gpot+m1_sel_in_spike
m1_sel_out = m1_sel_out_gpot+m1_sel_out_spike
m1_sel_gpot = m1_sel_in_gpot+m1_sel_out_gpot
m1_sel_spike = m1_sel_in_spike+m1_sel_out_spike

m2_sel = m2_sel_in_gpot+m2_sel_out_gpot+\
         m2_sel_in_spike+m2_sel_out_spike
m2_sel_in = m2_sel_in_gpot+m2_sel_in_spike
m2_sel_out = m2_sel_out_gpot+m2_sel_out_spike
m2_sel_gpot = m2_sel_in_gpot+m2_sel_out_gpot
m2_sel_spike = m2_sel_in_spike+m2_sel_out_spike

# Count the number of graded potential and
# spiking ports exposed by each LPU:
N1_gpot = SelectorMethods.count_ports(m1_sel_gpot)
N1_spike = SelectorMethods.count_ports(m1_sel_spike)

N2_gpot = SelectorMethods.count_ports(m2_sel_gpot)
N2_spike = SelectorMethods.count_ports(m2_sel_spike)
```

Using the above LPU interface data, we construct an inter-LPU connectivity pattern by instantiating the `Pattern` class, setting its port input/output and transmission types, and populating it with connections:

```
# Initialize connectivity pattern that can link
# ports in m1_sel with ports in m2_sel:
pat12 = Pattern(m1_sel, m2_sel)

# Set the input/output and transmission type attributes of each port in the
# pattern's two interfaces:
pat12.interface[m1_sel_out_gpot] = [0, 'in', 'gpot']
pat12.interface[m1_sel_in_gpot] = [0, 'out', 'gpot']
pat12.interface[m1_sel_out_spike] = [0, 'in', 'spike']
pat12.interface[m1_sel_in_spike] = [0, 'out', 'spike']
pat12.interface[m2_sel_in_gpot] = [1, 'out', 'gpot']
pat12.interface[m2_sel_out_gpot] = [1, 'in', 'gpot']
```

```
pat12.interface[m2_sel_in_spike] = [1, 'out', 'spike']
pat12.interface[m2_sel_out_spike] = [1, 'in', 'spike']

# Create the connections between ports:
pat12['/a/out/gpot[0]', '/b/in/gpot[0]'] = 1
pat12['/a/out/gpot[1]', '/b/in/gpot[1]'] = 1
pat12['/b/out/gpot[0]', '/a/in/gpot[0]'] = 1
pat12['/b/out/gpot[1]', '/a/in/gpot[1]'] = 1
pat12['/a/out/spike[0]', '/b/in/spike[0]'] = 1
pat12['/a/out/spike[1]', '/b/in/spike[1]'] = 1
pat12['/b/out/spike[0]', '/a/in/spike[0]'] = 1
pat12['/b/out/spike[1]', '/a/in/spike[1]'] = 1
```

Certain types of patterns can also be constructed using several class methods provided by the `Pattern` class that afford faster performance than the above for selectors containing large numbers of ports. For example, the above pattern can also be constructed as follows:

```
pat12 = Pattern.from_concat(m1_sel, m2_sel,
                            from_sel=m1_sel_out+m2_sel_out,
                            to_sel=m2_sel_in+m1_sel_in,
                            gpot_sel=m1_sel_gpot+m2_sel_gpot,
                            spike_sel=m1_sel_spike+m2_sel_spike, data=1)
```

We can then pass the defined LPU class and the parameters to be used during instantiation to a `Manager` class instance that connects them together with the above pattern. The `setup_logger` function may be used to enable output of log messages generated during execution:

```
logger = setup_logger(screen=True, file_name='neurokernel.log',
                      mpi_comm=MPI.COMM_WORLD, multiline=True)

man = Manager()

m1_id = 'm1   '
man.add(MyModule, m1_id, m1_sel, m1_sel_in, m1_sel_out,
        m1_sel_gpot, m1_sel_spike,
        np.zeros(N1_gpot, dtype=np.double),
        np.zeros(N1_spike, dtype=int),
        device=0)
m2_id = 'm2   '
man.add(MyModule, m2_id, m2_sel, m2_sel_in, m2_sel_out,
        m2_sel_gpot, m2_sel_spike,
        np.zeros(N2_gpot, dtype=np.double),
        np.zeros(N2_spike, dtype=int),
        device=1)
man.connect(m1_id, m2_id, pat12, 0, 1)
```

After all LPUs and connectivity patterns are provided to the manager, the emulation may be executed for a specified number of steps as follows. Neurokernel uses the dynamic process creation feature of MPI-2 supported by OpenMPI to automatically spawn as many MPI processes are needed to run the emulation. Each LPU class and its associated instantiation parameters are transmitted to a spawned process where the class is instantiated and its run loop executed.

```
# Compute number of execution steps given emulation duration
# and time step (both in seconds):
duration = 10.0
dt = 1e-2
steps = int(duration/dt)

man.spawn()
man.start(steps)
man.wait()
```

2.2.6 Neurodriver - a Configurable LPU Implementation

To facilitate the use of Neurokernel for developing LPU models, we developed a package called Neurodriver that enables declarative construction of LPU models composed of supported neuron and synapse models. Neurodriver contains a subclass of the `Module` class described in § 2.2.4 that can execute a property graph describing an LPU's circuit; this graph may be loaded from a GEXF file or explicitly defined in Python as a NetworkX data structure [53]. Neurodriver currently supports the Leaky Integrate-and-Fire, Morris-Lecar, and Hodgkin-Huxley neuron models, as well as alpha function and conductance-based synapses. A stochastic model of the photoreceptors in the fly retina has also been developed for Neurodriver as part of the retina/lamina model described in § 2.3.1. Neurons may be configured to transmit spikes (if they produce action potentials) or membrane potential values to synapses.

Neurodriver provides a plugin mechanism that may be used to add support for new models by subclassing base neuron and synapse Python classes that provide a uniform framework for invoking the GPU-based numerical equations associated with a model. Once

a new model class has been added to Neurodriver's Python path, Neurodriver can execute LPU graphs that reference the model.

2.3 Results

To evaluate Neurokernel's ability to facilitate interfacing of functional brain modules that can be executed on GPUs, we employed Neurokernel's programming model (§ 2.2.3) to interconnect independently developed LPUs in the fruit fly's early visual system to provide insights into the representation and processing of the visual field by the cascaded LPUs. We also evaluated Neurokernel's scaling of communication performance in simple configurations of the architecture parameterized by numbers of ports and LPUs and assessed Neurodriver's scaling of execution performance for neuropil-scale circuits of point neuron and conductance-based synapse models.

2.3.1 Integration of Independently Developed LPU Models

The integrated early visual system model we considered consists of models of the fruit fly's retina and lamina. The retina model comprises a hexagonal array of 721 ommatidia, each of which contains 6 photoreceptor neurons. The photoreceptor model employs a stochastic model of how light input (photons) produce a membrane potential output. Each photoreceptor consists of 30,000 microvilli modeled by 15 equations per microvillus, a photon absorption model, and a model of how the aggregate microvilli contributions produce the photoreceptor's membrane potential [73]; the entire retina model employs a total of about 1.95 billion equations. The lamina model consists of 4,326 Morris-Lecar neurons configured to not emit action potentials and about 50,000 conductance-based inhibitory synapses expressing histamine [80]. The LPUs were linked by 4,326 feed-forward connections from the retina to the lamina; the connections from the retina to the lamina were configured to map output ports exposed by the retina to input ports in the lamina based upon the neural superposition rule [70]. The source code for the visual system model is available at

`http://github.com/neurokernel/retina-lamina`

The combined retina and lamina models were executed on up to 4 Tesla K20Xm NVIDIA GPUs with an 8 second natural video scene provided as input to the retinal model's photoreceptors. The computed membrane potentials of specific photoreceptors in each retinal ommatidium and of select neurons in each cartridge of the lamina were recorded (Fig. 2.6). In this example, the observed R1 photoreceptor outputs demonstrate the preservation of visual information received from the retina by the lamina LPU. The L1 and L2 lamina neuron outputs demonstrate the signal inversion taking place in the two pathways shaping the motion detection circuitry of the fly. These initial results illustrate how Neurokernel's API enables LPU model designers to treat their models as neurocomputing modules that may be combined into complex information processing pipelines whose input/output properties may be obtained and evaluated.

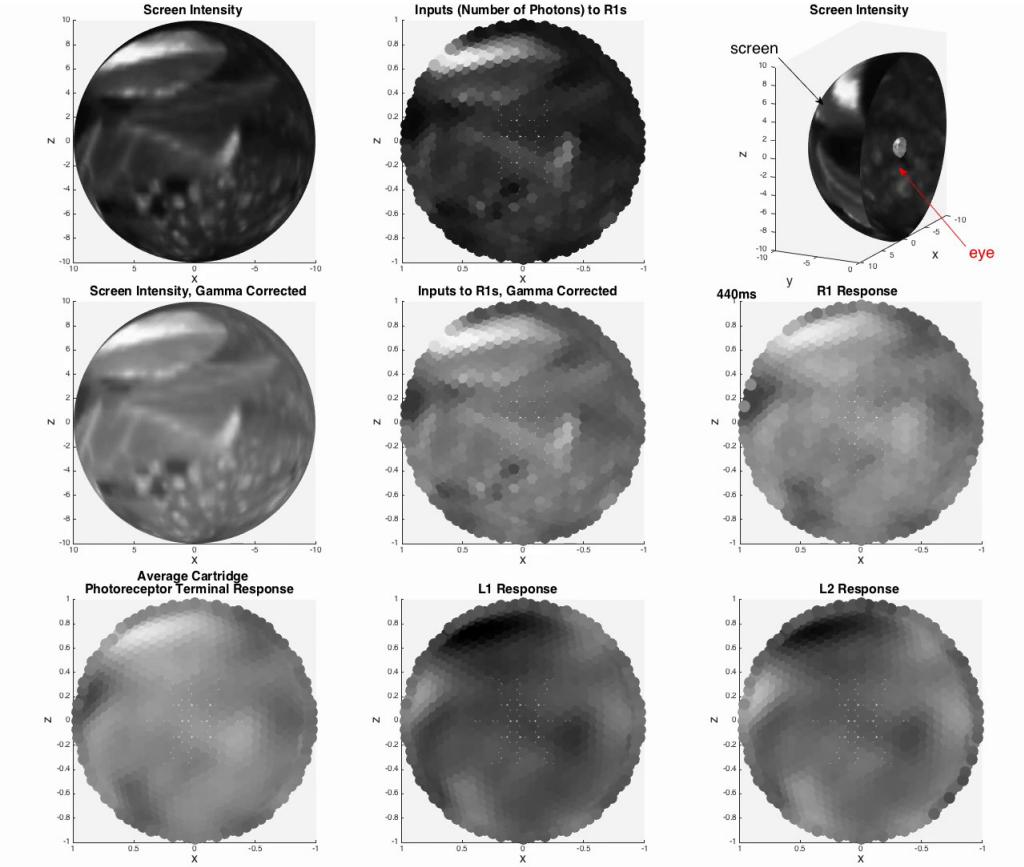


Figure 2.6: Example of natural input to the combined retina/lamina model. The hexagonal tiling depicts the array of ommatidia in the retina and the corresponding retinotopic cartridges in the lamina. Outputs of select photoreceptors in the retina (R1) that are fed to neurons in the lamina and outputs of specific neurons in the lamina (L1, L2) are also depicted.

2.3.2 Module Communication Performance

We compared the performance of emulations in which port data stored in GPU memory is copied to and from host memory for traditional network-based transmission by OpenMPI to that of emulations in which port data stored in GPU memory is directly passed to OpenMPI's communication functions. The latter functions enabled OpenMPI to use NVIDIA's GPUDirect Peer-to-Peer technology to perform accelerated transmission of data between GPUs whose hardware supports the technology by bypassing the host system's CPU and

memory [101]. All tests discussed below were performed on a host containing 2 Intel Xeon 6-core E5-2620 CPUs, 32 Gb of RAM, and 4 NVIDIA Tesla K20Xm GPUs running Ubuntu Linux 14.04, NVIDIA CUDA 7.0, and OpenMPI 1.8.5 built with CUDA support. The code required to obtain the benchmarks is included in the Neurokernel source code available online.

2.3.2.1 Scaling over Number of LPU Output Ports

To evaluate how well inter-LPU communication scales over the number of ports exposed by an LPU on a multi-GPU machine, we constructed and ran emulations comprising multiple connected instances of an LPU class with an empty `run_step()` method (see § 2.2.4) and measured (1) the average time taken per execution step to synchronize the data exposed by the output ports in each of two connected LPUs with their respective destination input ports; (2) the average throughput per execution step (in terms of number of port data elements transmitted per second) of the synchronization, where each port is stored either as a 32-bit integer or double-precision floating point number (both of which occupy 8 bytes).

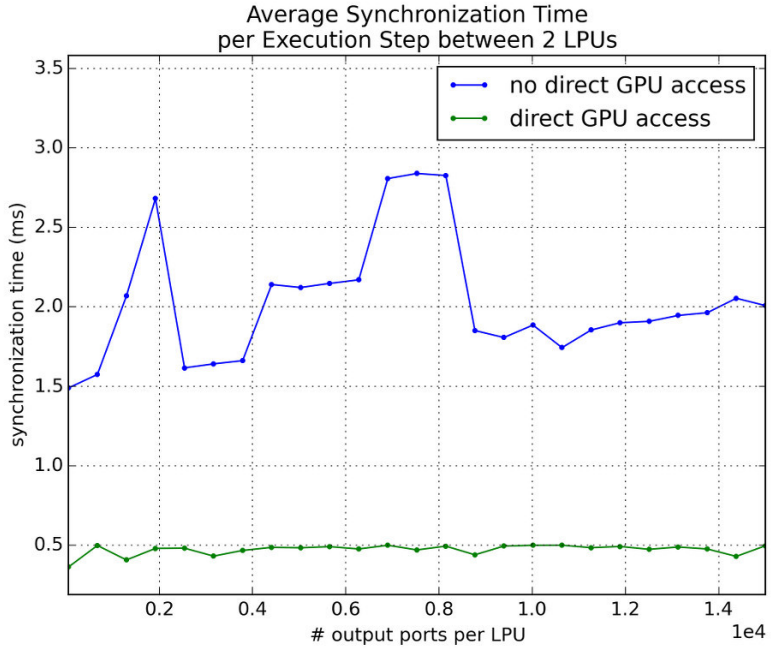
We initially examined how the above performance metrics scaled over the number of output ports exposed by each LPU in a 2-LPU emulation and over the number of LPUs in an emulation where each LPU is connected to every other LPU and the total number of output ports exposed by each LPU is fixed. We compared the performance for scenarios where data in GPU memory is directly exposed to OpenMPI to that for scenarios where the data is copied to the host memory prior to transmission; the former scenarios enabled OpenMPI to accelerate data transmission between GPUs using NVIDIA's GPUDirect Peer-to-Peer technology. The metrics for each set of parameters were averaged over 3 trials; the emulation was executed for 500 steps during each trial.

The scaling of performance over number of ports depicted in Fig. 2.7 clearly illustrate the ability of GPU-to-GPU communication between locally hosted GPUs to ensure that increasing the number of ports exposed by an LPU does not increase model execution time

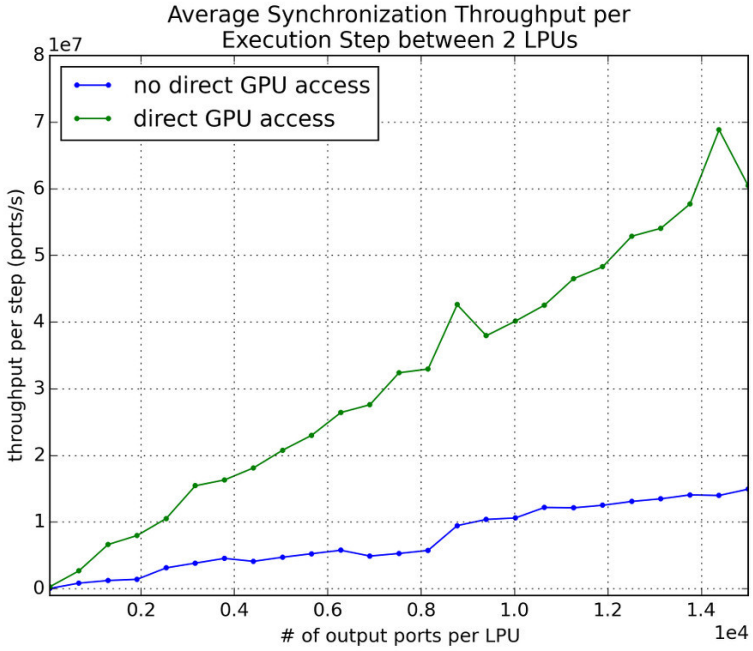
for numbers of ports similar to the numbers of neurons in actual LPUs. We also observed noticeable speedups in synchronization time for scenarios using more than 2 GPUs as the number of ports exposed by each LPU is increased (Fig. 2.8). As the number of GPUs in use reached the maximum available in our test system, overall speedup diminished; this appears to be due to gradual saturation of the host's PCI bus.

2.3.2.2 Scaling over Number of LPUs

Current research on the fruit fly brain is mainly focused on LPUs in the fly's central complex and olfactory and vision systems. Since the interplay between these systems will be key to increasing understanding of multisensory integration and how sensory data might inform behavior mediated by the central complex, we examined how well Neurokernel's communication mechanism performs in scenarios where LPUs from these three systems are successively added to a multi-LPU emulation. Starting with the pair of LPUs with the largest number of inter-LPU connections, we sorted the 19 LPUs in the above three systems in decreasing order of the number of connections contributed with the addition of each successive LPU and measured the average speedup in synchronization time per execution step due to direct GPU-to-GPU data. The number of connections for each LPU was based upon estimates from a mesoscopic reconstruction of the fruit fly connectome; these numbers appear in Document S2 of the supplement of [127]. The LPU class instances were designed to send and receive data only; no other computation was performed or benchmarked during execution. To amortize inter-LPU transmission costs, the LPUs were partitioned across the available GPUs using the METIS graph partitioning package [67] to minimize the total edge cut. The speedup afforded by direct GPU-to-GPU data (Fig. 2.9) illustrates that current GPU technology can readily power multi-LPU models based upon currently available connectome data.



(a) Average synchronization time per execution step



(b) Average synchronization throughput (in number of ports per unit time) per execution step.

Figure 2.7: Synchronization performance for an emulation comprising 2 interconnected LPUs accessing 2 different GPUs on the same host scaled over number of output ports exposed by each LPU. The number of output ports was varied over 25 equally spaced values between 50 and 15,000.

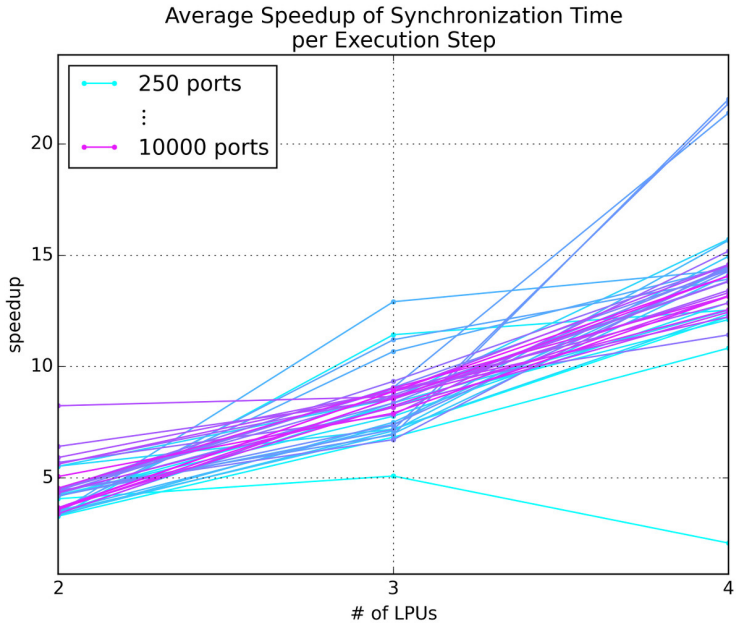


Figure 2.8: Speedup of average synchronization time per execution step for an emulation scaled over number of LPUs, where each LPU is mapped to a single GPU. The total number of output ports exposed by each LPU was varied between 250 and 10,000 at 250 port intervals.

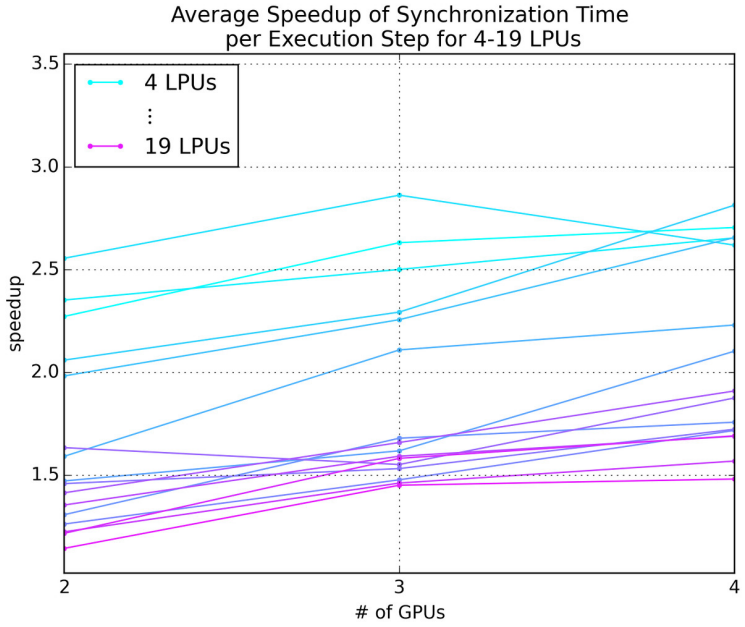


Figure 2.9: Synchronization performance for an emulation comprising between 4 and 19 interconnected LPUs selected from the central complex, olfactory, and vision systems partitioned over 2 to 4 GPUs on the same host.

2.3.3 Neurodriver Performance

To assess whether Neurodriver affords reasonable performance when executing neural circuits containing numbers of neurons and synapses comparable to those found in actual neuropils, we compared its performance to that of Brian2GeNN, a package that enables the Brian simulator [52] to use the GeNN code generation framework [144] to accelerate execution of spiking neuronal network circuits defined using Brian's Python interface using a GPU. Brian2GeNN was used for this comparison rather than other GPU-based simulators (§ 2.4) because it is the only available GPU-based neural simulator other than Neurodriver under active development as of 2016 that has a functioning Python interface and can support the same neuron and synapse models as Neurodriver. The code required to run these benchmarks is available online at `http://github.com/neurokernel/neurodriver-benchmark`

For both Neurodriver and Brian2GeNN, we constructed networks of Leaky Integrate-and-Fire neurons randomly connected by alpha function synapses. A constant current was injected into a subset of neurons in each implementation to elicit spiking activity. Both implementations were run for 3 seconds at a time resolution of $1 \cdot 10^4$ seconds on the same host system described in § 2.3.2. The number of neurons in each network was scaled from 100 to 12000 neurons and from 2500 and $1.6 \cdot 10^6$ synapses, respectively. Fig. 2.10 depicts the average times over 3 trials for each network size plotted with respect to number of neurons and number of synapses.

Given the differences between the respective designs of Neurodriver and Brian2GeNN, several considerations were taken to make the comparison as fair as possible:

- The execution time of the run loop of Neurodriver's LPU class was measured on the spawned MPI process after class instantiation and initialization of all internal variables. Since Brian2GeNN must currently perform code generation during every simulation run, its execution time was measured to include invocation of the simulation run mechanism and the time taken by code generation.

- A constant input signal was provided to a subset of the neurons in both the Neurodriver and Brian2GeNN circuit implementations. The input was copied into the requisite GPU variables in both implementations at every execution step.
- Since Brian2GeNN transfers the output produced by the GeNN-generated code back to the parent Python session, spikes generated at every run step of the Neurodriver implementation were also copied back to host memory after every execution step.

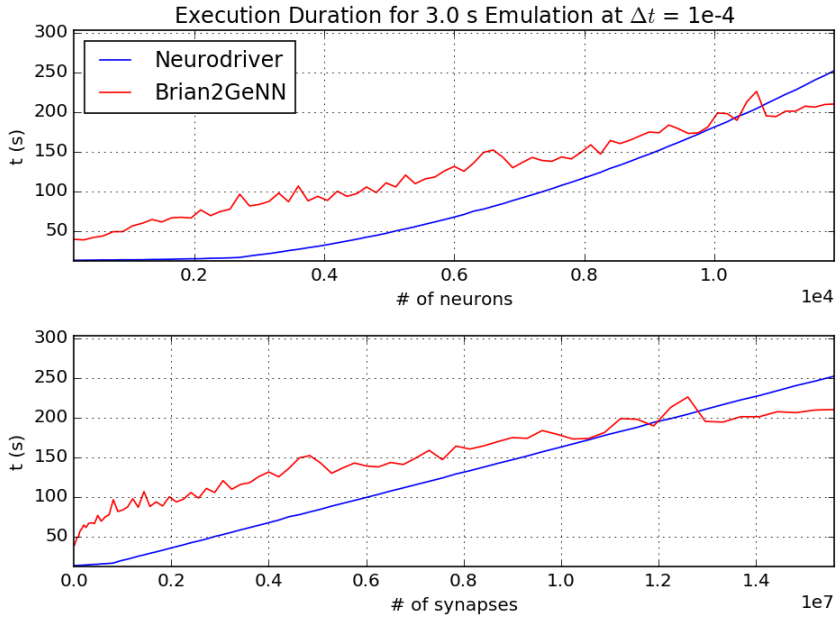


Figure 2.10: Comparison of performance of Neurodriver and Brian2GeNN when simulating 3 seconds of activity produced by randomly connected spiking neural networks containing between 100 and 12000 neurons and between 2500 and $1.6 \cdot 10^6$ synapses in response to a constant input to a subset of the neurons. The execution times were averaged over 3 trials per network size. Both plots depict the same average execution times with respect to numbers of neurons and synapses, respectively.

The above results indicate that Neurodriver exhibits superior performance compared to Brian2GeNN for networks containing up to about $1.05 \cdot 10^4$ neurons and $1.3 \cdot 10^7$ synapses. For larger networks, the efficiency of the code generated by GeNN enables Brian2GeNN to achieve better scaling of execution times with respect to network size than Neurodriver.

Given that only a few of the neuropils in the fruit fly contain numbers of components greater than the above range [22, 80], these results demonstrate that Neurodriver can provide competitive performance for multi-LPU emulations that target most of the fly brain neuropils. Although Brian2GeNN's performance scales better than that of Neurodriver with respect to network size, it is worth noting that Neurodriver's use of Neurokernel's communication API enables it to be used in emulations that require multiple GPUs and/or require interaction between neurons in different LPUs that do not emit spikes. We address the possibility of employing approaches similar to Brian2GeNN in future releases of Neurokernel in § 5.2.2.

2.4 Related Work

2.4.1 General Purpose Neuronal Network Simulators

Computational neuroscientists can currently choose from an array of actively developed neuronal network simulation software to power their models. GENESIS [11], NEURON [19], and MOOSE [116] enable construction of neural circuit models using components such as multicompartmental neuron models capable of simulating high levels of biophysical detail. NEST provides well-tested support for over 50 published neuron models and over 10 synapse models [43]. NEURON has been used to simulate randomly connected networks of spiking point neurons up to $4{\cdot}10^6$ neurons with $4{\cdot}10^{10}$ connections on a Blue Gene/P supercomputer ($3{\cdot}10^5$ cores, 144 TB RAM) with a performance of about 0.005 of real-time speed [57]; NEST has been used to simulate spiking networks containing about 5 times as large [56] on the K supercomputer ($7 \cdot 10^5$ cores, 1.4 PB RAM) with similar execution speeds. Since these packages do not currently support the use of GPUs, their performance on less powerful desktop computer systems is considerably more limited, however. Both GENESIS and NEURON provide graphical user interfaces and their own scripting languages (SLI and HOC, respectively) for defining neuron/synapse models and networks, although they and MOOSE all currently provide Python interfaces.

The Brian simulator [52, 130] provides neuroscientists with the ability to define a wide range of neuron and synapse models in terms of actual differential equations while largely eliminating the need to explicitly address numerical considerations associated with implementing such models. Although the project does leverage vectorized operations and a code generation backend to accelerate model execution [130], it prioritizes programmability over performance through its use of Python rather than a compiled language such as C++ and its focus upon easy model specification and the flexibility to mathematically define arbitrary models. An extension to Brian under development as of 2016 called Brian2GeNN aims to utilize the GeNN code generation framework [144] to provide GPU support; GeNN is further discussed in § 2.4.2.

The PyNN project [30] also focuses upon ease of use by providing a high-level Python interface to a range of commonly used neuron and synapse models while leaving the efficient implementation of the models to other simulation engines; PyNN currently supports the use of NEURON, NEST, PCSIM, and Brian as backends. PyNN's object oriented interface defines classes corresponding to populations of neurons, projections comprising synaptic connections between populations, and connectors that encapsulate the algorithm used to create a projection.

While the above projects have considerably increased the ease of defining neuronal circuit models, they do not provide a straightforward means of supporting collaborative brain model development because of the lack of a programming model that enables independently developed circuits to be interconnected. Most of these simulators support the notions of neuron populations and connectivity projections from one population of neurons to another. Projects alone, however, are insufficient for conceptually modularizing the brain into functional circuits that may be simultaneously investigated by different researchers because they do not define how modeling components in one circuit are to communicate with those in another circuit independently of the respective components' identities. GPU support in these projects is also nonexistent or under development as of 2016.

2.4.2 GPU-based Neuronal Network Simulators

The use of GPU parallel computing technology by neuronal network simulators has significantly advanced the performance available to researchers without access to large-scale supercomputers. Although most general-purpose neural simulators (§ 2.4.1) have yet to incorporate support for GPUs, an increasing range of GPU-based simulators are currently available.

NeMo is a spiking neural network simulator capable of simulating networks comprising 10^5 neurons and 10^7 synapses on a single GPU at real-time execution speeds using a novel just-in-time spike delivery technique to reduce memory bandwidth requirements [38, 39]. NeMo supports user-configurable synaptic plasticity and conduction delays, but does not implement any models other than the Izhikevich neuron model [63]. Neurons that do not produce spikes are also not supported.

CARLSim 3 [97, 117, 10] is a simulator that supports accelerated execution of networks of Izhikevich neurons on both commodity x86 CPUs and NVIDIA GPUs. Implemented in C/C++ and CUDA, it supports a programming interface modeled on the design of PyNN [30] to ease its use by model developers. Unique features provided by CARLsim 3 include an implementation of spike-timing dependent plasticity (STDP) that supports dopaminergic neuromodulation and an interface to the ECJ automatic parameter tuning system that can be used to tune almost any parameter of a simulated network. The simulator has been used to implement models of visual processing, neuromodulation, synaptic plasticity, and attention comprising up to 10^5 neurons and 10^6 synapses at 50% of real-time performance. CARLSim 3 does not provide a Python interface and does not yet support multiple GPUs.

The Nengo [6] simulator enables construction of neuronal networks that can model cognitive processes described in terms of the Neural Engineering Framework (NEF) [36]. It supports Python and can utilize GPUs to accelerate performance by means of PyOpenCL [71]. Nengo has been used to realize Spaun [37], a functional brain model comprising $2.5 \cdot 10^6$ spiking neurons that can perform a range of cognitive tasks. Neural circuit models that do

not utilize the NEF principle (which has yet to be empirically verified) cannot be easily implemented in Nengo, however.

GeNN is a simulator package that generates and compiles automatically optimized GPU code to obtain significant execution acceleration for spiking neural networks comprising up to 10^6 point neurons on a single GPU [143]. As GeNN itself does not possess a Python interface, work is underway to combine the flexibility of Brian with GeNN's support for GPU-based simulations by means of a backend called Brian2GeNN [144].

Despite the high level of parallelism they provide, the relatively limited memory bandwidth of GPUs poses challenges in the effective use of multiple GPUs to execute models that involve high levels of coupling between model components. Spiking neural network simulators can exploit spike sparsity to mitigate the costs of inter-GPU data transfer. Neocortical Simulator (NCS) 6 [59], for example, supports simulation of large-scale networks of Izhikevich neurons or hybrid spiking neurons whose membrane potential follows a Leaky Integrate-and-Fire model after crossing a threshold and whose subthreshold dynamics follow the Hodgkin-Huxley model. NCS6 is capable of executing up to 10^6 neurons of the above types connected by up to $5 \cdot 10^7$ synapses on multiple CPUs or NVIDIA GPUs. Similarly, HRLSim [92] can execute networks comprising $1.1 \cdot 10^5$ Leaky Integrate-and-Fire or Izhikevich neurons and $1.1 \cdot 10^6$ synapses in real-time using multiple NVIDIA GPUs, although its source code does not appear to be publicly available.

Given that significant portions of the fly brain (e.g., the lamina and medulla in the fly vision system) consist of neurons that have not been observed to emit action potentials, simulators that only support networks of spiking neurons sacrifice a significant level of biological plausibility to achieve superficially impressive execution performance. In contrast to the above simulators that focus on optimizing performance for networks composed entirely of neurons that communicate via spikes, the Myriad simulator explicitly targets densely integrated biophysical networks of neurons that require multiple state updates at every simulation step [121, 120, 122]. To mitigate the cost associated with large numbers of

state updates, Myriad employs a novel code generation approach that translates neuronal networks defined using a Python interface into networks of low-level computational elements that can be efficiently parallelized on a GPU. The developers of Myriad plan to release it as open source in the future.

2.4.3 Neuromorphic Simulation Platforms

Neuromorphic platforms whose design is directly inspired by the brain have the potential to execute large-scale neural circuit models at speeds that significantly exceed those achievable with traditional von Neumann computer architectures while consuming significantly less power than other parallel computing hardware platforms.

SpiNNaker and FACETS/BrainScaleS are scalable neuromorphic platforms designed to eventually provide the performance required to emulate the human brain. SpiNNaker combines multiple independent digital ARM microprocessors with a customized on-chip interconnect [115]. Inspired by the features of biological neural circuits, SpiNNaker provides a model of natively parallel model of computation that supports event-driven processing, memory access by any processor without notification or synchronization, and the ability to reconfigure the hardware while it is running. SpiNNaker has been used to implement both spiking neural network models composed of Izhikevich neurons and artificial neural network algorithms such as the multilayer perceptron (MLP). FACETS/BrainScaleS takes a different architectural approach that combines local analog neuron and synapse computations with asynchronous spike event communication; each FACETS hardware subsystem comprises a custom mixed-signal ASIC for emulation of spiking neurons and synapses combined with a digital ASIC for communication [14]. It can support networks comprising up to $2 \cdot 10^5$ programmable spiking neurons and $4.5 \cdot 10^7$ synapses. PyNN support for both SpiNNaker and FACETS/BrainScaleS is available.

Neurogrid is neuromorphic platform capable of real-time simulation of $1 \cdot 10^6$ spiking neurons and $1 \cdot 10^9$ synapses with power requirements of only a few watts [8]. Neurogrid

uses shared circuits to emulate most components of a neural circuit to maximize the number of synaptic connections it can support. Like FACETS, Neurogrid employs a mixed-signal design; all circuits other than axonal arbors are implemented in analog, while spike events are transmitted digitally. In addition to the above hardware, Neurogrid provides a software stack with a GUI interface and support for model specification in Python.

TrueNorth is a low-power chip for running massive networks of spiking neurons. Comprising 4096 neurosynaptic cores that each contain a network of input and output lines connected by programmable synapses, a single TrueNorth chip is capable of running networks of 10^6 spiking neurons and $2.56 \cdot 10^8$ synapses in real-time [90]. Neurons and synapses may be individually configured to obtain a range of different behaviors. TrueNorth's architecture can be tiled in two dimensions to construct systems that support even larger networks. The architecture has been used as a substrate for computationally intensive tasks such as convolutional networks and machine learning algorithms. Networks designed to run on TrueNorth currently must be specified using a compositional language designed for the architecture.

In contrast to the above platforms that are built upon customized hardware, NeuroFlow is a general-purpose spiking neural network simulator that can run on commercially available FPGAs [21]. It currently supports common point neuron models, exponential and alpha function synapses, and STDP. To obviate the need for specialized knowledge regarding the use of FPGA hardware, NeuroFlow provides a Python API that uses PyNN. Using a single FPGA, NeuroFlow can achieve execution speeds almost 3 times as great as those of GPU-based simulators such as CARLSim 3 and almost 34 times as great as those of CPU-based simulators such as NEST for networks containing up to $5.9 \cdot 10^5$ neurons.

Despite the exciting possibilities afforded by hardware with non-von Neumann architectures, uncertainty regarding the appropriate computational paradigm for modeling the brain implies that more progress must be made on exploring models of brain processing before we can design an optimal hardware platform for brain emulation. Neurokernel's cur-

rent design is therefore predicated upon the use of technologies such as commodity GPUs and Python that afford maximal model construction flexibility to a wide audience with reasonable computational hardware performance. As neuromorphic technology matures and becomes available to the wider neurocomputing community, we anticipate extending Neurokernel's compute plane to support the use of such hardware alongside and eventually in the place of GPU technology to power whole brain emulations.

2.4.4 Simulator Interfacing Packages

Although the increasingly wide array of neuronal network simulators actively in use do afford a range of unique features, there has been only modest interest in run-time simulator interoperability that can enable models implemented for different simulators to interact during run-time. MUSIC is an API for run-time data exchange between neural circuit models executed on different simulators [33]. MUSIC's API associates communication ports with data sources or sinks in the connected models; these ports support transmission of either spike data or continuous values such as membrane potentials. A key advantage of MUSIC's design is that it eliminates the need for communication handshakes between connected simulators, which can run at different time resolutions if so desired. To use MUSIC, existing neural simulators must use MPI and be explicitly modified to support MUSIC's API; as of the present, communication of spike events between different simulators using MUSIC has only been implemented for NEST [43] and MOOSE [116]. In contrast to Neurokernel's port labeling syntax, MUSIC does not provide any port labeling scheme to facilitate management of large numbers of ports, nor does it define a programming model for encapsulating models with an implementation-independent communication interface. Given that it was designed prior to the recent rise in interest in using multiple GPUs for large simulations and the concomitant development of technologies for accelerated inter-GPU data transfers such as NVIDIA's GPUDirect and CUDA-enabled OpenMPI, MUSIC's communication scheme also does not address the performance questions that arise due to the limited GPU memory

transfer bandwidth. As of 2016, MUSIC no longer seems to be under development and does not appear to be in active use by any popular simulation platforms.

The PCSIM spiking neural network simulator was one of the first general-purpose simulation packages to provide a primary Python interface despite being implemented in a compiled language [106]. A unique feature provided by PCSIM is its support for encapsulating network elements implemented using other simulators such as Brian (provided that the implemented elements can exchange data with PCSIM's Python API). Custom network elements must expose input and output ports that may be connected to native PCSIM modeling components; ports may only be labeled using integer ranges, however. No longer under development since 2010, PCSIM does not support the use of GPUs.

2.4.5 Whole Brain Simulation Projects

As of 2016, no comprehensive computational model of the entire fruit fly brain predicated upon connectome data exists. There are, however, several ongoing efforts to develop brain or nervous system models for other model organisms with similar complexity.

The OpenWorm Project aims to develop a whole-body biophysical simulation of the nematode *C. elegans* to examine hypotheses as to how its behaviors arise from its biological architecture [133]. It capitalizes upon on the extremely small number of neurons in the worm's nervous system and the full reconstruction of its connectome [139]. The project comprises several related software packages such as Sibernetic [104], a platform for simulating the worm's interactions with its fluid environment on a CPU or GPU, and Geppetto [18], a more general web-based platform for simulation and visualization of biological systems. OpenWorm shares many of the same open science goals as that of the work in this thesis. Neurokernel's design requirements differ from those of OpenWorm owing to the much higher level of complexity of the fly's connectome. Neurokernel's LPU interface API, for example, is predicated upon the assumption that successful construction of an accurate whole fly brain emulation must necessarily involve integration of multiple LPU models of different

provenance. Similarly, Neurokernel's communication mechanism presupposes the eventual need for multiple GPUs as fly brain models become more comprehensive and biologically plausible.

The Flysim project has constructed a CPU-based simulation of 22,000 neurons from the FlyCircuit database [22] in the fly brain [60]. This model only employs spiking neurons and ionotropic synapses, however, does not currently distinguish between different modules within the brain, and does not support the use of GPUs. To better support the demands of models that account for more detailed fly neuron data that will be available in the near future, developers of this project are currently investigating the porting of their simulation to Neurokernel to utilize its GPU support.

The Green Brain Project aims to construct a modular model of the honeybee brain that describes how it realizes olfactory and visual detection, classification, and learning functions, as well as multisensory integration of olfactory and visual information. This project utilizes the GeNN neural network simulator to execute models of sensory subsystems in the bee brain [25], some of which have already been published [142].

2.5 Summary

Although the computational power of spiking neural networks has been shown to be greater than other neural network models [83], attempting to model the fruit fly brain without accounting for the nontrivial number of its neurons that do not emit spikes poses significant problems. The currently minimal support for executing heterogeneous networks comprising non-spiking neurons on GPUs was a major factor in the implementation of the Neurodriver component. Future versions of Neurodriver stand to benefit from GPU-based implementations simulation engines currently under development such as Brian2GeNN and Myriad that did not exist when the Neurokernel project was initiated.

Currently available neural simulation software affords researchers with a range of ways of constructing neural circuit models. These include tools that enable models to be explicitly

expressed as systems of differential equations [52], structured documents [48], or explicit calls to a high-level programming API [19, 30, 37]. They also include tools for defining and manipulating neural connectivity patterns [51, 9, 32]. A platform for developing emulations of the entire fruit fly brain, however, must provide programming services for expressing the functional architecture of the whole brain (or its subsystems) in terms of subunits with high-level information processing properties that clearly separate between the internal design of each subunit and how they communicate with each other. Neurokernel's architecture specifically targets these gaps by providing both the high-level APIs needed to explicitly define and manipulate the architectural elements of brain models as well as the low-level computational substrate required to efficiently execute those models' implementations on multiple GPUs (see Fig. 2.2).

Existing tools for interfacing neural models or simulators such as [106, 33] currently provide no native support for the use of GPUs and none of the aforementioned services required to scale over multiple GPU resources. Neurokernel addresses the problem of model incompatibility in the context of fly brain modeling by ensuring that GPU-based LPU model implementations and inter-LPU connectivity patterns that comply with its APIs are interoperable regardless of their internal implementations.

Despite the impressive performance GPU-based spiking neural network software can currently achieve for simulations comprising increasingly large numbers of neurons and synapses, enabling increasingly detailed fruit fly brain models to efficiently scale over multiple GPUs will require resource allocation and management features that are not yet provided by currently available neural simulation packages that support GPUs (§ 2.4.2). By explicitly providing services and APIs for management of GPU resources, Neurokernel will enable fly brain emulations to benefit from the near-term advantages of scaling over multiple GPUs while leaving the door open to anticipated improvements in GPU technology that can further accelerate the performance of fly brain models.

The challenges of reverse engineering neural systems have spurred a growing number

of projects specifically designed to encourage collaborative neuroscience research endeavors. These include technologies for model sharing [58, 48, 50], curation of publicly available electrophysiological data [128], and the construction of comprehensive nervous system models for specific organisms [133]. For collaborative efforts at fruit fly brain modeling to succeed, however, there is a need to both ensure the interoperability of independently developed LPU models without modification of their internal implementations while simultaneously enforcing a model of the overall brain connectivity architecture. By imposing mandatory communication interfaces upon models, Neurokernel explicitly ensures that LPU models may be combined with other compatible models to construct subsystem or whole brain emulations.

Although the Neurokernel project is specifically focused upon reverse engineering the fruit fly brain, the framework's ability to capitalize upon the structural modularity of the brain and facilitate collaborative modeling stand to benefit efforts to reverse engineer the brains of other model organisms. To this end, Neurokernel has already been used to successfully scale up the retinal model described in § 2.3.1 to emulate the retina of the house fly [74], which comprises almost 10 times as many differential equations (18.8 billion) as that of the fruit fly (1.95 billion). Further development of Neurokernel's support for multiple GPUs (§ 5.2) and - eventually - neuromorphic hardware will open the doors to collaborative modeling of the brains of even more complex organisms such as the zebra fish and mouse.

Chapter 3

NeuroArch: a Graph dB for Representation of Executable Fly Brain Circuits

3.1 Introduction

NeuroArch is a software package for specification, storage, and querying of both biological data regarding the fruit fly brain and executable models built upon that data within a single graph database. A key aim of NeuroArch is to enable the algorithmic construction of neural circuit models based upon large-scale biological data sets by closing the gap between biological data and the models that depend upon them; we describe the high-level requirements for representation of fly brain circuit data to achieve this aim in § 3.2. NeuroArch employs a data model that preserves the structural and semantic relationships between different biological and modeling objects; this data model is presented in § 3.3 and its mapping into a graph database described in § 3.4. In § 3.5, we present features of NeuroArch's API that exploit the data model to fulfill some of the requirements described in § 3.2 and present

Parts of this chapter appear in [45, 46].

demonstrations of the API's functionality in § 3.6. We compare NeuroArch with currently available open fruit fly biological data resources, neural model sharing services, and neural model specification technologies in § 3.7 and summarize NeuroArch's unique design features in § 3.8.

3.2 Data Representation Requirements

3.2.1 Represented Information

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

3.2.1.1 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. [62].

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.

3.2.1.2 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 [79].

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.

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.

3.2.2 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.

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 [27].

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.

3.2.3 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.

3.3 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 § 3.2.1. These hierarchies describe how entities at one level of granularity/abstraction are defined in terms of entities at some lower level of granularity/abstraction.

3.3.1 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. 3.1). 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 § 3.2.1.1 is deemed to be attributes of specific entities in the data model and therefore does not appear in Tab. 3.1.

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

3.3.2 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 (§ 3.2.1.1); 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 § 3.3.1:

`/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`

3.3.3 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. 3.2). 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 3.2: Ownership relationships between executable circuit entities in NeuroArch's data model.

3.3.4 Combined Hierarchy of Biological and Executable Circuit Entities

Fig. 3.1 depicts the combined hierarchies of biological and executable circuit entities supported by the data model. NeuroArch permits additional entities beyond those described in Fig. 3.1 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.

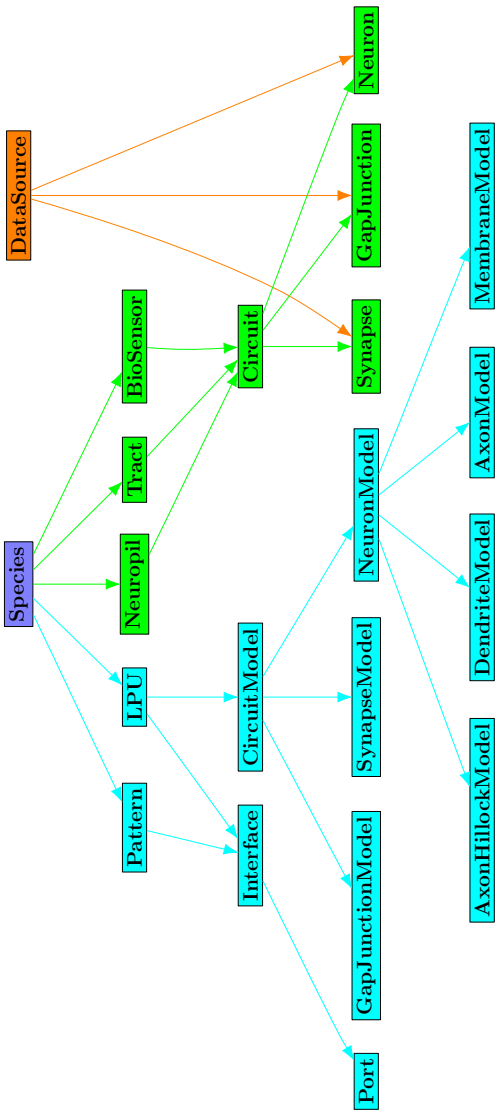


Figure 3.1: Objects and relationships in NeuroArch's data model. Green nodes denote biological circuit entities, while cyan nodes correspond to executable circuit entities; the purple node representing the species associated with specific biological or executable circuit entities is the root of both 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.

3.4 Mapping the Data Model into an Object Graph Database

NeuroArch is implemented in Python and built upon the open-source database OrientDB[1]. 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[2] graph traversal language supported by many graph databases, and the availability of an actively developed Python interface[3] 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. 3.1 or 3.2.

3.4.1 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. 3.2 and 3.3, respectively.

[1] `http://orientdb.com`

[2] `http://github.com/tinkerpop/gremlin/`

[3] `http://github.com/ostico/pyorient`

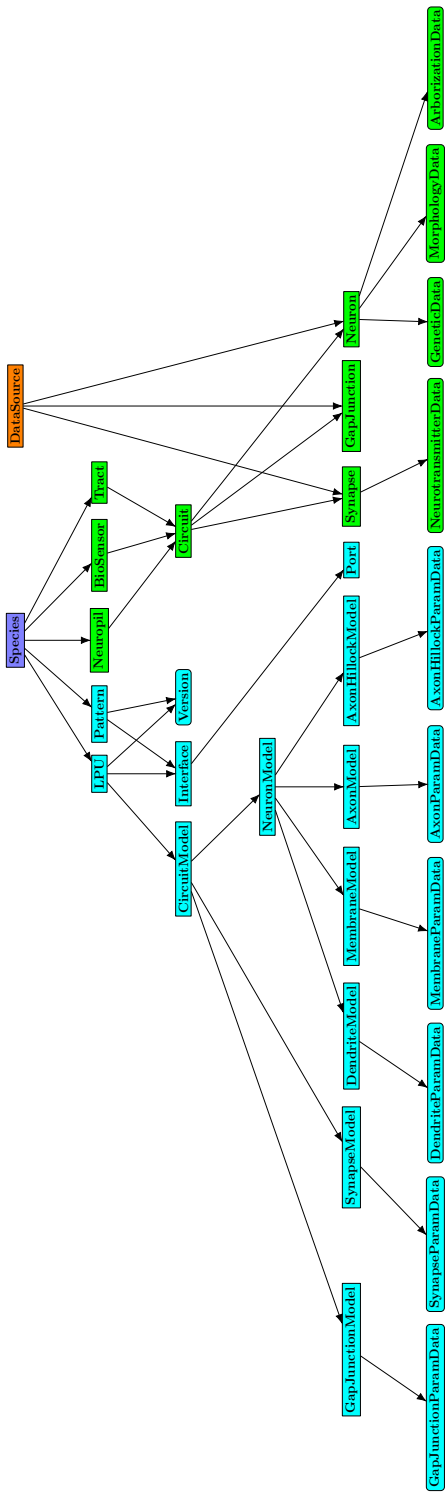


Figure 3.2: Object types and containment/ownership relationships in NeuroArch's database. Green nodes denote objects corresponding to biological circuit entities, while cyan nodes denote objects corresponding to executable circuit entities. Rectangular nodes correspond to entities defined by NeuroArch's 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.

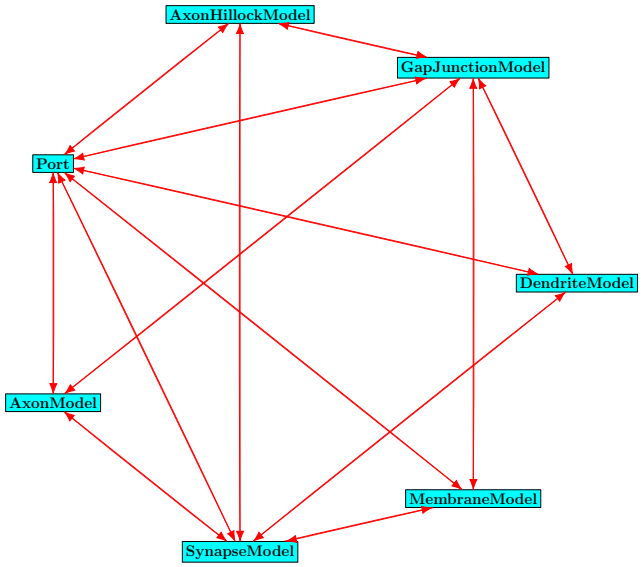


Figure 3.3: Data transmission relationships (red) between executable circuit objects (cyan) in NeuroArch's database.

3.4.2 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. 3.3).

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

Table 3.3: Objects used to store biological data.

3.4.3 Storage of Executable Circuit Data Objects

As with the biological data objects described in § 3.4.2, attributes of objects representing components of executable circuits may be mapped to separate nodes to facilitate certain queries (Tab. 3.4).

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

Table 3.4: Objects used to store executable circuit component data.

3.4.4 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. 3.4). Each `LPU` or `Pattern` node instance owns its own fully independent copy of the subgraph of lower level components that describe its version.

3.4.4.1 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 (§ 3.2.1.1, Fig. 3.2).

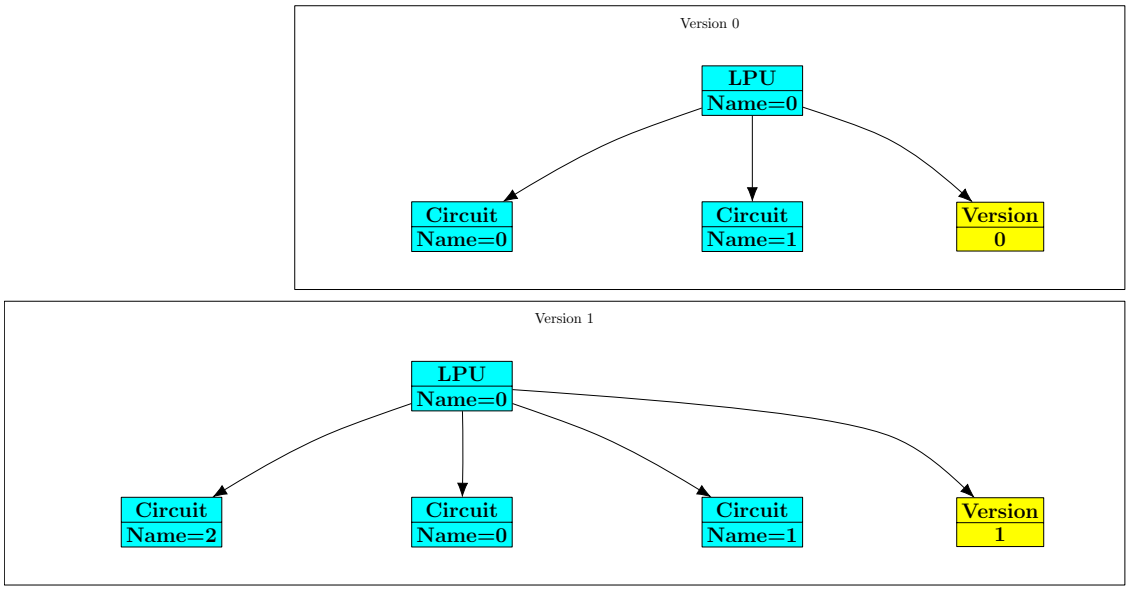
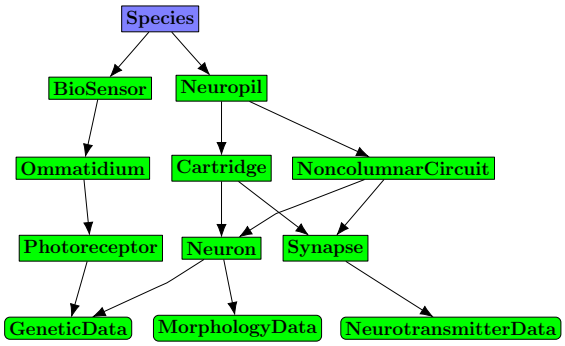


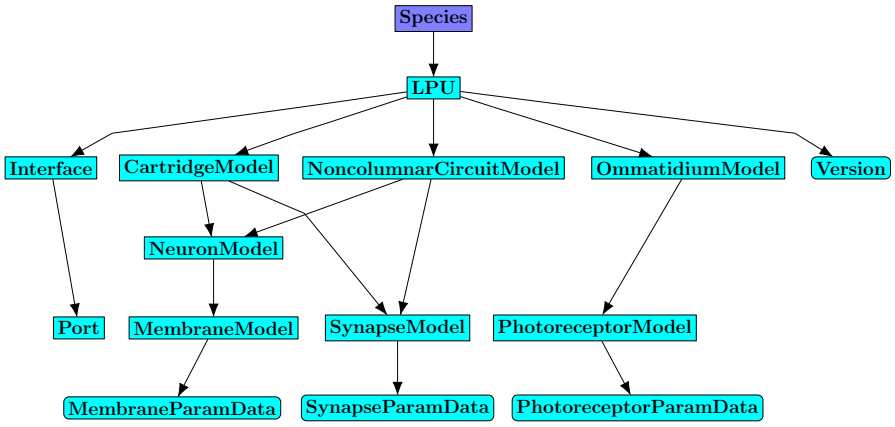
Figure 3.4: Representation of multiple versions of a single `LPU` (with name 0) using an additional `Version` node (yellow). Version 1 of the sample LPU differs from version 0 by virtue of the presence of an additional `Circuit` node in its subgraph.

3.4.5 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 retina as described in [79] can be mapped to the data model as depicted in Figs. 3.5 and 3.6 and Tab. 3.5.



(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. 3.2 are listed in Tab. 3.5a.



(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. 3.2 are listed in Tab. 3.5b.

Figure 3.5: 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. 3.2)

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 3.5: 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.

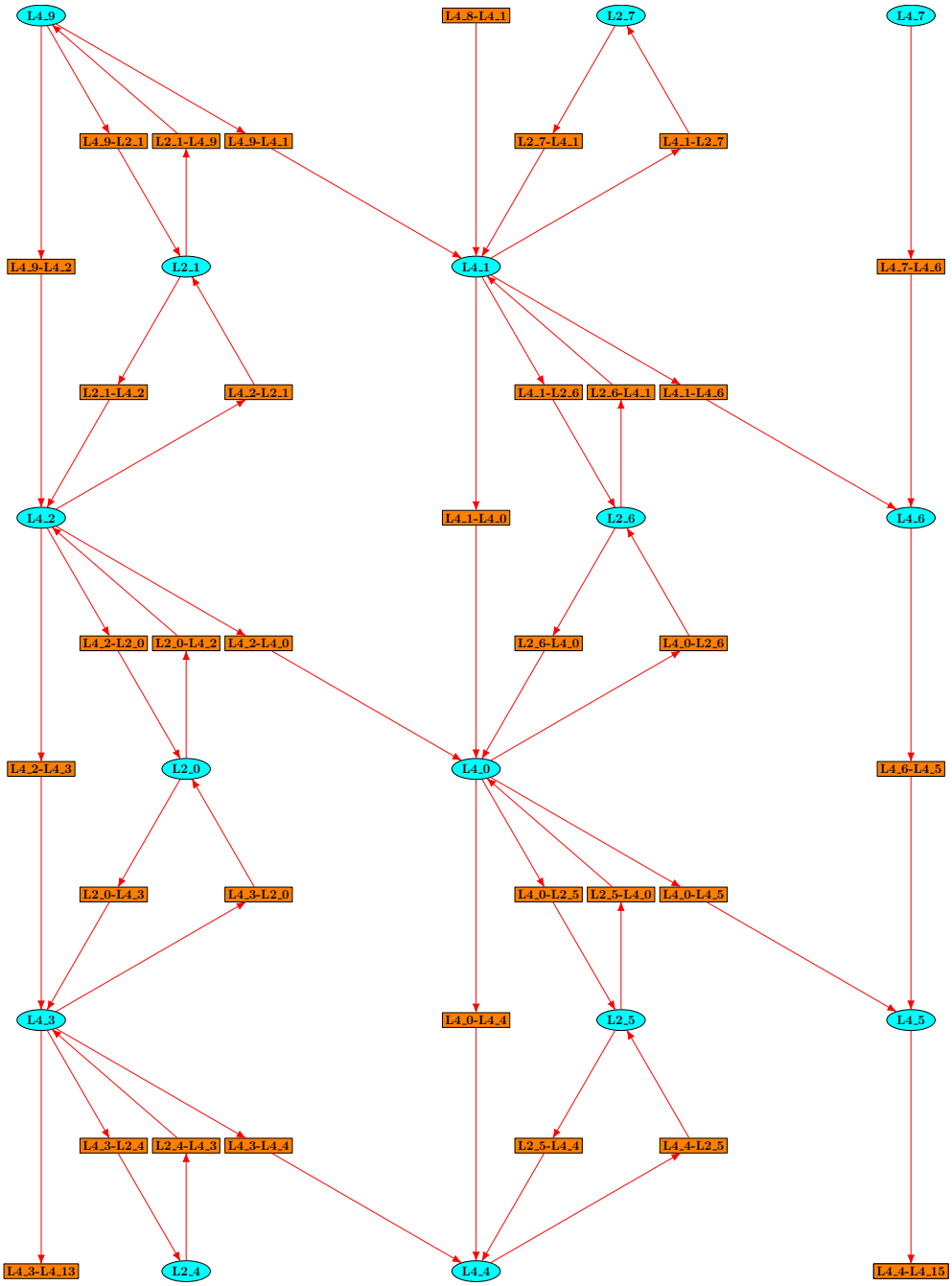


Figure 3.6: Data transmission (red) relationships between a subset of the circuit design components of the lamina LPU. This diagram only depicts the nodes corresponding to L2 and L4 neurons in several adjacent cartridges (cyan) and synapses between them (orange).

3.5 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 [123], 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.

3.5.1 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 § 3.3 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 (§ 3.5.3). 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.

3.5.2 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 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'])
```

3.5.3 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
```

3.5.4 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[4] [89] and NetworkX[5] [53]; 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 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
```

[4] `http://pandas.pydata.org`

[5] `http://networkx.github.io`

```
# 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')
```

3.5.5 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 [44] differs from that assumed by NeuroArch's data model (§ 3.3.3), 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.

3.6 Testing Neuroarch's Functionality

To test the features described in § 3.5, 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 Neurodriver repository [6]. NeuroArch's OGM was invoked within a Neurokernel emulation to extract these LPUs and pattern circuits, convert them to the current graph

[6] http://github.com/neurokernel/neurodriver

structure expected by Neurokernel (§ 3.5.5), 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 [7]. 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.

We also 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.

Finally, we used NeuroArch to generate of models of the fly's central complex executable by Neurokernel using incomplete biological information. The experimental scenarios enabled by the use of NeuroArch are presented in § 4.

3.7 Related Work

3.7.1 Open Biological Data Repositories

The immense interest in the fruit fly as a model organism in neuroscience and other biological fields has led to a growing array of open fruit fly biological data resources. These

[7] `http://github.com/neurokernel/vision`

range from highly detailed solitary datasets such as the fruit fly medulla connectome [1] to databases such as NeuroMorpho [4] that expose digital reconstructions of neuron morphologies from imaging data for the fruit fly and other model organisms. In addition to providing extensive sets of neuron morphologies, databases such as FlyCircuit [22] also provide spatial distribution, neurotransmitter data, genetic driver, and neural tract information for each stored neuron. FlyBase provides genomic reference information for several fruit fly species organized into acknowledged gene sets [5]. These resources afford varying levels of queriability. Some datasets such as those provided by [1] can only be accessed as raw data files. NeuroMorpho and FlyCircuit provide online search tools that permit the use of metadata or anatomical characteristics in querying available neuron data. FlyBase permits users to search for specific genes or groups of related genes and to navigate to specific points in the fly genome using genetic coordinates or landmarks [34].

The profusion of data modalities represented by currently available open fly data resources places an increasing burden upon users of this data to relate the different modalities during research. Although databases such as FlyCircuit do contain some linkage between anatomical neuron data, source genetic drivers used to identify a specific neuron, and drivers of related neurons, the database only accounts for a portion of the fly brain and does not cross-reference other important resources such as FlyBase. To address this disparate array of resources, the Virtual Fly Brain (VFB) project [91] integrates fly brain data from various sources behind a single online user interface using an consortium-developed ontology that provides a common framework for labeling fruit fly anatomical features [27]. Users may graphically browse brain data by anatomical region and construct ontology queries for specific neuron data by combining search elements such as the regions innervated by a neuron and expressed genes or phenotypes associated with a neuron.

The rapidly growing magnitude of fly neuron datasets has also spurred development of technologies for more efficient navigation of collected data. NBLAST is an algorithm for quantifying pairwise similarity of unannotated neuron data by spatial position within the

brain and local geometry [26]. Inspired by algorithms for finding matches between genetic sequences, NBLAST can be used to cluster large neuron datasets and search for neurons proximate to a given query neuron's location or with similar geometric characteristics. An online instance of the algorithm is available that may be used to query the FlyCircuit database and visualize query results in 3D.

Query mechanisms such as those provided by VFB and NBLAST afford the possibility of constructing more complex queries than simple searches for data annotation labels and terms. VFB's query mechanism has the additional advantage of supporting expression of constructed queries in a clearly intelligible format. Although the accessibility of these tools is invaluable for manually studying and analyzing neuron data, their lack of support for any public query API significantly limits their utility in software-driven model development; data obtained from manually performed queries against the above repositories must currently be downloaded and reformatted in order to be used by applications designed to infer model structure. NeuroArch explicitly targets this shortcoming with a database interface designed for algorithm developers to use within model generation programs rather than manually. Encapsulation of typical queries required to retrieve circuit data for execution enables researchers to focus upon defining how models are constructed from biological data than on the low-level desiderata of how to translate between different data formats and search for specific data points required for model construction.

Links between related data such as genetic driver lines and the neurons they pinpoint can be accessed to varying extents by the query interfaces of existing online fly brain data resources; however, the underlying graph structure formed by integration of different biological data sets is not exposed for explicit traversal. Restricting the possible queries model that may be performed against integrated data to those a user may manually enter significantly limits the extent to which model developers can exploit large biological data sets to algorithmically construct models that are not feasible to assemble manually. By storing fly data in a graph database that supports powerful general-purpose graph query languages such as

Gremlin [123], the current range of queries encapsulated by NeuroArch's OGM can be easily extended to include additional queries model developers may require in the future. Although NeuroArch's current OGM is designed to for writing desktop model generation programs, the underlying graph database technology makes it possible to expose NeuroArch's API through web interfaces should the need arise.

NeuroArch's design aims for representation of fly brain data are similar to those of Bio4j, an open-source platform for integration of open bioinformatic datasets using typed graph models [105]. 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. In contrast to Bio4j, however, NeuroArch also includes models created using biological data within the same data model and graph database to enable the same query tools to be used both for exploring existing data and improving existing models via programs rather than by manual means.

3.7.2 Model Representation Technologies

The increasing importance of biologically detailed neuron and network models in shedding light on how the brain implements its information processing functions has prompted the development of a range of neuroinformatic technologies for representation and sharing of neuronal model descriptions. Although simulators such as GENESIS, NEURON, and NEST provide their own native languages for model specification (§ 2.4), the lack of interoperability between these languages has led to the development of cross-simulator support

for model specification in Python and open model description languages such as NeuroML, NineML, and SpineML that build upon the Extensible Markup Language (XML) to provide simulator-independent formats for specifying the structure and parameters of neural circuit models. As noted in § 2.4, PyNN provides a high-level Python interface for constructing neural circuit using a range of neuron and synapse models that can be run using several neuronal network simulators [30]. NeuroML 1.x enables specification of neuronal models at multiple levels of detail ranging from low-level descriptions of electrochemical channels through conductance-based compartmental neuron models to descriptions of network connectivity between neurons [48]. NineML provides additional flexibility by enabling the use of neurons with arbitrary dynamics and arbitrary network connectivity patterns [114]. To address limitations in NeuroML 1.x and the incomplete state of the NineML format, NeuroML 2.0 [49] and SpineML [118] respectively provide support for point neuron networks and specification of simulation runtime information. A key feature of NeuroML 2.0 is the Low Entropy Model Specification Language (LEMS), an XML dialect designed to describe the dynamical behavior of modeling components.

NeuroML and related specification formats provide the means of specifying connectivity patterns between neurons and synapses in a circuit. These patterns may be expressed as explicit connections between individual neurons and synapses, or may employ select algorithmic templates that describe how a pattern may be generated [48]. The number of templates available places limitations upon what patterns may be easily defined, however. To enable additional pattern construction flexibility, the Connection Set Algebra (CSA) provides a means of describing a wide range of connectivity patterns independent of neuron population sizes and specific neuron and synapse models [32]. New patterns may be constructed from existing patterns using operators in the algebra. Support for CSA is available as a standalone Python package and is being incorporated into the NineML specification.

From the perspective of a circuit designer, formats such as NeuroML provide a human-readable means of neural circuit model expression that hides the model's numerical com-

plexity. As circuit models grow in both size and detail to account for increasingly comprehensive neuronal data sets such as [4, 22, 16, 1], both manual construction and modification of circuit model expressed using specification formats becomes unfeasible. While amenable to processing by the array of available tools designed for structured documents, neuronal model specification formats lack the query capabilities afforded by database platforms that are needed to efficiently navigate the graph structure and modify large circuits. NeuroArch addresses this limitation by making the easily queryable graph database representation of integrated executable circuit components and the biological data used to infer them the primary representation manipulated by model designers; neural circuit specification or graph file formats are used to either import or record snapshots of NeuroArch's database contents, but do not constitute the main representation with which model construction/manipulation applications interact.

Given that the precise formulation of neuronal models is essential to their critical evaluation [99], open formats for unambiguous neuronal model specification enable researchers to share and compare different computational models that would otherwise have to be manually reconstructed from published descriptions. These formats, however, do not explicitly prescribe a means of interfacing independently developed network models even when those models are represented using the same format. This limits their utility as a basis for explicitly collaborative construction of brain models in which different functional subcircuits may be designed by different researchers. NeuroArch's data model addresses this limitation by explicitly requiring that circuit models designed to interact with other circuits contain communication interfaces that may be linked to the compatible interfaces exposed by other circuits; moreover, such models may be immediately integrated and concurrently evaluated with Neurokernel if they are composed of neuron and synapse models supported by Neurodriver (§ 2.2.6).

3.7.3 Model Sharing Resources

Online resources such as ModelDB [58] enable researchers to publicly share neuronal and network models in conjunction with associated publications. ModelDB imposes few constraints on how models must be represented and does not preserve any information as to how a model may have been improved or altered since its creation; modification of an existing model or its integration with models of other neural circuits in the brain still may require significant efforts on the part of a researcher. The Open Source Brain Project (OSB) [50] remedies some of these limitations by standardizing upon NeuroML for model specification. Individual models are maintained as separate projects on the public revision control site GitHub[8] to record a history of changes and improvements made to a model. Apart from organizing stored models into basic navigable categories such as author, neuron type, concept, and simulation environment, neither ModelDB nor OSB supports direct querying of model internals. ModelDB is integrated with a searchable database of neuronal properties called NeuronDB [87], but the latter resource does not incorporate the most recent large fruit fly datasets available from FlyCircuit, NeuroMorpho, and other sites. OSB does not currently provide any direct integration with existing biological databases. NeuroArch's data model, by contrast, explicitly represents both biological data and the executable circuits inferred from them within a single resource. Circuit model designers using NeuroArch may therefore immediately build upon existing models within NeuroArch's database by testing new hypotheses as to how to interpret biological data during model construction without having to manually import or translate existing models from other representations.

3.8 Summary

The explosion in publicly available fly connectome data and increasing need for open science approaches to model development in recent years have motivated the development of a

[8] http://github.com

profusion of valuable online biological data repositories and neuroinformatic tools for respectively sharing neurobiological and modeling data within the research community. Existing data repositories do not provide the means of programmatic access required to efficiently access large sets of biological data from programs that implement model construction algorithms. The relational structure of existing fly databases also precludes utilization of the inherent graph structure of fly connectome data and the graph relationships between linked data of different modalities. NeuroArch targets these limitations by providing a graph-based representation of stored biological data and a high-level API for sophisticated querying of brain data by programs rather than manual users. NeuroArch also reduces the complexity of translating biological data from disparate sources into executable circuits model by exposing information regarding both through a single interface; NeuroArch's API lets brain modelers focus on how to infer accurate circuit models from connectome data rather than the technical aspects of extracting and loading biological data required for model construction or translating models between different representation formats.

Chapter 4

Generating an Executable Model of the Central Complex

4.1 Introduction

In this chapter, we present the use of the open pipeline developed in § 2 and 3 to generate executable models of functional units in the fruit fly brain from incomplete biological data. The central complex (CX) is a group of neuropils in the fruit fly brain known to play an essential role in the spatial representation and memory of visual data, directional control of locomotion, and integration of spatial information for locomotor control [112, 132]. Although increasingly detailed information regarding the structure of the various neurons in these neuropils has become available [82, 141], information regarding the synaptic connectivity and local neuron circuitry in the CX still remains far from complete. The fact that the CX neuropils do not directly receive sensory input signals further complicates analysis of its information processing functions because of their dependence upon the preprocessing performed by other neuropils.

After reviewing the nomenclature used to label neuropils in § 4.2.1, we describe how projection neurons in the CX neuropils may be uniquely labeled in terms of their arborization

regions and terminal polarities in § 4.2.2 We review the high-level structure of the CX neuropils and accessory neuropils directly connected to those in the CX, identify the relevant arborization regions within them in § 4.3, and identify the families of local and projection neurons that innervate them in § 4.5. Using arborization data compiled for these neurons (listed in Appendix 5.4), we detail the use of NeuroArch to enable the inferring of synapses between a subset of the known CX projection neurons with overlapping arborizations. These hypothesized synapses are then used to construct executable LPU models corresponding to two of the CX neuropils in § 4.6. Finally, we review other work on neural circuit model generation and modeling CX in the context of the approach advanced in this chapter in § 4.7 and conclude the chapter in § 4.8.

4.2 Terminology

4.2.1 Neuropil Nomenclature

Drosophila neuropils are identified in this document using the nomenclature described in [62]. Some neuropils are referred to by different names either in other literature or in other insects; [62, Tab. S13] maps the employed nomenclature to that used in [22] and - for the most part - in [82]. For neuropils that occur in pairs, upper case denotes the neuropil on the left side of the fly brain (from a dorsal perspective the fly) while lower case denotes the neuropil on the right side of the fly brain.

- Antennal Lobe (AL).
- Anterior optic tubercle (AOTU) - Also known as optic tubercle (OPTU [22]).
- Antler (ATL) - Corresponds to dorsal part of caudalcentral protocerebrum (CCP) [62, Tab. S13].
- Bulb (BU) - Also known as lateral triangle (LT, LAT, Lat Tri [82, Tab. S13] or LTR [54]).

- Crepine (CRE) - Posterior part also known as dorsal part of IDFP [62, Tab. S13]; comprises a region called the rubus (RUB) [141, p. 1001] or round body (RB) [141, p. 1031].
- Lateral accessory lobe (LAL) - Also known ventral body (VBO [54]) or inferior dorsofrontal protocerebrum (IDFP) [82, Fig. S1]. Comprises the gall (GA) [62, Tab. S13], whose dorsal and ventral portions are referred to as the dorsal and ventral spindle bodies (DSB and VSB, respectively) [141, p. 1021].
- Ellipsoid body (EB) - Also known as lower central body (CBL [112]).
- Fan-shaped body (FB) - Also known as upper central body (CBU [112]).
- Inferior Bridge (IB) - Corresponds to ventral part of caudalcentral protocerebrum (CCP) [62, Tab. S13].
- Lobula (LO).
- Lobula Plate (LOP).
- Noduli (NO).
- Posterior slope (PS) - Corresponds to caudalmedial protocerebrum (CMP) and - possibly - part of the ventromedial protocerebrum (VMP) [62, Tab. S13].
- Protocerebral Bridge (PB).
- Superior medial protocerebrum (SMP) - Corresponds to superior dorsolateral protocerebrum (SDFP) and medial part of inner dorsolateral protocerebrum (IDLP) [62, Tab. S13].
- Ventrolateral protocerebrum (VLP) - Contains optic glomeruli [62, p. 42, Supp.].
- Wedge (WED) - Also known as the caudal ventrolateral protocerebrum (CVLP) [62, p. 42, Supp.].

4.2.2 Neuron Labeling

Most neurons innervating the various CX and accessory neuropils possess at least two distinct clusters of dendrites (postsynaptic terminals) and/or axons (presynaptic terminals) that occupy geometrically distinct regions of the innervated neuropils [54]. These clusters are referred to as arborizations (Fig. 4.1). Since many CX neurons belong to distinct sets of morphologically similar neurons with similar arborization patterns, it is useful to use the latter to uniquely label each CX neuron type. If neurotransmitter profiles are ignored and each CX neuron type is assumed to be represented by a single neuron, then each neuron's label unambiguously encodes the geometric regions of its arborizations and whether each arborization contains dendrites, axons, or both. This labeling scheme can be described in terms of the following parsing expression grammar (PEG) [40]; the grammar may be used to extract the arborizations of a particular neuron for constructing models of the CX circuitry (e.g., by using overlapping presynaptic and postsynaptic arborizations to infer synaptic connectivity). Note that a special case for handling the string LRB in the ⟨name⟩ rule (which corresponds to the left RB region of CRE) is necessary to prevent that string from being incorrectly parsed into LB and RB.

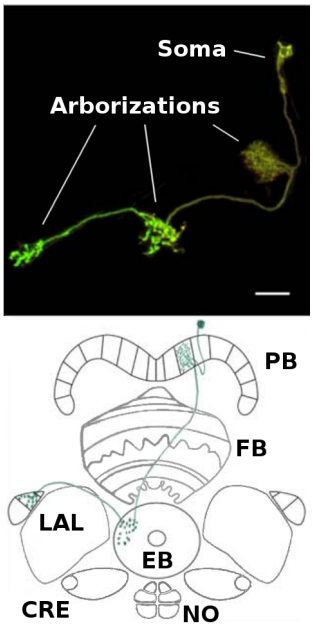


Figure 4.1: Example of neuron arborizations for a PB-EB-LAL neuron (§ 4.5.4.8) [141]. Each of the neuron arborizations occupies a specific region in different neuropils. (©2015 Wiley Periodicals, Inc.)

$$\langle \text{label} \rangle := \langle \text{arborization} \rangle \, (\langle \text{hyphen} \rangle \langle \text{arborization} \rangle) +$$

$$\langle \text{arborization} \rangle := \langle \text{neuropil} \rangle \langle \text{slash} \rangle \langle \text{regions} \rangle \langle \text{slash} \rangle \langle \text{neurite type} \rangle$$

$$\langle \text{regions} \rangle := \langle \text{region} \rangle \, (\langle \text{bar} \rangle \langle \text{region} \rangle) *$$

$$\langle \text{neuropil} \rangle := (\text{BU/bu/CRE/cre/EB/FB/IB/ib/LAL/}$$
$$\text{lal/NO/no/PB/PS/ps/SMP/smp/WED/wed})$$

$$\langle \text{region} \rangle := \langle \text{tuple2} \rangle / \langle \text{tuple3} \rangle / \langle \text{name} \rangle$$

$$\langle \text{tuple2} \rangle := \langle \text{left paren} \rangle \langle \text{name} \rangle \langle \text{comma} \rangle \langle \text{name} \rangle \langle \text{right paren} \rangle$$

$$\langle \text{tuple3} \rangle := \langle \text{left paren} \rangle \langle \text{name} \rangle \langle \text{comma} \rangle \langle \text{name} \rangle \langle \text{comma} \rangle \langle \text{name} \rangle \langle \text{right paren} \rangle$$

$$\langle \text{name} \rangle := \text{LRB/} \, (\langle \text{side} \rangle ? \, (\langle \text{integer} \rangle / \langle \text{range} \rangle / \langle \text{alpha} \rangle / \langle \text{list} \rangle)) \, /$$
$$(\langle \text{side} \rangle ! \, (\langle \text{integer} \rangle / \langle \text{range} \rangle / \langle \text{alpha} \rangle / \langle \text{list} \rangle))$$

$$\langle \text{side} \rangle := (\text{L/R/LR/RL})$$

$$\langle \text{neurite type} \rangle := (\text{s/b/bs/sb})$$

$$\langle \text{range} \rangle := \langle \text{left bracket} \rangle \langle \text{integer} \rangle \langle \text{hyphen} \rangle \langle \text{integer} \rangle \langle \text{right bracket} \rangle$$

$$\langle \text{list} \rangle := \langle \text{left bracket} \rangle \langle \text{alpha} \rangle \left(\langle \text{comma} \rangle \langle \text{alpha} \rangle \right) * \langle \text{right bracket} \rangle$$

$$\langle \text{integer} \rangle := [0-9]+$$

$$\langle \text{alpha} \rangle := [\text{a}-\text{z}, \text{A}-\text{Z}, 0-9]+$$

$$\langle \text{hyphen} \rangle := -$$

$$\langle \text{bar} \rangle := |$$

$$\langle \text{slash} \rangle := /$$

$$\langle \text{comma} \rangle := ,$$

$$\langle \text{left paren} \rangle := ($$

$$\langle \text{right paren} \rangle :=)$$

$$\langle \text{left bracket} \rangle := [$$

$$\langle \text{right bracket} \rangle :=]$$

Neuropils are denoted by their abbreviated names as specified in § 4.2.1 and [62]; regions or compartments within neuropils are described and assigned names in § 4.3. The neurite type may be spine (s), bouton (or bleb) (b), or a combination thereof (bs, sb). In the absence of detailed data regarding synapses, information flow polarity is assumed to be reflected by neurite type; spines are assumed to be postsynaptic (and accept input), while boutons are assumed to be presynaptic (and emit output) [141, p. 1002]. Left and right are assumed to be with respect to a dorsal view of the fly.

4.3 Structure of Neuropils in and Associated with the Central Complex

This section presents details regarding the high-level structure of the various CX neuropils and the accessory neuropils to which they are connected.

4.3.1 Protocerebral Bridge (PB)

The PB neuropil comprises 18 regions called glomeruli [141] connected to other substructures within the CX (Fig. 4.2). The local neuron population of PB comprises 8 [141, p. 1007] or 10 [82, p. 1743] types of local neurons (Fig. 4.10, Tab. 3). A single PB region label matches the following regular expression:

$$\langle \text{glomerulus} \rangle := [\text{L}, \text{R}][1-9]$$

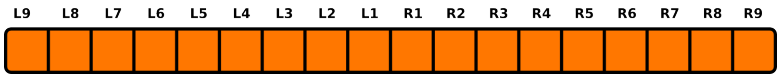


Figure 4.2: Schematic of regions in PB used to identify neurons by their arbors [82, 141].

4.3.2 Fan-Shaped Body (FB)

The FB neuropil comprises multiple lateral layers [112]; most recent work suggests the presence of 9 layers [141, p. 1011]. The neuropil is subdivided vertically into 8 [82] or 7 [141, p. 1010] columns called segments [54]; however, it seems that only some of its layers (1-5) exhibit clearly columnar structure [141, p. 1008] (4.3). Regions in FB are connected by local neurons called pontine neurons; some of these neurons connect adjacent layers, while others connect adjacent segments [54, p. 349, 352]. A representative class of pontine neurons comprising symmetric neurons that connect each segment in one side of FB with each segment in the other side such that the presynaptic and postsynaptic arborizations are

4 segments apart [54, p. 352] (although more recent work suggests that each neuron might be a bundle of 2 neurons [145, p. 1439]) is depicted in Tab. 5. Other classes dorsoventrally connecting different layers in FB may exist, but they have not been systematically identified. A single region in FB is denoted by a label matching the regular expression

$$\langle \text{region} \rangle := \backslash(\overbrace{[1-9]}^{\text{layer}}, \overbrace{[\text{L},\text{R}][1-4]}^{\text{segment}}\backslash)$$

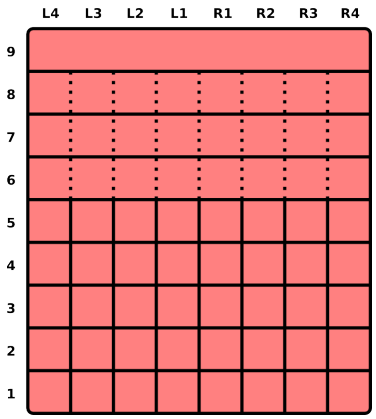


Figure 4.3: Schematic of regions in FB used to identify neurons by their arbors [82, 141].

4.3.3 Ellipsoid Body (EB)

The EB neuropil is a toroidal structure that comprises 16 wedges [141, p. 1013] (Fig. 4.4a), 8 tiles [141, p. 1018] (Fig. 4.4b), 3 shells (anterior, medial, posterior) [141, p. 1013], and 4 rings [82] (Fig. 4.4c). Wedges extend radially through full radius of the EB torus and occupy the posterior and medial shells or all 3 shells [141, p. 1013]. Tiles are restricted to the posterior shell [141, p. 1014]; tiles geometrically overlap with corresponding wedges as described in Tab. 4.1. Although EB appears to contain local neurons [22], these neurons have not yet been systematically identified; there is some evidence for EB pontine neurons in related fly species such as *Neobellieria* [113, p. 11]. Each region in EB is denoted by a

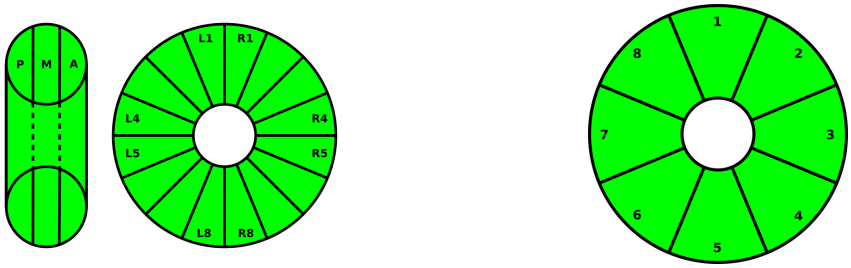
label that matches the regular expression

$$\langle \text{region} \rangle := \overbrace{[1-8]}^{\text{tile}} | \backslash (\overbrace{[\text{L}, \text{R}][1-8]}^{\text{wedge}}, \overbrace{[\text{P}, \text{M}, \text{A}]+}^{\text{shell}}, \overbrace{[1-4]}^{\text{ring}} \backslash)$$

For EB regions other than tiles, the region denoted by a label comprises the volume intersected by the specified wedges, shells, and rings. For example, $(\text{L1}, [\text{P}, \text{M}], 4)$ represents the volume in which wedge L1, shells P and M, and ring 4 overlap.

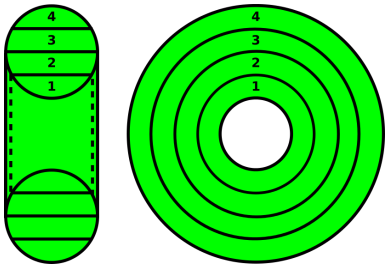
Tile	**Wedge**
EB/1/x	EB/([L1,R1],P,x)/x
EB/2/x	EB/(R[2,3],P,x)/x
EB/3/x	EB/(R[4,5],P,x)/x
EB/4/x	EB/(R[6,7],P,x)/x
EB/5/x	EB/([R8,L8],P,x)/x
EB/6/x	EB/(L[6,7],P,x)/x
EB/7/x	EB/(L[4,5],P,x)/x
EB/8/x	EB/(L[2,3],P,x)/x

Table 4.1: Geometric overlap between EB tiles and wedges.



(a) Shells (L) and wedges (R)[141].

(b) Tiles [141].



(c) Rings [146].

Figure 4.4: Schematics of regions in EB. All circular schematics are anterior; sagittal views in Figs. 4.4a and 4.4c are posterior to anterior from left to right).

4.3.4 Noduli (NO)

The NO neuropils comprise 3 distinct structures (NO1, NO2, NO3) divided into subcompartments (Fig. 4.5) [141, p. 1017]. In contrast to the other CX neuropils, the noduli lack segregated populations of local neurons [22, p. 5]. Each NO region label matches the following regular expressions:

$$\langle \text{subcompartment} \rangle := \begin{cases} [\text{L}, \text{R}] & \text{for NO1} \\ [\text{L}, \text{R}][\text{V}, \text{D}] & \text{for NO2} \\ [\text{L}, \text{R}][\text{A}, \text{M}, \text{P}] & \text{for NO3} \end{cases}$$

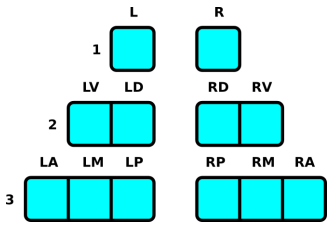


Figure 4.5: Schematic of regions in NO used to identify neurons by their arbors [141].

4.3.5 Bulb (BU)

Each of the BU neuropils comprises multiple regions referred to as microglomeruli. There appear to be 80 microglomeruli in each BU neuropil (Fig. 4.6) [82, p. 1741]. These microglomeruli ostensibly exhibit retinotopic organization [125]. Each of the BU region labels matches the regular expression

$$\langle \text{microglomerulus} \rangle := [\text{L}, \text{R}][0-9]+$$

where the integer portion of the labels ranges from 1 to 80.

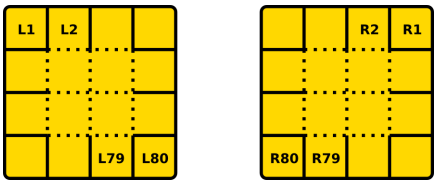


Figure 4.6: Schematic of regions in BU used to identify neurons by their arbors [82]. The relative positions of the regions does not necessarily correspond to their actual physical positions.

4.3.6 Lateral Accessory Lobe (LAL)

Each LAL neuropil comprises a region called the gall that is subdivided into a tip, dorsal, and ventral subregion; the remainder of LAL is referred to as the hammer body (HB) (Fig. 4.7)

[82]. Each of these regions has a label that matches the regular expression

$$\langle \text{region} \rangle := [\text{L}, \text{R}]\,(\text{HB}|\text{GT}|\text{DG}|\text{VG})$$

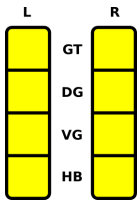


Figure 4.7: Schematic of regions in LAL used to identify neurons by their arbors [141].

4.3.7 Crepine (CRE)

Each CRE neuropil is divided into two regions (Fig. 4.8); these match the regular expression

$$\langle \text{region} \rangle := [\text{L}, \text{R}]\,(\text{RB}|\text{CRE})$$

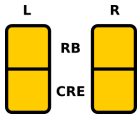


Figure 4.8: Schematic of regions in CRE used to identify neurons by their arbors [141].

4.3.8 Other Neuropils (IB, PS, SMP, WED)

Distinct regions of interest within IB, PS, SMP, and WED have not been identified; they are therefore regarded as comprising single regions on each side of the fly brain. Each region in these neuropils matches the regular expression

$$\langle \text{region} \rangle := [\text{L}, \text{R}]\,(\text{IB}|\text{PS}|\text{SMP}|\text{WED})$$

4.4 Central Complex Input Pathways and Neuron Responses

The neuropils in the CX are connected to various neuropils, but evidently not to any that directly receive sensory input except the antennal lobe (AL) [54, Fig. 24a]. Apart from connections between the CX neuropils and the accessory neuropils depicted in Fig. 4.9, connections have been observed between superior/inferior protocerebra and FB, between AOTU and BU [103, p. 9], and between VLP and PB [110, p. 9]. Preprocessed visual data from LO appears to enter the EB from BU via AOTU [102, p. 939], while additional visual input enters PB from other optic glomeruli in VLP [110, p. 9]. Other input enters FB via LAL. There also seems to be evidence of CX receiving mechanosensory information from the fly's legs [132, p. 6].

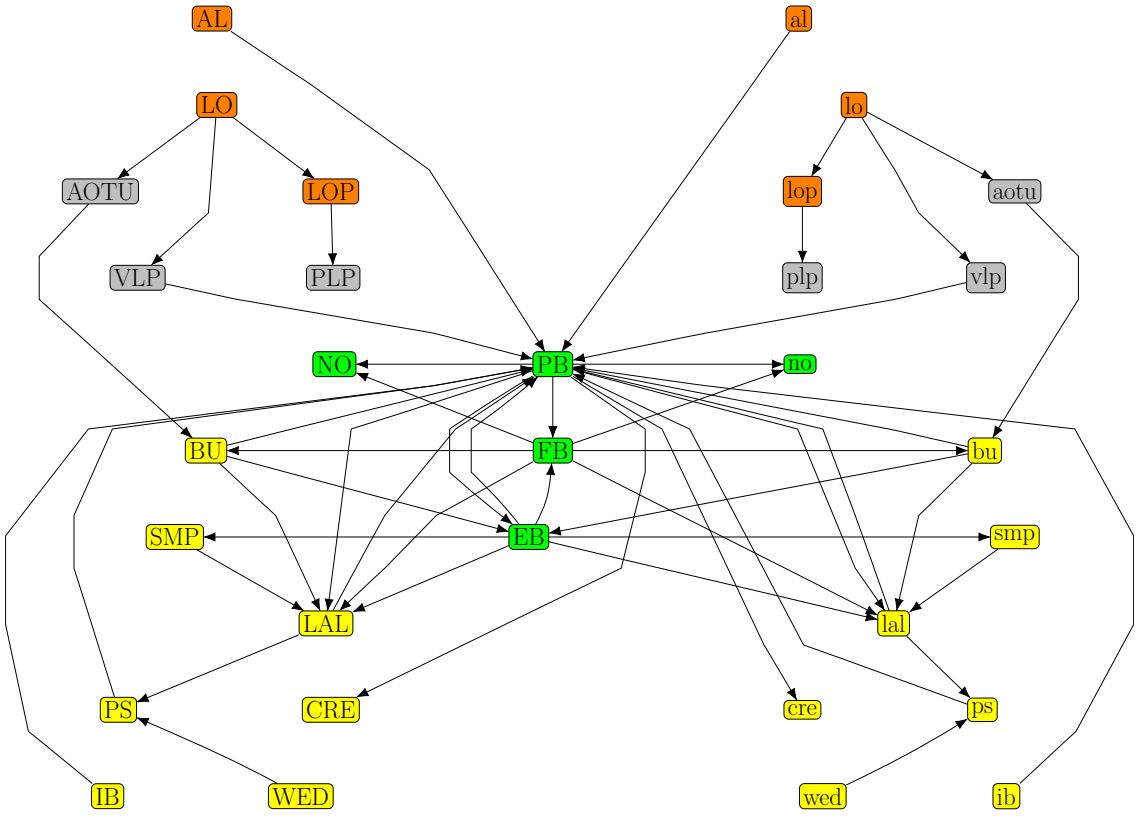


Figure 4.9: Information flow between CX neuropils (green), sensory neuropils (orange), neuropils that receive input from sensory neuropils (gray), and other accessory neuropils connected to the CX (yellow). Only known pathways are depicted.

Neuropil	**References**
FB	[54, p. 349, 352], [145, p. 1439]
PB	[82, p. 1743],[141, p. 1007]

Table 4.2: Identified local neurons in CX neuropils.

Spiking responses have been recorded from cells in FB during CX-related experiments using *Drosophila* [138, p. 64], from neurons supplying PB, tangential/pontine cells in FB, and ring cells in EB in *Neobellieria* [113], and from CX neurons in other insects [7].

4.5 Identified Neurons in the Central Complex

Local and projection neurons innervating the central complex can be classified into several families, most of which are characterized by unique arborization patterns. CX neuron families are listed in Tab. 4.2 and 4.3; known arborization patterns are described later in this section. In all neuropil innervation diagrams depicted below, arrow heads represent presynaptic arborizations and arrow tails represent postsynaptic arborizations.

4.5.1 Index of Identified Neurons

Neuron Family	Locations of Postsynaptic Arborizations (Dendrites)	Locations of Presynaptic Arborizations (Axons)	References
AL-PB	AL	PB	[54, p. 362, Fig. 24a]
BU-EB	BU	EB	[54, p. 352, Fig. 20a-d]
BU-LAL	BU	LAL	[54, Fig. 14]
DAL	SMP	LAL, SMP	[20, p. 680]
EB-FB	EB, FB	EB, FB	[54, Fig. 13a]
EB-FB-LAL-SMP	EB	FB, LAL, SMP	[54, p. 353, Fig. 21a-b]
EB-LAL-PB	EB	EB, LAL, PB	[82, Fig. 4J-M]
EB-NO	EB, NO	EB, NO	[54, Fig. 13b]
F	BU, FB, LAL, NO	FB, NO	[54, Fig. 14, 22, p. 353, 356]
FB-BU-LAL	FB	BU, LAL	[54, Fig. 14]
FB-NO	FB, NO	FB	[54, Fig. 11]
IB-LAL-PS-PB	IB, LAL, PS	PB	[82, p. 1743, Fig. 4A] [141, Fig. 3N]
PB-EB	PB	EB	[54, p. 350, Fig. 12b]
PB-EB-BU	PB	BU, EB	[54, Fig. 10b]
PB-EB-LAL	PB	EB, LAL	[82, Fig. 5E]
PB-EB-NO	PB	EB, NO	[82, p. 1745, Fig. 5G]
PB-FB	PB	FB, PB	[54, Fig. 12b]
PB-FB-CRE	PB	CRE, FB	[82, Fig. 6F] [141, Fig. 3L]
PB-FB-EB	EB, FB, PB	EB, FB	[54, Fig. 12e,f]
PB-FB-LAL	PB	FB, LAL	[82, Fig. 6F-H]
PB-FB-LAL-CRE	PB	CRE, FB, LAL	[141, Fig. 3M]
PB-FB-NO	PB	FB, NO	[82, p. 1746, Fig. 5L]
PS-IB-PB	IB, PS	PB	[141, Fig. 3S-T]
PS-PB	PS	PB	[141, Fig. 3R]
WED-PS-PB	PS, WED	PB	[82, p. 1744, Fig. 4B,D]

Table 4.3: Identified projection neurons connecting CX and accessory neuropils. Additional inputs from vision neuropils and AOTU to ATL, BU, LAL, PLP, PS, SMP, VLP and outputs to locomotion neuropils have also been observed [102, p. 939], [110, Fig. 6], [82], [103, p. 9].

Neuron Family	**Locations of Arborizations**	**References**
CC	LAL, NO	
EB-NO	EB, NO	[54, p. 351]
FB-EB	EB, FB	[54, p. 351]
FB-NO	FB, NO	[54, Fig. 11]

Table 4.4: Projection neurons connecting CX and accessory neuropils with unresolved neurite types.

4.5.2 Neurotransmitter Profiles

A range of neurotransmitters appear to be present in the CX neuropils (Tab. 1). Neurotransmitters associated with specific CX neural pathways have been identified (Tab. 2); however, the neurotransmitter associated with each specific arborization remains unclear.

4.5.3 Local Neurons

4.5.3.1 PB Local Neurons

Different studies of PB have identified 8 [141, p. 1007] or 10 [82, p. 1743] distinct local neurons. Tab. 3 and Fig. 4.10 assume the presence of 8 glomeruli on each side of PB as indicated by [141], that R2-R9 in [141] correspond to R1-R8 in [82], and that postsynaptic arborizations are spaced 7 glomeruli apart in all but the first 2 neuron types.

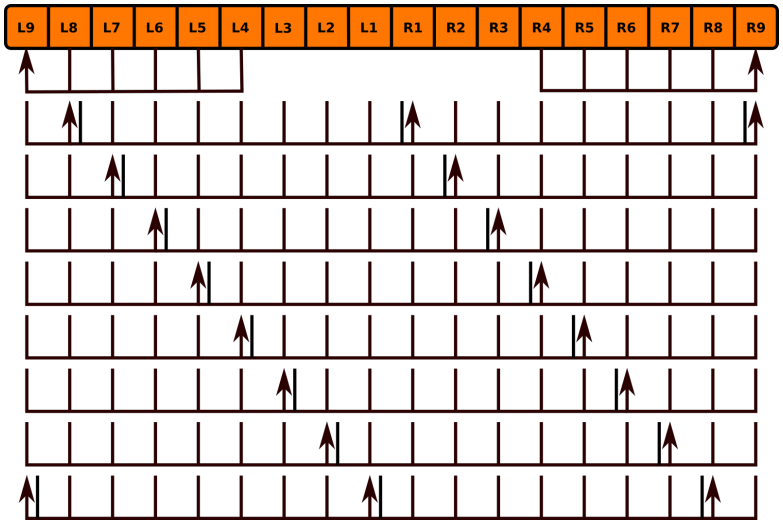


Figure 4.10: Innervation pattern of PB local neurons (Tab. 3).

4.5.3.2 FB Local Neurons

Several classes of local neurons referred to as pontine neurons have been observed to connect different regions of FB with each other [54, p. 349], [146, p. 1507]. One class (Tab. 5)

comprises symmetric neurons thatconnect each segment in one side of FB at the with each segment in the other side such that the presynaptic and postsynaptic arborizations are 4 segments apart [54, p. 352] (although more recent work suggests that each neuron might be a bundle of 2 neurons [145, p. 1439]). Judging by the structure of pontine neurons in other insects [55], arborizations might not be strictly confined to targeted regions. Other classes dorsoventrally connecting different layers in FB may exist, but they have not been systematically identified [54, p. 349]. It is unclear whether local neurons other than pontine neurons exist in FB.

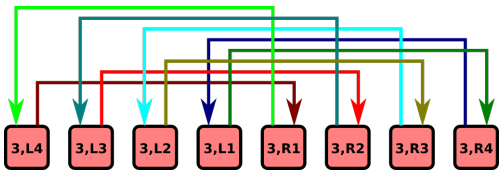


Figure 4.11: Innervation pattern of FB local neurons (Tab. 5).

4.5.3.3 EB Local Neurons

Although there appear to be local neurons in EB [22], they do not appear to have been systematically identified yet.

4.5.4 Projection Neurons

4.5.4.1 BU-EB Projection Neurons

Neurons with postsynaptic arborizations in BU and presynaptic arborizations in EB are typically referred to as ring or R neurons [54, p. 352] by virtue of the shape of their EB arborizations. 5 types of ring neurons (R1, R2, R3, R4m, R4d) have been observed [146, p. 1509]; specific ring neuron types appear to be essential to different visual behaviors [31, p. 120]. Each ring neuron type arborizes in a single microglomerulus [125, p. 262] and a different portion of the EB radius (Fig. 4.12); these types correspond to different sets of BU microglomeruli (and hence comprise multiple neurons). About 20 of each of these

types of neurons have been estimated in each hemisphere of the fruit fly brain [146, p. 1510]; combined with visual confirmation of the presence of 80 microglomeruli (§ 4.3.5), this suggests that there are 16 of each neuron type present in BU. Some ring neurons are GABAergic [82, p. 1750], while others are glutamatergic [82, Fig. 7C]; their synaptic connections to other neurons in EB therefore seem to be inhibitory. There is recent evidence that some ring neurons may be cholinergic and hence possess excitatory synapses [88, p. 1598]. Coincident synapses (i.e., those in which two independent presynaptic zones coincide with a single postsynaptic zone) have been observed in EB between ring neurons in specific domains and other neurons both in and outside of those domains. [88, p. 1592]; it seems that such synapses may also exist between other neurons that innervate EB [88, p. 1594]. Connections between AOTU and BU have been observed [103, p. 9]; these presumably constitute a pathway for input visual information from LO via AOTU [102, p. 939] to EB via ring neurons.

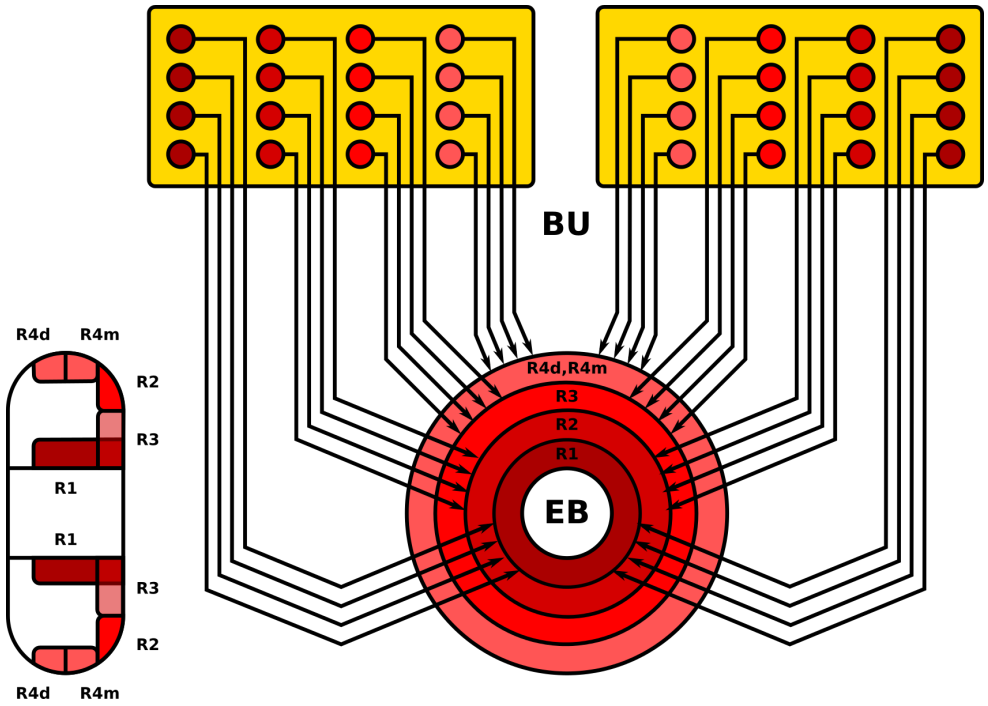


Figure 4.12: Spatial organization of ring neuron arborizations in BU microglomeruli (yellow) and regions in EB (red); posterior is left of sagittal section of EB, anterior is right. The depicted EB regions do not exactly correspond to the rings in Fig. 4.4c. Only a fraction of the microglomeruli/neurons are depicted.

4.5.4.2 EB-FB-LAL-SMP Projection Neurons

Neurons with presynaptic ring-shaped arborizations in EB and postsynaptic arborizations in other neuropils are referred to as extrinsic ring neurons. Two types (ExR1, ExR2) have been observed; these neurons appear to constitute a dopaminergic pathway [82, Fig. 7C]. ExR1 neurons are presynaptic in EB, FB, and LAL, and postsynaptic in SMP [54, p. 353] [VirtualFlyBrain]. ExR2 appears to have presynaptic arborizations in EB and arborizations in SMP, but the remainder of the neuron's structure has not been reconstructed.

4.5.4.3 EB-LAL-PB Projection Neurons

These neurons correspond to the EB-PB-VBO or EIP neurons in [82, Fig. 2]; they constitute a cholinergic pathway [82, Fig. 7C]. The neuron arborizations in Tab. 4 and Fig. 4.13 make

the assumption that the C, O, and P rings in [82] collectively correspond to the P and M shells in [141] and that the C and P rings collectively correspond to the P shell.

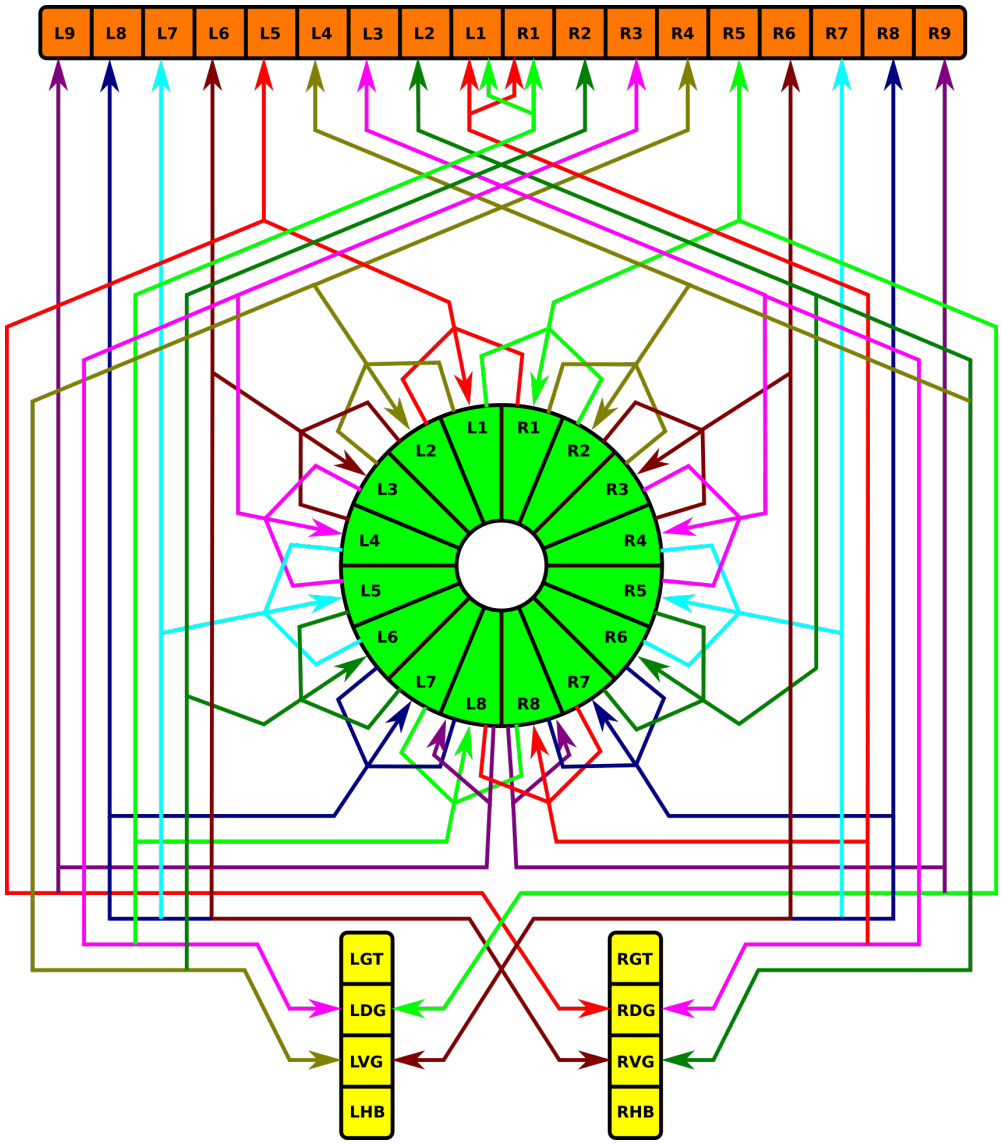


Figure 4.13: Neuropil innervation pattern for EB-LAL-PB neurons (Tab. 4).

4.5.4.4 F Projection Neurons

F neurons innervate entire layers of FB, with some types associated with specific layers [54, p. 353]. Fm neurons have bleb-like (presynaptic) arborizations in layer 2 of FB; Fm1 neurons

have spiny (postsynaptic arborizations in SLP or SIP, Fm2 neurons have spiny arborizations in LAL, and Fm3 neurons have spiny arborizations in ICL [54, p. 353]. Some Fl neurons have spiny branches in LAL [54, p. 354], while other Fl neurons have spiny branches that innervate the entire BU [54, p. 354].

4.5.4.5 IB-LAL-PS-PB Projection Neurons

These neurons corresponds to the CVLP-IDFP-VMP-PB or CIVP neurons in [82, Fig. 2] and the PB-LAL-PS neurons in [141, Fig. 3N]. They receive indirect input from the vision system and innervate all glomeruli in PB indicated in Tab. 6:

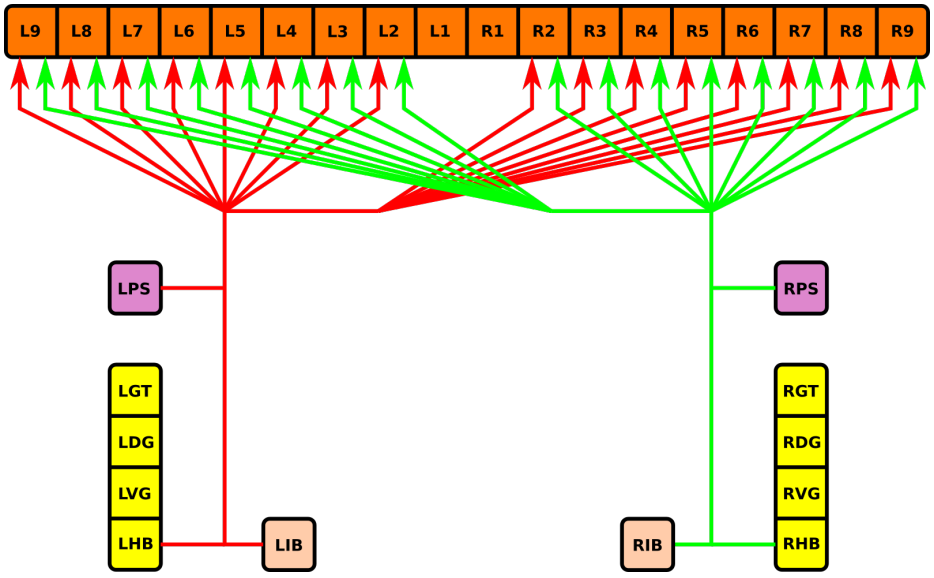


Figure 4.14: Neuropil innervation pattern for IB-LAL-PS-PB neurons (Tab. 6).

4.5.4.6 PB-EB-BU Projection Neurons

Neurons with postsynaptic arborizations in PB and presynaptic arborizations in EB and BU have been observed. These correspond to the PB-EB-LTR neurons in [54, Fig. 10b]. Details regarding possible neuron types have not been determined.

4.5.4.7 PB-EB-NO Projection Neurons

Neurons with postsynaptic arborizations in PB and presynaptic arborizations in EB and NO are referred to as PEN neurons in [82, p. 1745]. They constitute a cholinergic pathway [82, Fig. 7C]. The connections in Tab. 8 and Fig. 4.15 are based upon [141, Fig. 14a], which differ from those described in [82, Fig. 5g].

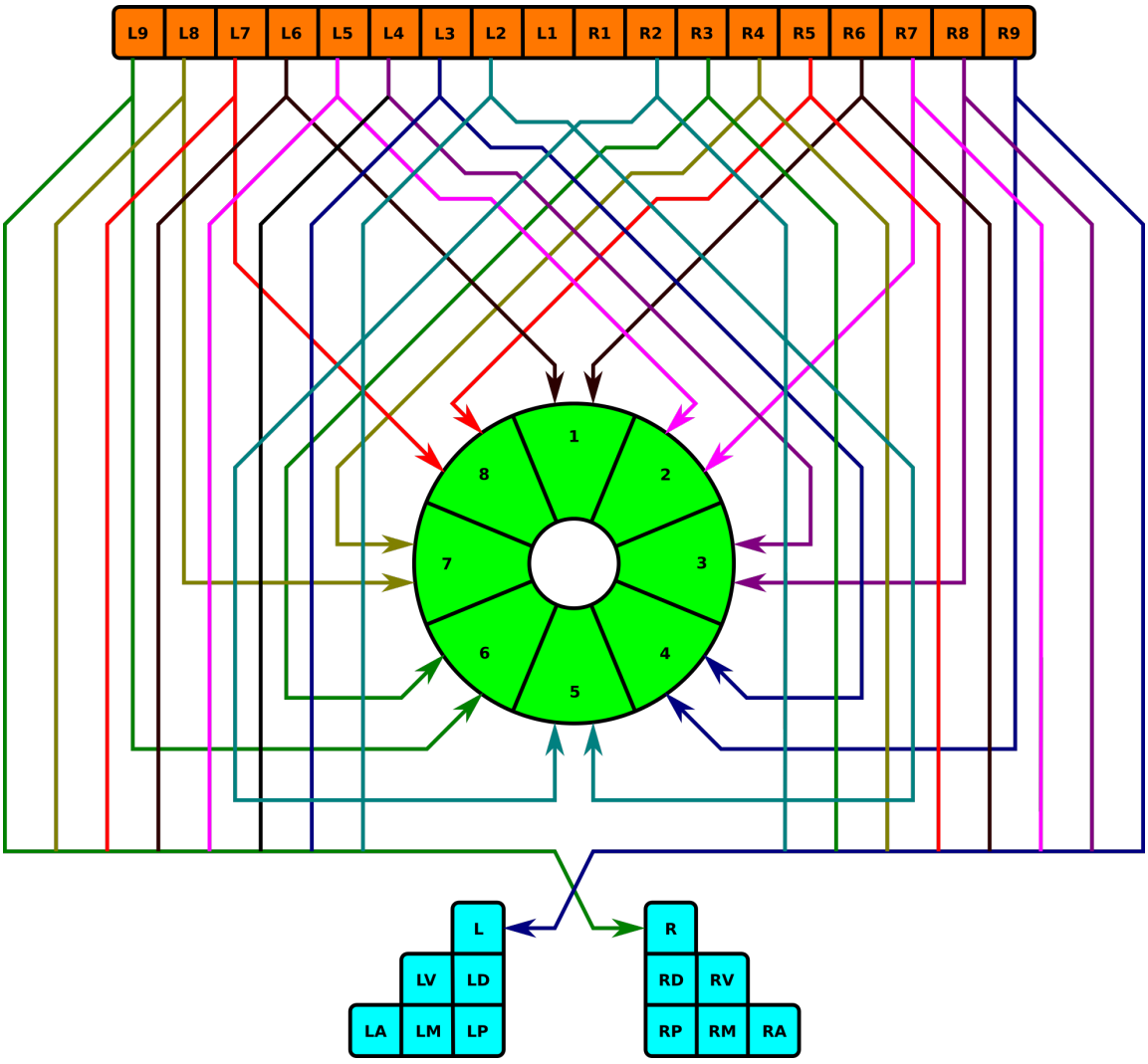


Figure 4.15: Neuropil innervation pattern for PB-EB-NO neurons (Tab. 8).

4.5.4.8 PB-EB-LAL Projection Neurons

These neurons have postsynaptic arborizations in PB and presynaptic arborizations in EB and LAL; they correspond to PB-EB-IDFP or PEI neurons in [82, Fig. 2]. These neurons constitute a cholinergic pathway [82, Fig. 7C]. The connections in Tab. 7 and Fig. 4.16 are based upon [141, Fig. 14b], which differ from those described in [82, Fig. 5e].

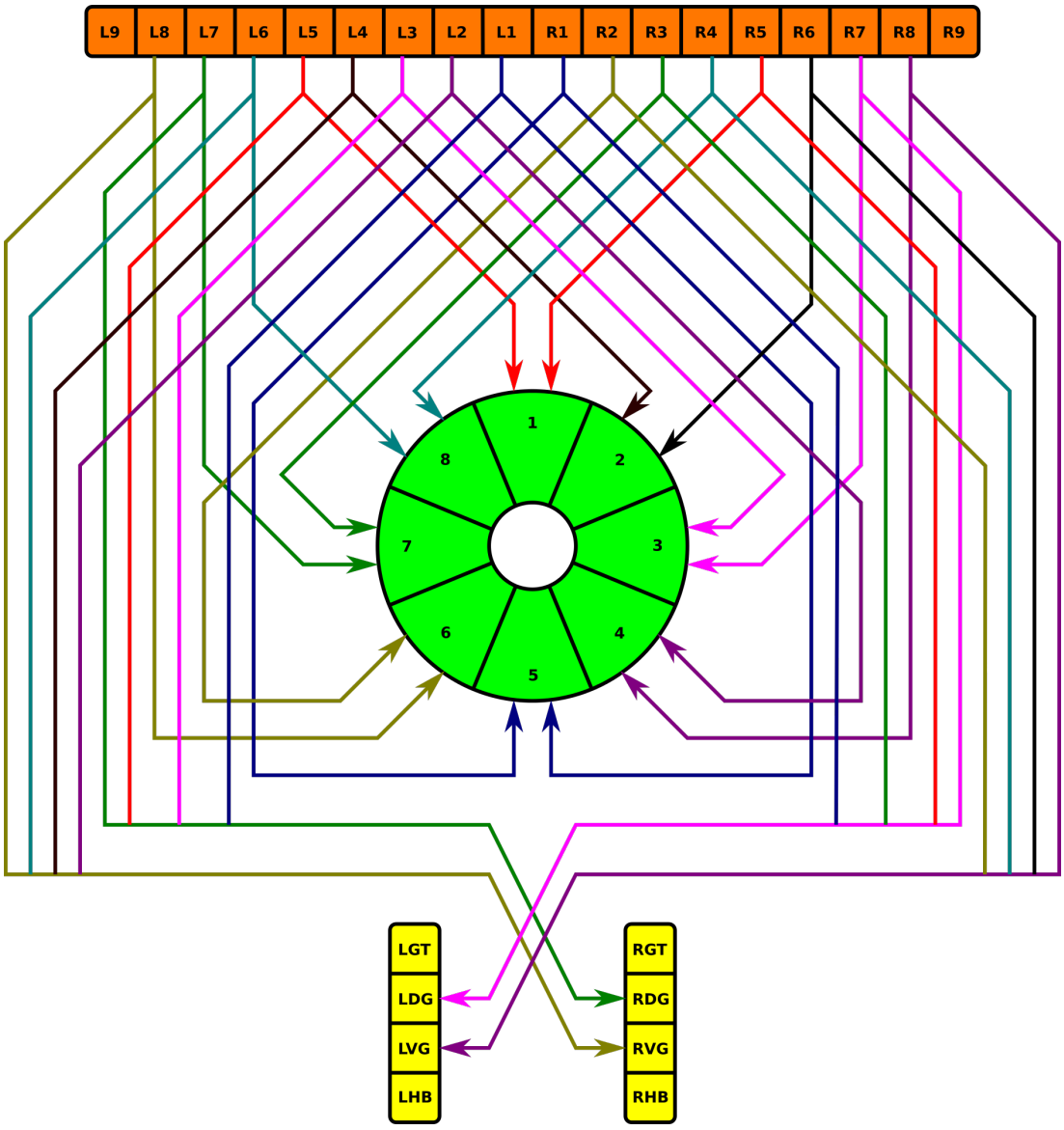


Figure 4.16: Neuropil innervation pattern for PB-EB-LAL neurons (Tab. 7).

4.5.4.9 PB-FB-CRE Projection Neurons

The neurons in Tab. 9 and Fig. 4.17 correspond to the PB-FB-VBO or PFI neurons that connect to RB in [82, Fig. 2]. They constitute a cholinergic pathway [82, Fig. 7C].

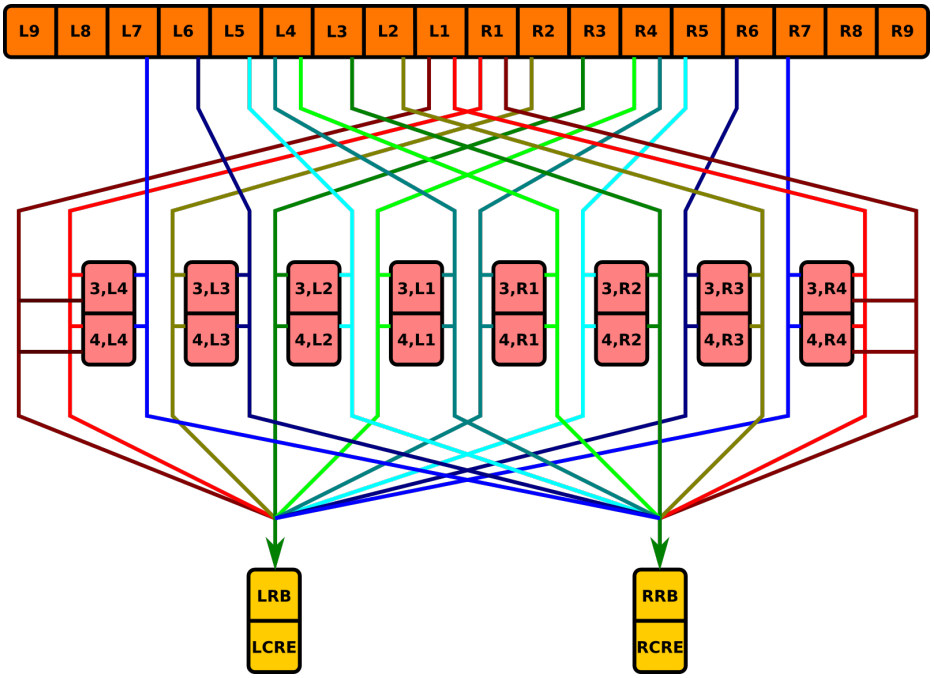


Figure 4.17: Neuropil innervation pattern for PB-FB-CRE neurons (Tab. 9).

4.5.4.10 PB-FB-NO Projection Neurons

Neurons with postsynaptic arborizations in PB and presynaptic arborizations in FB and NO are referred to as the vertical fiber system [54, Fig. 5b]; they correspond to the PFN neurons described in [82, p. 1745]. These neurons constitute a cholinergic pathway [82, Fig. 7C]. The 5 sets of PB-FB-NO neurons are listed in Tabs. 10, 11, 12, 13, and 14 and depicted in Fig. 4.18.

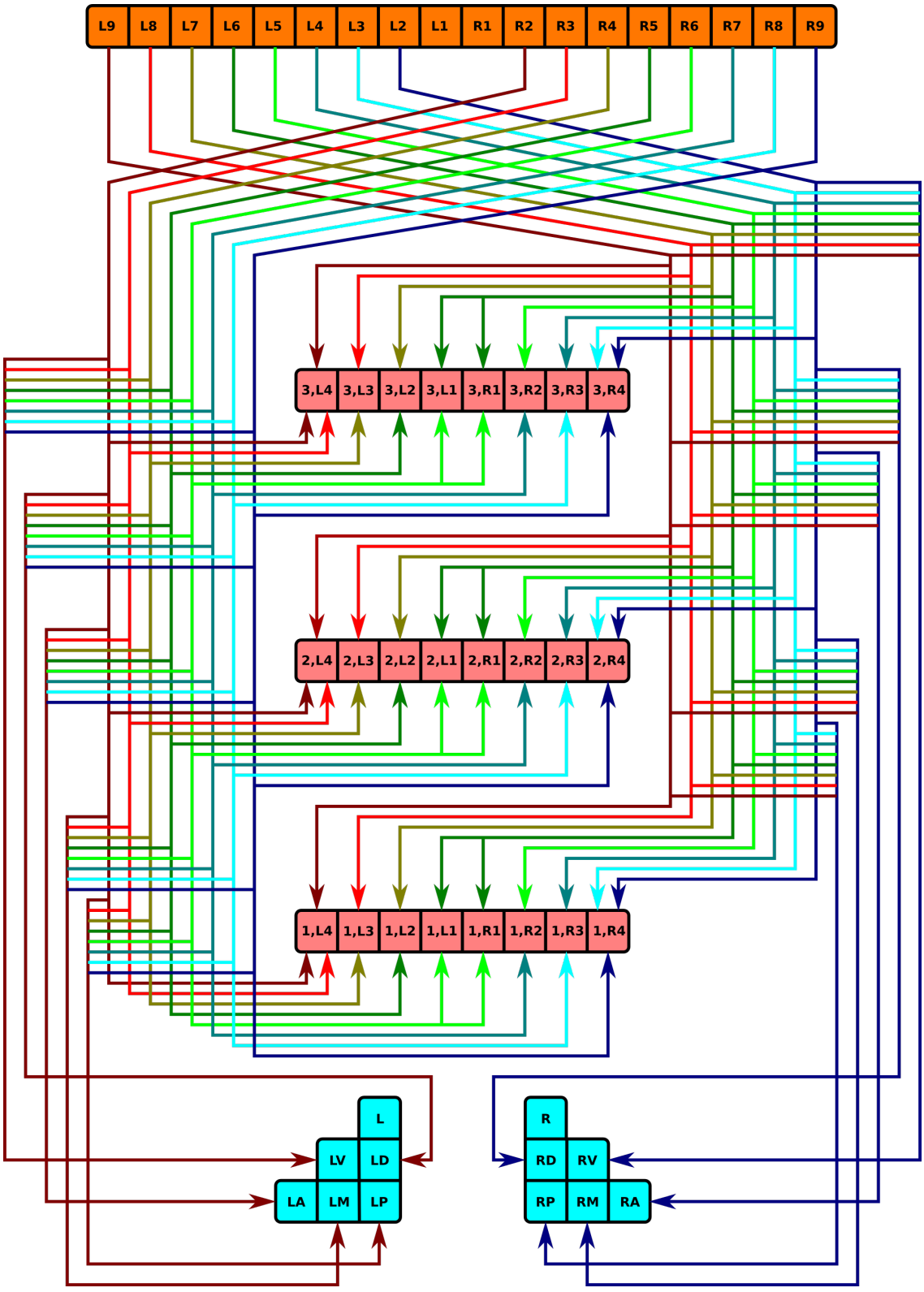


Figure 4.18: Neuropil innervation pattern for PB-FB-NO neurons (Tabs. 10, 11, 12, 13, 14).

4.5.4.11 PB-FB-LAL Projection Neurons

Neurons with postsynaptic arborizations in PB and FB and presynaptic arborizations in LAL are referred to as PFI neurons in [82, p. 1745]. They correspond to the PB-FB-VBO or PFI neurons that connect to HB in [82, Fig. 2], and are also referred to as the horizontal fiber system [54, Fig. 6b]. [1] These neurons constitute a cholinergic pathway [82, Fig. 7C].

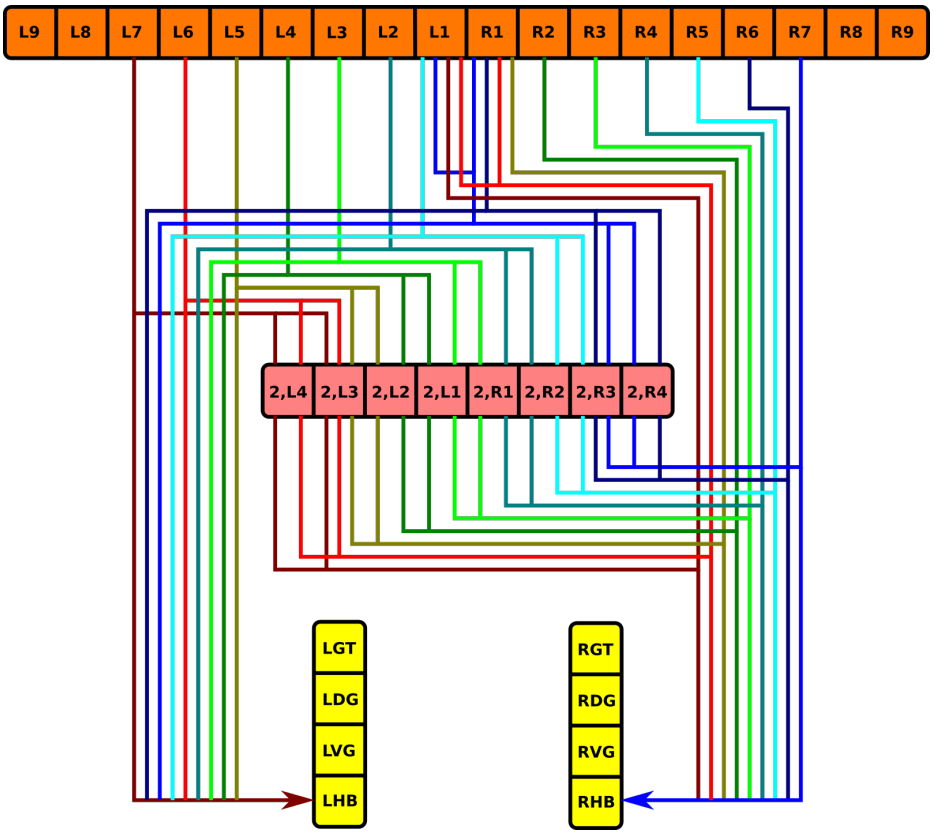


Figure 4.19: Neuropil innervation pattern for PB-FB-LAL neurons innervating layer 2 of FB (Tab. 15).

[1] These sources appear to consider CRE as part of LAL; this document treats CRE separately.

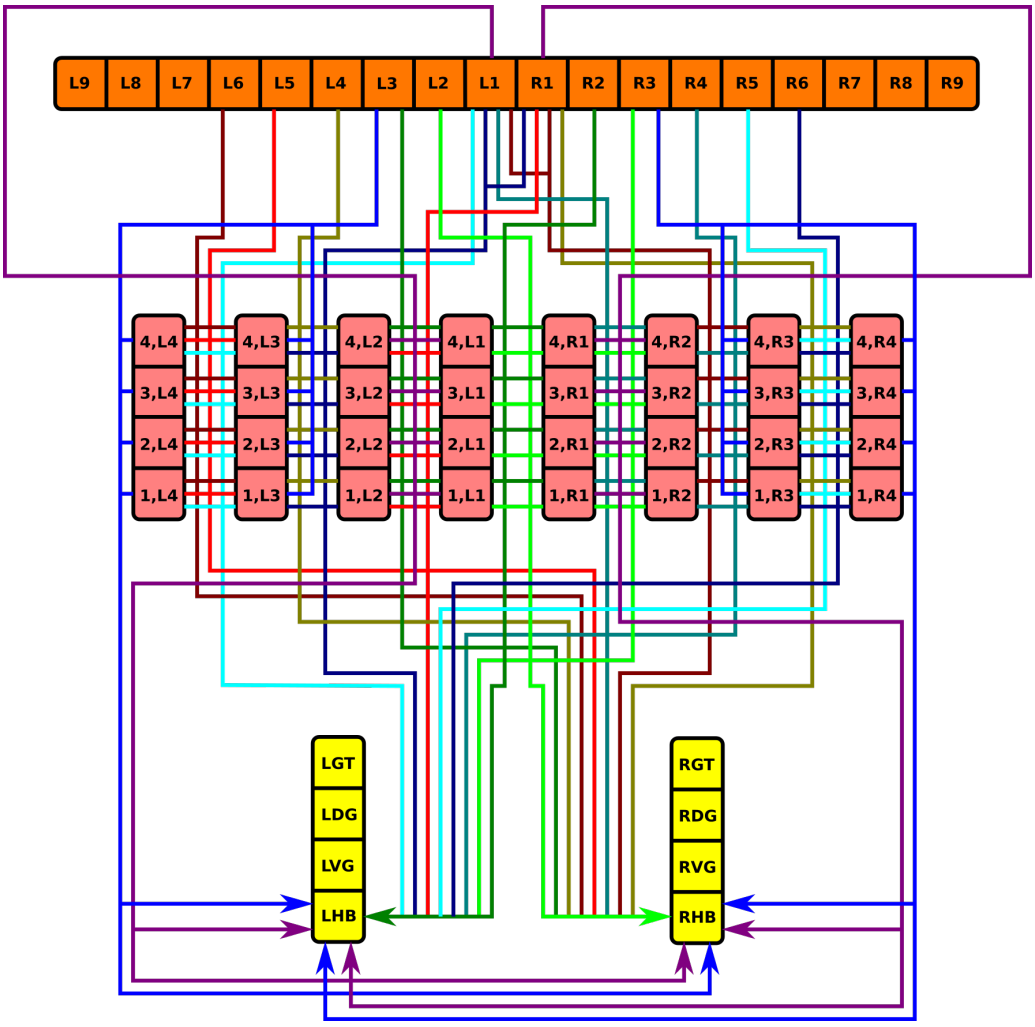


Figure 4.20: Neuropil innervation pattern for PB-FB-LAL neurons (Tab. 16, 17).

4.5.4.12 WED-PS-PB Projection Neurons

The neurons in Tab. 18 correspond to CCP-VMP-PB or CVP neurons in [82, Fig. 2] and receive input from the vision system. They constitute a cholinergic pathway [82, Fig. 7C].

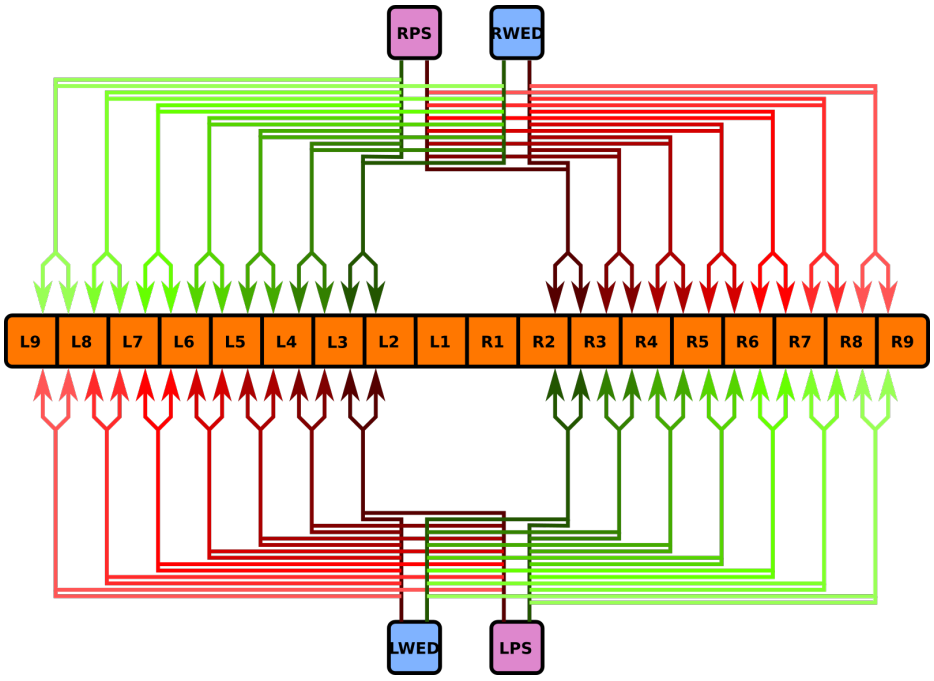


Figure 4.21: Neuropil innervation pattern for WED-PS-PB neurons (Tab. 18).

4.6 Generating an Executable Circuit Model

4.6.1 Neuron Organization

In light of the current lack of data regarding synapses between the various neurons identified in the central complex neuropils, data regarding the arborizations of these neurons was used to infer the presence or absence of synapses to generate an executable model of the central complex. Local and projection neurons were assigned to LPUs as indicated in Tab. 4.5.

LPU	Neuron Families
BU, bu	BU-EB
EB	EB-LAL-PB
FB	FB local
PB	PB local, PB-EB-NO, PB-EB-LAL, PB-FB-CRE, PB-FB-NO, PB-FB-LAL, WED-PS-PB, IB-LAL-PS-PB

Table 4.5: Assignment of neuron families to LPUs in generated CX model.

Although the BU-EB neurons have not been systematically characterized, available information regarding these neurons (§ 4.5.4.1) was used to hypothesize the arborization structure for a total of 80 neurons in each hemisphere of the fly brain (Tab. 19). Likewise, we also hypothesized isomorphic sets of pontine neurons that link the following regions in FB based upon [54, p. 349]:

- nonadjacent segments in layers 1, 2, 4, and 5 (Tab. 20);
- adjacent segments within the same layer in layers 1-5 [54, Fig. 9a] (Tab. 21).
- adjacent layers within the same segment for layers 1-5 [54, Fig. 9a] (Tab. 22).
- nonadjacent layers within the same segment, with both presynaptic and postsynaptic terminals in each layer [54, Fig. 9b] (Tab. 23); based upon the latter source, we assume two sets of neurons connecting layers 1 and 8 and 2 and 7, respectively.

4.6.2 Executable Circuit Generation

To infer the presence of synaptic connections between neurons, the above neuron names were loaded into a NeuroArch database [46] in accordance with its data model. Using a parser for the grammar described in § 4.2.2, each neuron's name was parsed to extract its constituent arborization records (Tab. 4.6); these records were reinserted into the database as `ArborizationData` nodes and connected to the `Neuron` nodes created for the neuron in each family listed above.

Field	Data Type	Sample Values
neurite	set of 'b' or 's'	[b], [b, s]
neuropil	string	PB, EB
region	set of strings or tuples	[L1], [(1, R1)]

Table 4.6: Fields in `ArborizationData` node. Region strings or tuples conform to the formats described in § 4.3.

After extraction of arborization data, all pairs of neurons in the database were compared to find those pairs with geometrically overlapping arborizations and differing neurite types

(i.e. presynaptic versus postsynaptic). This resulted in the creation of `Synapse` nodes that were connected to the associated `Neuron` node pairs in NeuroArch's database.

To illustrate the synapse inference process, consider the neurons EB/([R3,R5],[P,M],[1-4])/s-EB/(R4,[P,M],[1-4])/b-LAL/RDG/b-PB/L3/b (Tab. 4) and PB/L4/s-EB/2/b-LAL/RVG/b (Tab. 7). Since the region EB/(R3,P,[1-4])/s overlaps with region EB/2/b (Tab. 4.1) and the terminal types of the two neurons in the overlapping region differ, we infer the presence of a synapse with information flow from the latter neuron to the former.

Neuron and synapse models were instantiated for each respective biological neuron and synapse in NeuroArch's database to construct LPUs corresponding to the BU, FB, EB, and PB neuropils. All neurons were modeled as Leaky Integrate-and-Fire neurons, and all synapses modeled to produce alpha function responses to presynaptic spikes. Since the goal of this model construction was to demonstrate algorithmic generation of an executable circuit rather than replicate a specific observed pattern of activity in the corresponding biological circuit, neuron and synapse parameters were set to ensure that some responses were elicited in response to the described inputs but were not otherwise tuned. Communication ports were created for every neuron model instance comprised by one LPU connected to a synapse model instance in the other neuropil; the connectivity pattern linking the ports associated with the neuron and synapse models was also added to the NeuroArch database. NeuroArch's API was used to extract the constructed LPUs and patterns and dispatch them to Neurokernel [44] for execution.

4.6.3 Executing the Circuit

To test the executability of the generated circuit and its ability to respond to input data, the generated model was driven by a simple visual stimulus consisting of an illuminated vertical bar proceeding horizontally across the 2D visual space (Fig. 4.22). Since the central complex neuropils do not receive direct connections from the vision neuropils, the visual stimulus was passed into three banks of receptive fields whose outputs were respectively provided to BU,

bu, and PB as input (Fig. 4.23). The receptive fields for BU and bu each consisted of 80 evenly spaced 2D grids of circular Gaussians that correspond to one of the microglomeruli in BU; each receptive field was connected to one BU-EB neuron such that the 16 neurons in each of the 5 groups described in § 4.5.4.1 processed input from a rectangle occupying $\frac{1}{5}$ of the 2D visual space. The receptive fields for PB consisted of 18 vertical rectangular regions with a constant magnitude; each receptive field was connected to all local and projection neurons that innervated the glomerulus corresponding to the receptive field region. The responses of the neurons in each family to the two input signals are organized in the same order in the respective raster plots.

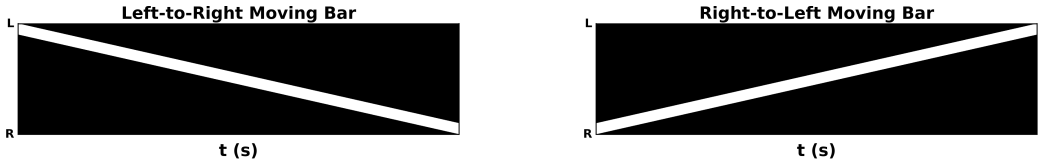


Figure 4.22: Moving bar visual input to generated CX model. The plots depict the movement of an illuminated vertical bar horizontally across a dark background.

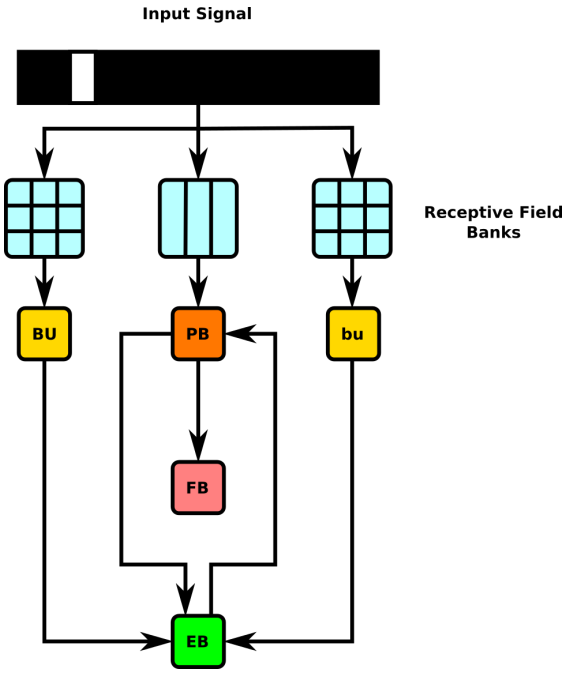


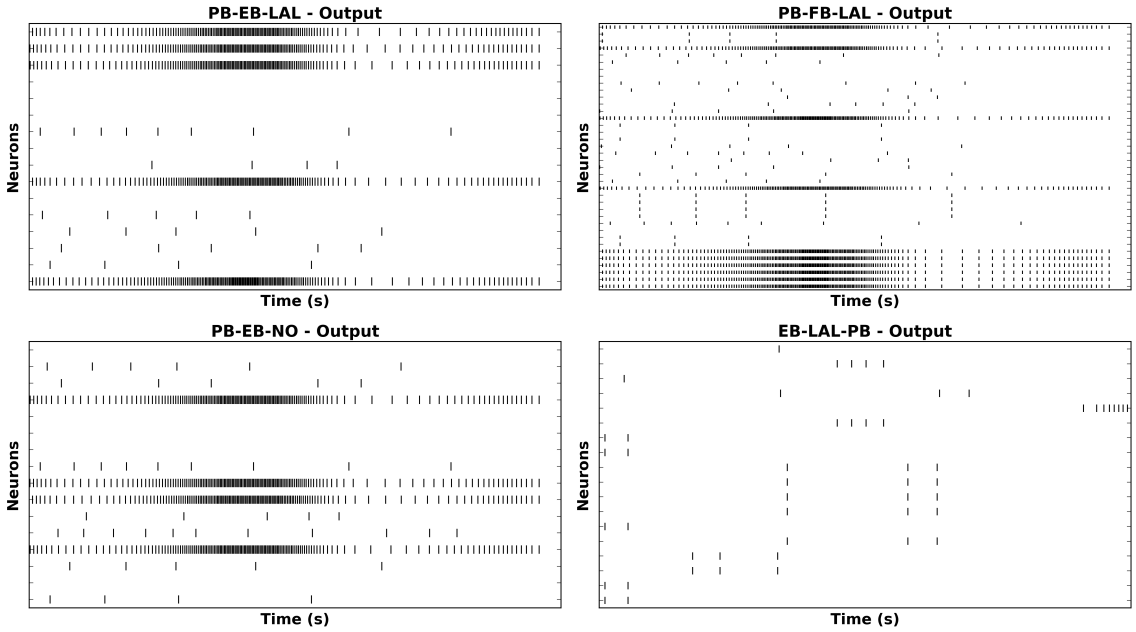
Figure 4.23: Schematic of information flow in generated CX model. 2D visual signals are passed through rectangular grids of Gaussian receptive fields whose outputs drive BU-EB neurons and through a bank of vertical rectangular receptive fields whose outputs drive neurons that innervate the PB glomeruli. The generated model only comprises neurons that innervate the depicted LPUs (BU, bu, EB, FB, and PB).

4.6.4 Use Cases

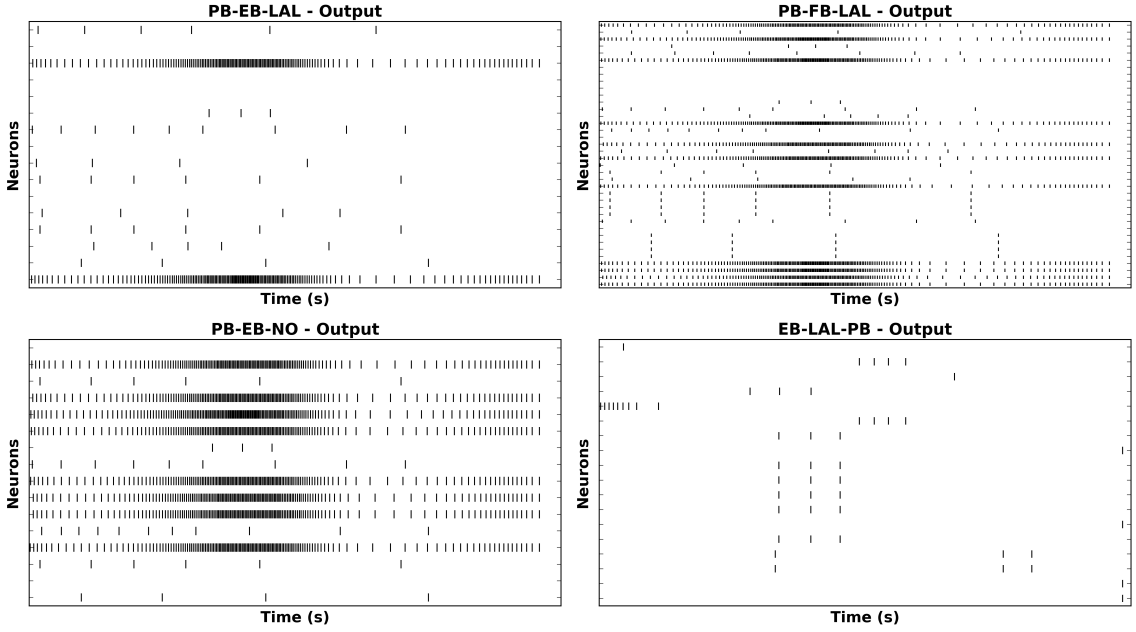
The NeuroArch and Neurokernel pipeline used to generate the CX model described above enables analysis and manipulation of the model using computational analogues to experimental techniques.

4.6.4.1 Virtual Electrophysiology

One can use NeuroArch/Neurokernel to concurrently probe the responses different sets of neurons in multiple neuropils in a computational experiment. Figs. 4.24 and 4.25 depict the responses of neurons innervating the PB and BU/bu neuropils to the signal depicted in Fig. 4.22.

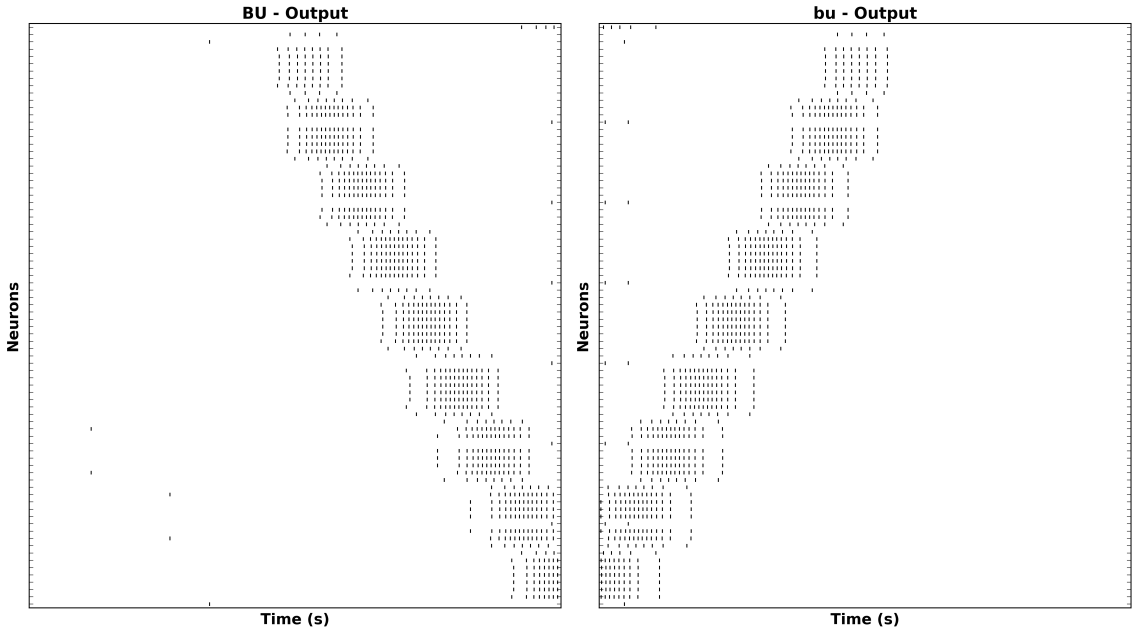


(a) Response to bar moving left to right.

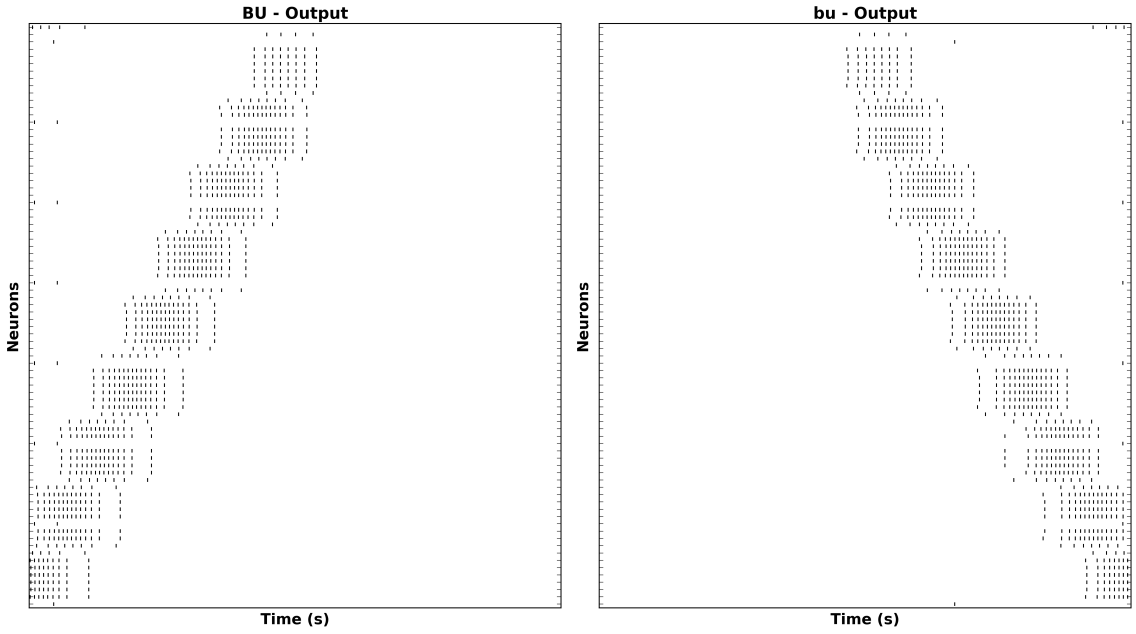


(b) Response to bar moving right to left.

Figure 4.24: Response of CX projection neurons innervating PB to moving bar input.



(a) Response to bar moving left to right.



(b) Response to bar moving right to left.

Figure 4.25: Response of CX projection neurons innervating BU/bu to moving bar input.

4.6.4.2 Virtual Genetic Manipulation

To test hypotheses regarding incompletely characterized parts of the fly brain, one can create models that either attempt to replicate abnormal behaviors or emulate abnormal circuit structures observed in different mutant fly strains. For example, one can attempt to model phenotypes corresponding to mutations affecting the structure of PB (e.g., *no bridge*, *tay bridge*, etc.) by altering the PB model generation process accordingly. Given that these mutations are known to alter the fly's step length [132, p. 7] and since neurons innervating the motor ganglia are known to be postsynaptic to those that innervate LAL, it is reasonable to expect that analogous modifications to the structure of PB may alter the observed output of CX projection neurons that innervate LAL.

We used NeuroArch to emulate the *no bridge* mutant by altering the PB local neurons to remove all local connections between the left and right sides of PB and positing the existence of additional local neurons caused by the mutation (Fig. 4.26); the synapse inference algorithm was then run on the modified database to construct a mutant CX model. Although descriptions of the *no bridge* mutant suggest that several of the medial glomeruli are not present, our model does not alter any of the other known neurons in CX. The effects of the mutation on the response of the PB projection neuron families can be observed by comparing the mutant model output in Fig. 4.27 to Fig. 4.24. As the BU-EB neurons do not receive any input from other neurons in the generated model, their responses in the mutant model (Fig. 4.28) are identical to those in the original model.

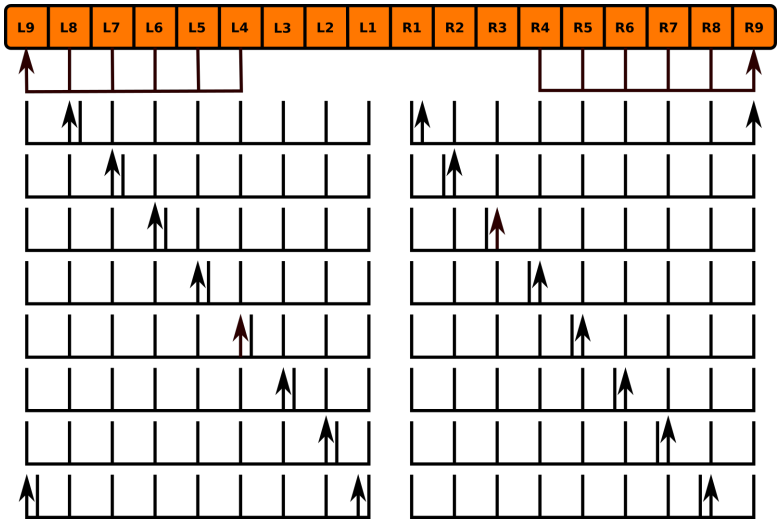
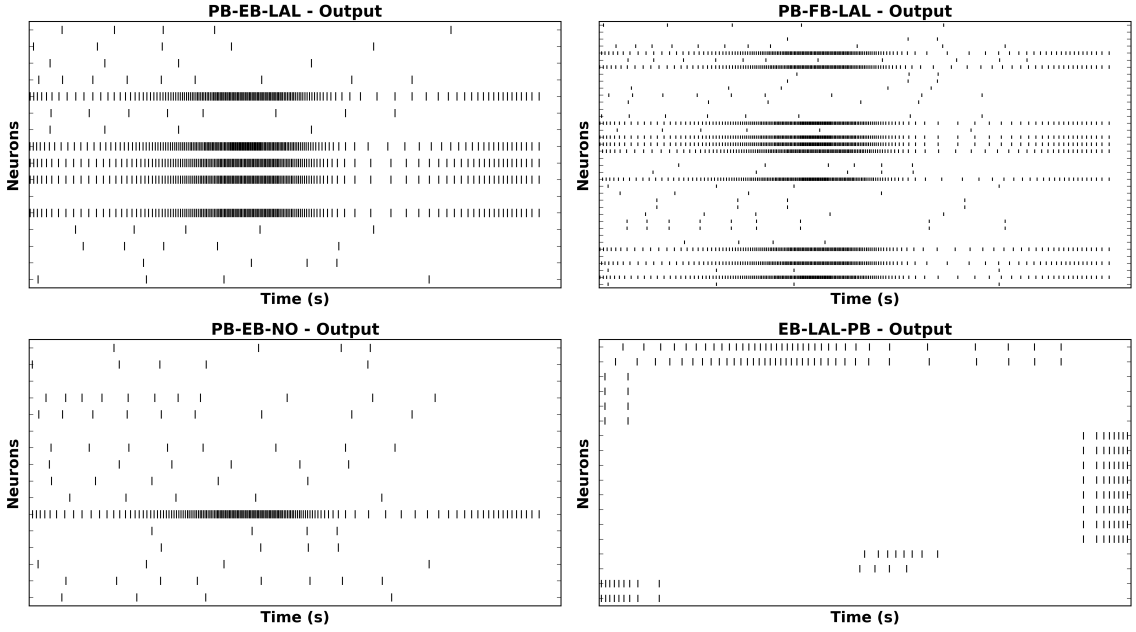
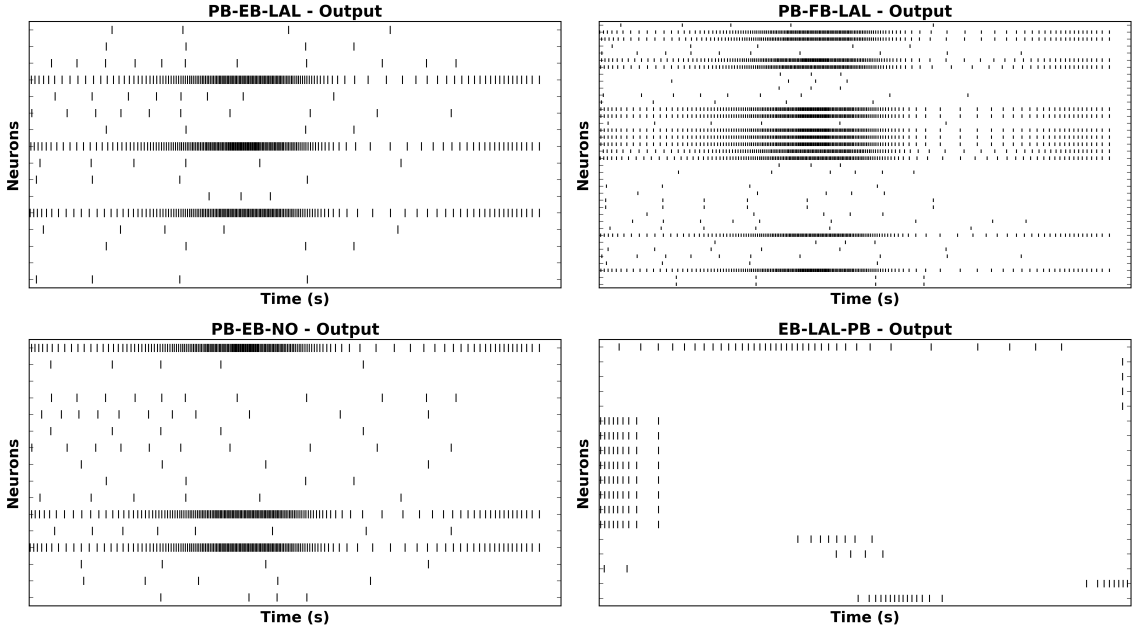


Figure 4.26: Hypothesized innervation pattern of PB local neurons in *no bridge* mutant (Tab. 24).

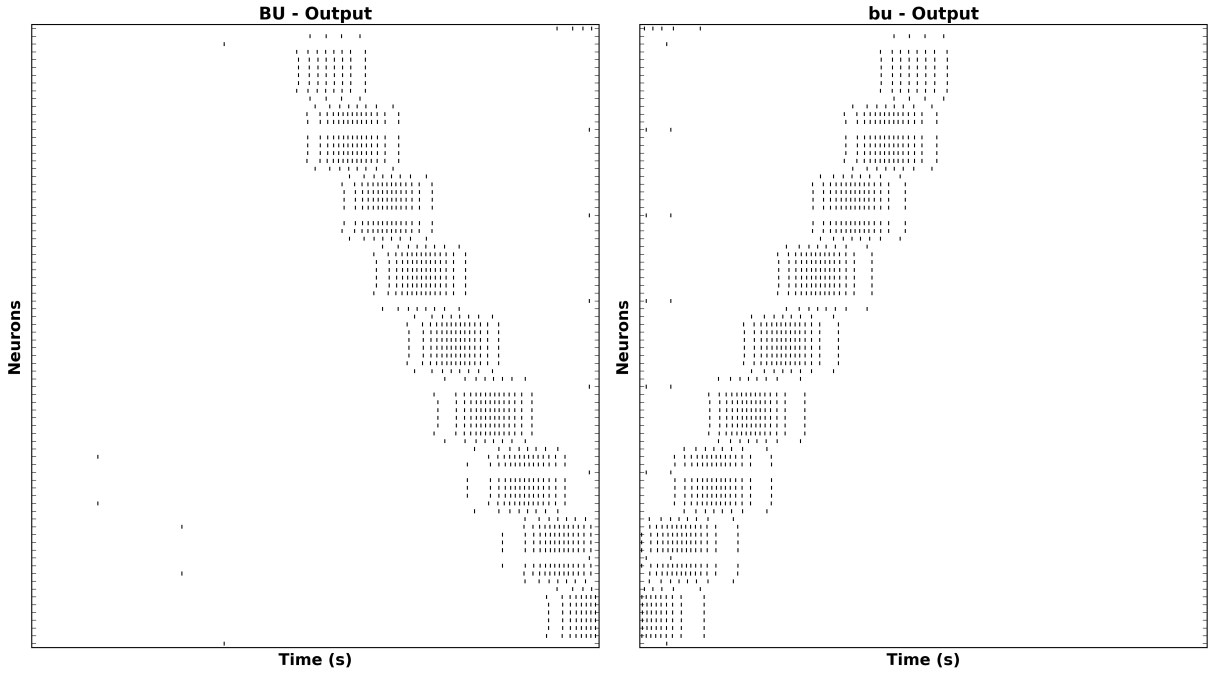


(a) Response to bar moving left to right.

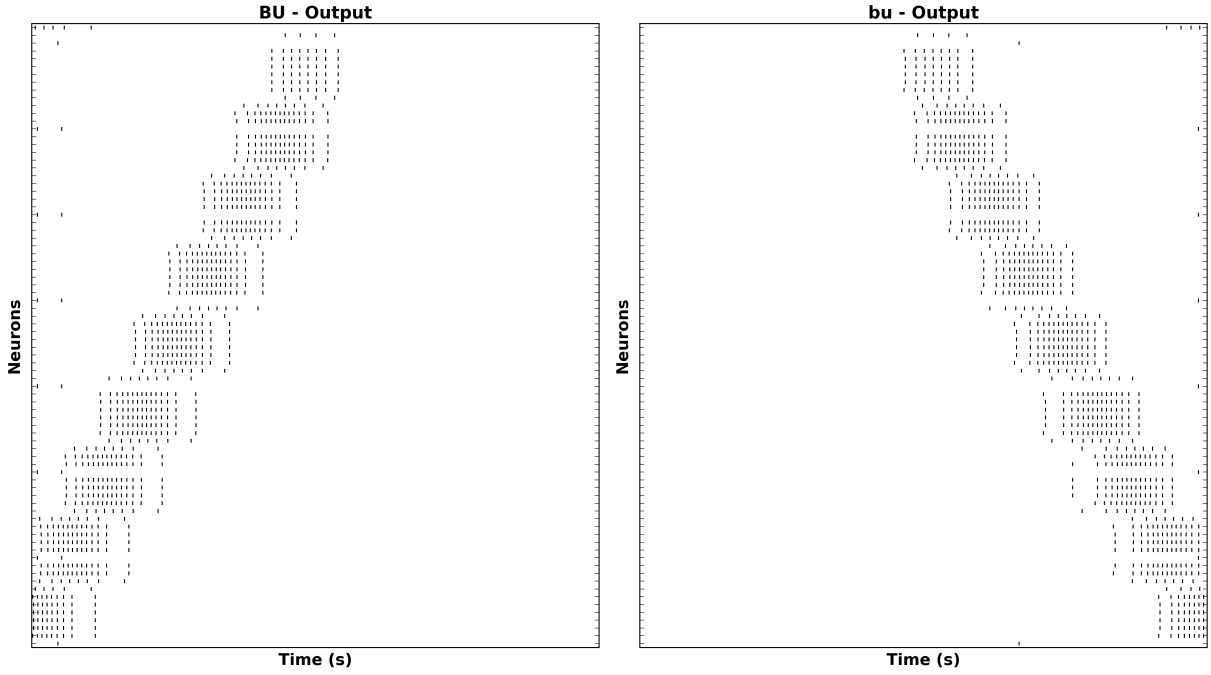


(b) Response to bar moving right to left.

Figure 4.27: Response of CX projection neurons innervating PB in constructed *no bridge* mutant CX model to moving bar input.



(a) Response to bar moving left to right.



(b) Response to bar moving right to left.

Figure 4.28: Response of CX projection neurons innervating BU/bu in constructed *no bridge* mutant CX model to moving bar input.

4.7 Related Work

In contrast to the neuropils known to be part of the fly's vision, audition, and olfaction systems, relatively few functional models of the spatial memory and locomotor control processing features of the fruit fly CX have been proposed or implemented. This dearth of models is reflective of the greater difficulty of characterizing the inputs of the CX neuropils compared to those of neuropils that directly receive sensory signals. One recent model speculated how neurons that transmit data from PB to FB could be involved in altering attractive and aversive responses to objects in the visual field, as well as how neurons that transmit data between PB and EB could play a role in short-term memory of object positions [132]. This model, however, does not account for several significant neural pathways such as that between BU and EB that are known to be essential to specific CX functions, and does not provide any executable hypothesis to confirm the putative roles of the neurons in specific functions attributed to CX.

The circular geometry of the EB neuropil has prompted comparisons with models of directional encoding that employ ring attractors. In light of the apparent role EB plays in spatial memory formation, a ring attractor network based upon the structure of EB was implemented that is capable of producing a hill of activity in a ring of neurons that tracks the estimated location of a visual target even when that target is briefly obscured; this behavior could explain how spatial information needed to enable detour behavior observed in live flies is represented in its brain [2]. While it does account for the observed presence of distinct excitatory and inhibitory pathways that innervate EB, the model does not address the interplay between the multiple inputs into EB from different neuropils (BU, PB) that receive data from the fly's vision system.

4.8 Summary

The many gaps in knowledge regarding both the internal circuitry of the CX neuropils and the nature of the inputs they receive from other brain regions virtually guarantees that making further progress in understanding how the CX implements its various functions will involve repeated redesign and modification of computational models. Advances in in vivo imaging of CX neuron activity have already begun to provide valuable detailed data on the responses of specific families of neurons in the CX neuropils [125, 126] that must be accounted for in hypotheses regarding CX circuit functions. The ability of the NeuroArch/Neurokernel pipeline to enable computational analogues of parallel recordings and targeted structural manipulation of the CX circuitry empowers researchers to translate these highly successful experimental paradigms from a biological context to the realm of circuit model design.

Chapter 5

Conclusions and Future Research Directions

5.1 Conclusions

5.1.1 When to Use the Pipeline

As indicated in § 2.5, the NeuroArch/Neurokernel pipeline aims to prioritize the programmability required to collaboratively build fly brain models over the optimization of model execution performance or extensive support for arbitrarily detailed neuron and synapse models. The pipeline specifically targets models of the brain expressed in terms of its constituent functional processing units, which in turn currently must be implemented in terms of neuron and synapse models supported by Neurodriver. Efforts to emulate the brains of other model organisms that need the concerted abilities of multiple researchers to succeed can benefit from utilizing this pipeline to modularize their brain modeling goals, although performance considerations will need to be reassessed for model organisms with more complex brains than the fruit fly. Conversely, neuroscientists who wish to study the properties of new neuron/synapse models not currently supported by Neurodriver, develop isolated circuit models that do not need to communicate with other circuits, or utilize non-neural concepts such as

mean field models may find existing simulators such as those detailed in § 2.4 preferable for their purposes. The Neurokernel/NeuroArch pipeline and other neuronal network simulators should therefore be regarded as complementary tools in the computational neuroscientist's arsenal.

5.1.2 Summary

The successful emulation of the fruit fly brain is an ambitious goal that requires the joint efforts of neurobiologists and computational researchers to succeed. To further this end, we have presented an open pipeline for fly brain model construction and execution that (1) enables multiple fruit fly researchers to combine their individual circuit design efforts into executable comprehensive models of the fly brain, and (2) enables the construction of fly brain models by generation of executable circuits from structured biological data rather than by explicitly specifying the structure of the executable circuit directly. We have demonstrated the power of this pipeline by using it to successfully integrate models of the fly retina and lamina into a working partial model of the fly vision system and to generate executable models of neuropils in the central complex from incomplete biological data regarding the neurons in those neuropils.

5.2 Neurokernel - Future Development

5.2.1 Automating Computational Resource Allocation

Although Neurokernel currently permits brain models to make use of multiple GPUs, it requires programmers to explicitly manage the GPU resources used by a model's implementation. Given that a functional API for building and interconnecting LPUs within Neurokernel's application has been obtained (§ 2.2.4), the next major goal is to implement a prototype GPU resource allocation mechanism within the control plane to automate selection and management of available GPUs used to execute a fly brain model. Direct access

to GPUs will also be restricted to modeling components implemented by LPU developers and added to Neurokernel's compute plane; models implemented or defined in the application plane will instantiate and invoke these components. These developments will permit experimentation with different resource allocation policies as LPU models become more complex to account for a greater level of biological detail. Restricting parallel hardware access to modeling components exposed by the compute plane will also facilitate development of future support for other parallel computing technologies such as non-NVIDIA GPUs or neuromorphic hardware.

5.2.2 Accelerated Neural Model Execution Engine

As simulator engines incorporate more extensive support for running sophisticated neural models on GPUs such as those afforded by Brian2GeNN or Myriad (§ 2.4.2), basing Neurokernel's compute plane upon them will enable Neurokernel to benefit from the extensive performance optimizations obtained by these packages while proving support for the collaborative programming model needed to develop a model of the entire fruit fly brain that general-purpose simulators lack. This development will hopefully enable Neurokernel's modular approach to modeling an entire brain to be scaled up to organisms with larger brains than the fruit fly in the future as new parallel computing technology becomes available.

5.2.3 In Vivo Model Validation

Efforts at reverse engineering the brain must ultimately confront the need to validate hypotheses regarding neural information processing against actual biological systems. In order to achieve biological validation of the Neurokernel, the computational modeling of the fruit fly brain must be tightly integrated with increasingly precise electrophysiological techniques and the recorded data evaluated with novel system identification methods [68, 69, 76, 75, 81, 77, 78]. This will enable direct comparison of the output of models executed by Neurokernel to that of corresponding neurons in the brain regions of interest. Given

that recently designed GPU-based systems for emulating neuronal networks of single spiking neuron types have demonstrated near real-time execution performance for networks of up to $\sim 10^5$ spiking neurons and $\sim 10^7$ synapses using single GPUs [97, 39, 117, 10], and in light of advances in the power and accessibility of neuromorphic technology [30, 115, 8, 90, 21], we anticipate that future advances in parallel computing technology will enable Neurokernel's model execution efficiency to advance significantly towards the time scale of the actual fly brain even as more realistic neuron and synapse models are employed. These advances will enable researchers to validate models of circuits in the live fly's brain within similar time scales and use the observed discrepancies to inform subsequent model improvements (Fig. 5.1).

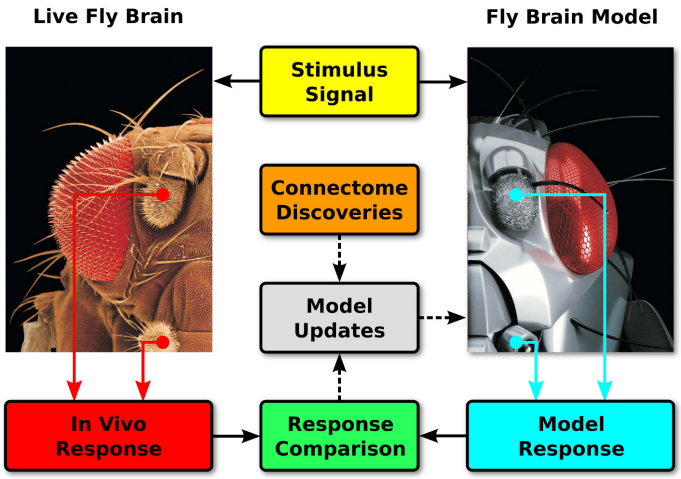


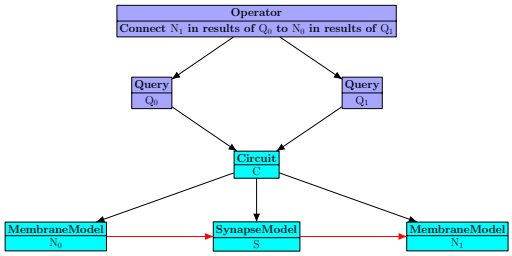
Figure 5.1: In vivo validation is essential to the development of accurate fly brain models. Neural responses to sensory stimuli are recorded from the live fly brain in real time and compared to the computed responses of the corresponding components in a fly brain model executed on the same time scale. Discrepancies between these responses and new connectome data may be used to improve the model's accuracy (fruit fly photograph adapted from Berger and fly robot image adapted from Vizcaíno, Benton, Gerber, and Louis, both reproduced with permission).

5.3 NeuroArch - Future Development

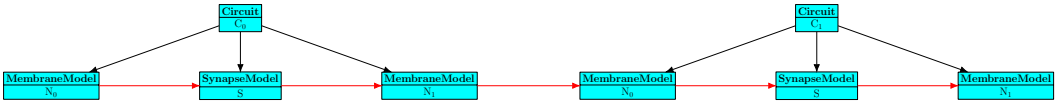
5.3.1 Model Construction Using Composition Operations

A major advantage of NeuroArch's OGM (§ 3.5.1) 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 [1]) provide incomplete data regarding the structure of neuropils, the process of inferring circuit functionality can potentially 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 must 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. 5.2). 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 will 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. 5.2a.

Figure 5.2: 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. 3.2 and 3.3, black edges denote ownership while red edges denote data transmission connections between objects.

5.3.2 Using NeuroArch Data for Neurokernel Resource Allocation

To improve the performance of model execution by Neurokernel, the structure of executable circuits stored in NeuroArch can be analyzed to estimate the computational resources required by Neurokernel to efficiently run a given circuit 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 will need to provide services for extracting relevant circuit information required to compute resource requirements. NeuroArch's data model can 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.

5.3.3 Support for Input/Output File Formats

NeuroArch's support for loading neural circuit data is currently limited to the GEXF graph storage format. Support for loading data from and saving data to additional specification formats used by other neuroinformatic tools such as SWC[1], CSV, NeuroML [48], NineML [114], or SpineML [118] would

(i) facilitate importing of existing data stored 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 (§ 5.3.4).

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 (§ 3.5.4). 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 the opportunity to define entirely new modeling elements and architectural abstractions (§ 3.3.3) 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

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

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 (§ 5.3.8), its data sharing services should also be extended to provide a way to store/load such data in a suitable file format.

5.3.4 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; (§ 5.3.3);

(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[2] or Bitbucket[3];

  (b) take advantage of the API provided by the Zenodo research data sharing service[4] to automatically request a DOI for a published model that could be made

[2] `http://github.com`

[3] `http://bitbucket.org`

[4] `http://zenodo.org/dev`

available to other researchers as the access point for obtaining model data for immediate loading into a NeuroArch instance.

5.3.5 Performance Assessment

Given that complex queries performed by NeuroArch's OGM can be computationally intensive for circuits with large numbers of components, 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.

5.3.6 Graphical Visualization of Circuit Data

NeuroArch's novel conflation of biological and circuit design data from multiple sources in a single graph database can drive new ways of graphically interacting with fly brain data. One interesting possibility is using NeuroArch as a backend to Geppetto [18], an open-source web application for exploration and visualization of biological models developed as part of the OpenWorm Project. Although Geppetto was originally designed to support simulations of *C. elegans*, its modular architecture can be extended to support technologies such as multiple GPUs required to emulate the fruit fly brain.

5.3.7 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.

5.3.8 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 should be extended to support representation of state data associated with the components of an executed circuit model at multiple times.

5.4 Generating a Modeling of the Central Complex - Future Development

The arborization data used to construct the CX model described in § 4 does not contain any information regarding the number of synapses between neurons, how the neurotransmitters expressed by different neurons should inform the design of their respective models, or what role local neurons play in processing. Analysis of central complex neuron morphologies and neurotransmitter profiles could fill some of these gaps and enable generation of more biologically plausible models of the CX neuropils.

Bibliography

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

[2] P. Arena, S. Maceo, L. Patane, and R. Strauss. A spiking network for spatial memory formation: Towards a fly-inspired ellipsoid body model. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, August 2013. `http://dx.doi.org/10.1109/IJCNN.2013.6706882`.

[3] J. Douglas Armstrong and Jano I. van Hemert. Towards a virtual fly brain. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1896):2387 –2397, June 2009. `http://dx.doi.org/10.1098/rsta.2008.0308`.

[4] 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`.

[5] Helen Attrill, Kathleen Falls, Joshua L. Goodman, Gillian H. Millburn, Giulia Antonazzo, Alix J. Rey, Steven J. Marygold, and the FlyBase Consortium. FlyBase: establishing a Gene Group resource for Drosophila melanogaster. *Nucleic Acids Research*, 44(D1):D786–D792, January 2016. `http://dx.doi.org/10.1093/nar/gkv1046`.

[6] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7, January 2014. `http://dx.doi.org/10.3389/fninf.2013.00048`.

[7] John A. Bender, Alan J. Pollack, and Roy E. Ritzmann. Neural Activity in the Central Complex of the Insect Brain Is Linked to Locomotor Changes. *Current Biology*, 20(10):921–926, May 2010. `http://dx.doi.org/10.1016/j.cub.2010.03.054`.

[8] B. V. Benjamin, Peiran Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716, May 2014. `10.1109/JPROC.2014.2313565`.

[9] Ulysses Bernardet and Paul F. M. J. Verschure. iqr: A tool for the construction of multi-level simulations of brain and behaviour. *Neuroinformatics*, 8(2):113–134, June 2010. http://dx.doi.org/10.1007/s12021-010-9069-7.

[10] M. Beyeler, K.D. Carlson, Ting-Shuo Chou, N. Dutt, and J.L. Krichmar. CARLsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. http://dx.doi.org/10.1109/IJCNN.2015.7280424.

[11] James Bower and David Beeman. *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulations System.* Springer, Santa Clara, Calif, December 1994.

[12] S. Bradner. The Internet Standards Process - Revision 3. *Internet RFCs, ISSN 2070-1721*, RFC 2026, October 1996. http://www.rfc-editor.org/rfc/rfc2026.txt.

[13] Romain Brette and Dan F. M. Goodman. Simulating spiking neural networks on GPU, December 2012. http://informahealthcare.com/doi/abs/10.3109/0954898X.2012.730170.

[14] Daniel Brüderle, Mihai A. Petrovici, Bernhard Vogginger, Matthias Ehrlich, Thomas Pfeil, Sebastian Millner, Andreas Grübl, Karsten Wendt, Eric Müller, Marc-Olivier Schwartz, Dan Husmann Oliveira, Sebastian Jeltsch, Johannes Fieres, Moritz Schilling, Paul Müller, Oliver Breitwieser, Venelin Petkov, Lyle Muller, Andrew P. Davison, Pradeep Krishnamurthy, Jens Kremkow, Mikael Lundqvist, Eilif Muller, Johannes Partzsch, Stefan Scholze, Lukas Zühl, Christian Mayr, Alain Destexhe, Markus Diesmann, Tobias C. Potjans, Anders Lansner, René Schüffny, Johannes Schemmel, and Karlheinz Meier. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological Cybernetics*, 104(4-5):263–296, May 2011. http://www.springerlink.com/content/xlh783325w537622/.

[15] Seth A. Budick and Michael H. Dickinson. Free-flight responses of Drosophila melanogaster to attractive odors. *Journal of Experimental Biology*, 209(15):3001 – 3017, 2006. http://dx.doi.org/10.1242/jeb.02305.

[16] Randal Burns, William Gray Roncal, Dean Kleissas, Kunal Lillaney, Priya Manavalan, Eric Perlman, Daniel R. Berger, Davi D. Bock, Kwanghun Chung, Logan Grosenick, Narayanan Kasthuri, Nicholas C. Weiler, Karl Deisseroth, Michael Kazhdan, Jeff Lichtman, R. Clay Reid, Stephen J. Smith, Alexander S. Szalay, Joshua T. Vogelstein, and R. Jacob Vogelstein. The Open Connectome Project Data Cluster: Scalable Analysis and Vision for High-Throughput Neuroscience. *Scientific and statistical database management: International Conference, SSDBM ...: proceedings. International Conference on Scientific and Statistical Database Management*, 2013. http://dx.doi.org/10.1145/2484838.2484870.

[17] Sebastian Busch, Mareike Selcho, Kei Ito, and Hiromu Tanimoto. A map of octopaminergic neurons in the Drosophila brain. *The Journal of Comparative Neurology*, 513(6):643–667, April 2009. http://dx.doi.org/10.1002/cne.21966.

[18] Matteo Cantarelli, Giovanni Idili, Adrian Quintana Perez, Boris Marin, and Jesus Martinez. Geppetto simulation platform for complex biological systems [Internet], 2016. http://geppetto.org.

[19] Nicholas T. Carnevale and Michael L. Hines. *The NEURON Book*. Cambridge University Press, Cambridge; New York, 2006.

[20] Chun-Chao Chen, Jie-Kai Wu, Hsuan-Wen Lin, Tsung-Pin Pai, Tsai-Feng Fu, Chia-Lin Wu, Tim Tully, and Ann-Shyn Chiang. Visualizing Long-Term Memory Formation in Two Neurons of the Drosophila Brain. *Science*, 335(6069):678–685, February 2012. http://dx.doi.org/10.1126/science.1212735.

[21] Kit Cheung, Simon R. Schultz, and Wayne Luk. NeuroFlow: A General Purpose Spiking Neural Network Simulation Platform using Customizable Processors. *Neuromorphic Engineering*, page 516, 2016. http://dx.doi.org/10.3389/fnins.2015.00516.

[22] Ann-Shyn Chiang, Chih-Yung Lin, Chao-Chun Chuang, Hsiu-Ming Chang, Chang-Huain Hsieh, Chang-Wei Yeh, Chi-Tin Shih, Jian-Jheng Wu, Guo-Tzau Wang, and Yung-Chang Chen. Three-dimensional reconstruction of brain-wide wiring networks in Drosophila at single-cell resolution. *Current Biology*, 21(1):1–11, January 2011. http://dx.doi.org/10.1016/j.cub.2010.11.056.

[23] M. Eugenia Chiappe, Johannes D. Seelig, Michael B. Reiser, and Vivek Jayaraman. Walking modulates speed sensitivity in Drosophila motion vision. *Current Biology*, 20(16):1470–1475, August 2010. http://dx.doi.org/10.1016/j.cub.2010.06.072.

[24] Dmitri B. Chklovskii, Shiv Vitaladevuni, and Louis K. Scheffer. Semi-automated reconstruction of neural circuits using electron microscopy. *Current Opinion in Neurobiology*, 20(5):667–675, October 2010. http://dx.doi.org/16/j.conb.2010.08.002.

[25] Alex Cope, Chelsea Sabo, Esin Yavuz, Kevin Gurney, James Marshall, Thomas Nowotny, and Eleni Vasilaki. The Green Brain Project - Developing a Neuromimetic Robotic Honeybee. In Nathan F. Lepora, Anna Mura, Holger G. Krapp, Paul F. M. J. Verschure, and Tony J. Prescott, editors, *Biomimetic and Biohybrid Systems*, number 8064 in Lecture Notes in Computer Science, pages 362–363. Springer Berlin Heidelberg, July 2013. http://dx.doi.org/10.1007/978-3-642-39802-5_35.

[26] Marta Costa, James D. Manton, Aaron D. Ostrovsky, Steffen Prohaska, and Gregory S.X.E. Jefferis. NBLAST: Rapid, sensitive comparison of neuronal structure and construction of neuron family databases. *bioRxiv*, page 006346, February 2016. http://dx.doi.org/10.1101/006346.

[27] 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.

[28] Lisandro D. Dalcin, Pablo A. Kler, Rodrigo R. Paz, and Alejandro Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, 2011. http://dx.doi.org/10.1016/j.advwatres.2011.04.013.

[29] Richard W. Daniels, Maria V. Gelfand, Catherine A. Collins, and Aaron DiAntonio. Visualizing glutamatergic cell bodies and synapses in Drosophila larval and adult CNS. *The Journal of Comparative Neurology*, 508(1):131–152, May 2008. http://dx.doi.org/10.1002/cne.21670.

[30] Andrew P. Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2:11, 2009. http://dx.doi.org/10.3389/neuro.11.011.2008.

[31] Alex D.M. Dewar, Antoine Wystrach, Paul Graham, and Andrew Philippides. Navigation-specific neural coding in the visual system of Drosophila. *Bio Systems*, 136:120–127, October 2015. http://dx.doi.org/10.1016/j.biosystems.2015.07.008.

[32] Mikael Djurfeldt. The Connection-Set Algebra - A Novel Formalism for the Representation of Connectivity Structure in Neuronal Network Models. *Neuroinformatics*, 10(3):287–304, July 2012. http://dx.doi.org/10.1007/s12021-012-9146-1.

[33] Mikael Djurfeldt, Johannes Hjorth, Jochen M. Eppler, Niraj Dudani, Moritz Helias, Tobias C. Potjans, Upinder S. Bhalla, Markus Diesmann, Jeanette Hellgren Kotaleski, and Örjan Ekeberg. Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics*, 8(1):43–60, March 2010. http://dx.doi.org/10.1007/s12021-010-9064-z.

[34] 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://dx.doi.org/10.1093/nar/gku1099.

[35] Joseph B. Duffy. GAL4 system in Drosophila: a fly geneticist's Swiss Army knife. *Genesis (New York, N.Y.: 2000)*, 34(1-2):1–15, October 2002. http://dx.doi.org/10.1002/gene.10150.

[36] Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. The MIT Press, new edition edition, August 2004.

[37] Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, November 2012. http://dx.doi.org/10.1126/science.1225266.

[38] A.K. Fidjeland, E.B. Roesch, M.P. Shanahan, and W. Luk. NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs. In *20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009. ASAP 2009*, pages 137–144, July 2009. http://dx.doi.org/10.1109/ASAP.2009.24.

[39] A.K. Fidjeland and M.P. Shanahan. Accelerated simulation of spiking neural networks using GPUs. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8, 2010. http://dx.doi.org/10.1109/IJCNN.2010.5596678.

[40] Bryan Ford. Parsing expression grammars: a recognition-based syntactic foundation. In *ACM SIGPLAN Notices*, volume 39, pages 111–122. ACM, 2004. http://dx.doi.org/10.1145/982962.964011.

[41] Mark A. Frye and Michael H. Dickinson. Closing the loop between neurobiology and flight behavior in Drosophila. *Current Opinion in Neurobiology*, 14(6):729–736, December 2004. http://dx.doi.org/10.1016/j.conb.2004.10.004.

[42] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004. http://dx.doi.org/10.1007/978-3-540-30218-6_19.

[43] Marc-Oliver Gewaltig and Markus Diesmann. NEST (NEural {simulation} {tool}). *Scholarpedia*, 2(4):1430, 2007. http://dx.doi.org/10.4249/scholarpedia.1430.

[44] Lev E. Givon and Aurel A. Lazar. Neurokernel: An open source platform for emulating the fruit fly brain. *PLoS ONE*, January 2016. http://dx.doi.org/10.1371/journal.pone.0146581.s001. Also available as Neurokernel RFC #4: http://dx.doi.org/10.5281/zenodo.31947.

[45] Lev E. Givon, Aurel A. Lazar, and Nikul H. Ukani. Neuroarch: A Graph-Based Platform for Constructing and Querying Models of the Fruit Fly Brain Architecture. *Frontiers in Neuroinformatics*, (42), Aug 2014.

[46] Lev E. Givon, Aurel A. Lazar, and Nikul H. Ukani. NeuroArch: A Graph dB for Querying and Executing Fruit Fly Brain Circuits. December 2015. http://dx.doi.org/10.5281/zenodo.44225.

[47] Lev E. Givon, Thomas Unterthiner, N. Benjamin Erichson, David Wei Chiang, Eric Larson, Luke Pfister, Sander Dieleman, Gregory R. Lee, Stefan van der Walt, Teodor Mihai Moldovan, Frédéric Bastien, Xing Shi, Jan Schlüter, Brian Thomas, Chris Capdevila, Alex Rubinsteyn, Michael M. Forbes, Jacob Frelinger, Tim Klein, Bruce Merry, Lars Pastewka, Steve Taylor, Feng Wang, and Yiyin Zhou. scikit-cuda 0.5.1: a Python interface to GPU-powered libraries [Internet], December 2015. http://dx.doi.org/10.5281/zenodo.40565.

[48] 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 Comput Biol*, 6(6):e1000815, June 2010. http://dx.doi.org/10.1371/journal.pcbi.1000815.

[49] Padraig Gleeson, Sharon Crook, Angus Silver, and Robert Cannon. Development of NeuroML version 2.0: greater extensibility, support for abstract neuronal models and interaction with systems biology languages. *BMC Neuroscience*, 12(Suppl 1):P29, July 2011. http://dx.doi.org/10.1186/1471-2202-12-S1-P29.

[50] 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://dx.doi.org/10.1186/1471-2202-13-S1-O7.

[51] 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.

[52] Dan F.M. Goodman and Romain Brette. The Brian simulator. *Frontiers in Neuroscience*, September 2009. http://dx.doi.org/10.3389/neuro.01.026.2009.

[53] 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.

[54] U. Hanesch, K. F. Fischbach, and M. Heisenberg. Neuronal architecture of the central complex in Drosophila melanogaster. *Cell and Tissue Research*, 257(2):343–366, 1989. http://dx.doi.org/10.1007/BF00261838.

[55] Stanley Heinze and Uwe Homberg. Neuroarchitecture of the central complex of the desert locust: Intrinsic and columnar neurons. *The Journal of Comparative Neurology*, 511(4):454–478, December 2008. http://dx.doi.org/10.1002/cne.21842.

[56] Moritz Helias, Susanne Kunkel, Gen Masumoto, Jun Igarashi, Jochen Martin Eppler, Shin Ishii, Tomoki Fukai, Abigail Morrison, and Markus Diesmann. Supercomputers ready for use as discovery machines for neuroscience. *Frontiers in Neuroinformatics*, 6:26, 2012. http://dx.doi.org/10.3389/fninf.2012.00026.

[57] Michael Hines, Sameer Kumar, and Felix Schürmann. Comparison of neuronal spike exchange methods on a Blue Gene/P supercomputer. *Frontiers in Computational Neuroscience*, 5, November 2011. http://dx.doi.org/10.3389/fncom.2011.00049.

[58] 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. PMID: 15218350.

[59] Roger V. Hoang, Devyani Tanna, Laurence C. Jayet Bray, Sergiu M. Dascalu, and Frederick C. Harris Jr. A novel CPU/GPU simulation environment for large-scale biologically realistic neural modeling. *Frontiers in Neuroinformatics*, 7:19, 2013. http://dx.doi.org/10.3389/fninf.2013.00019.

[60] Yu-Chi Huang, Cheng-Te Wang, Guo-Tzau Wang, Ta-Shun Su, Pao-Yueh Hsiao, Ching-Yao Lin, Chang-Huain Hsieh, Hsiu-Ming Chang, and Chung-Chuan Lo. The Flysim project - persistent simulation and real-time visualization of fruit fly whole-brain spiking neural network model. In *Frontiers in Neuroinformatics*, Leiden, Netherlands, August 2014. http://dx.doi.org/10.3389/conf.fninf.2014.18.00043.

[61] Stephen J. Huston and Vivek Jayaraman. Studying sensorimotor integration in insects. *Current Opinion in Neurobiology*, 21(4):527–534, August 2011. http://dx.doi.org/10.1016/j.conb.2011.05.030.

[62] 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://dx.doi.org/10.1016/j.neuron.2013.12.017.

[63] E.M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, 2003. http://dx.doi.org/10.1109/TNN.2003.820440.

[64] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: open source scientific tools for Python, 2001. http://www.scipy.org.

[65] Lily Kahsai and Åsa M.E. Winther. Chemical neuroanatomy of the Drosophila central complex: Distribution of multiple neuropeptides in relation to neurotransmitters. *The Journal of Comparative Neurology*, 519(2):290–315, 2011. http://dx.doi.org/10.1002/cne.22520.

[66] Eric R. Kandel, Henry Markram, Paul M. Matthews, Rafael Yuste, and Christof Koch. Neuroscience thinks big (and collaboratively). *Nature Reviews Neuroscience*, 14(9):659–664, September 2013. http://dx.doi.org/10.1038/nrn3578.

[67] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. http://dx.doi.org/10.1137/S1064827595287997.

[68] Anmo J. Kim, Aurel A. Lazar, and Yevgeniy B. Slutskiy. System identification of Drosophila olfactory sensory neurons. *Journal of Computational Neuroscience*, 30(1):143–161, August 2011. http://dx.doi.org/10.1007/s10827-010-0265-0.

[69] Anmo J. Kim, Aurel A. Lazar, and Yevgeniy B. Slutskiy. Projection neurons in Drosophila antennal lobes signal the acceleration of odor concentrations. *eLife*, page e06651, May 2015. http://dx.doi.org/10.7554/eLife.06651.

[70] Kuno Kirschfeld. Die projektion der optischen umwelt auf das raster der rhabdomere im komplex auge von musca. *Experimental Brain Research*, 3:248–270, 1967.

[71] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, and Ahmed Fasih. PyCUDA and PyOpenCL: a scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174, March 2012. http://dx.doi.org/10.1016/j.parco.2011.09.001.

[72] Aurel A. Lazar. Programming telecommunication networks. *IEEE Network*, 11(5):8–18, October 1997. http://dx.doi.org/10.1109/65.620517.

[73] Aurel A. Lazar, Konstantinos Psychas, Nikul H. Ukani, and Yiyin Zhou. A Parallel Processing Model of the Drosophila Retina, August 2015. NK RFC #3, http://dx.doi.org/10.5281/zenodo.30036.

[74] Aurel A. Lazar, Konstantinos Psychas, Nikul H. Ukani, and Yiyin Zhou. Retina of the fruit fly eyes: a detailed simulation model. *BMC Neuroscience 2015*, 16(Suppl 1):301, 2015. http://dx.doi.org/10.1186/1471-2202-16-S1-P301.

[75] Aurel A. Lazar and Yevgeniy B. Slutskiy. Channel Identification Machines for Multidimensional Receptive Fields. *Frontiers in Computational Neuroscience*, 8, 2014. http://dx.doi.org/10.3389/fncom.2014.00117.

[76] Aurel A. Lazar and Yevgeniy B. Slutskiy. Functional Identification of Spike-Processing Neural Circuits. *Neural Computation*, 26(2), February 2014. http://dx.doi.org/10.1162/NECO_a_00543.

[77] Aurel A. Lazar and Yevgeniy B. Slutskiy. Spiking Neural Circuits with Dendritic Stimulus Processors. *Journal of Computational Neuroscience*, 38(1):1–24, 2015. http://dx.doi.org/10.1007/s10827-014-0522-8.

[78] Aurel A. Lazar, Yevgeniy B. Slutskiy, and Yiyin Zhou. Massively Parallel Neural Circuits for Stereoscopic Color Vision: Encoding, Decoding and Identification. *Neural Networks*, 63:254–271, 2015. http://dx.doi.org/10.1016/j.neunet.2014.10.014.

[79] 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.

[80] Aurel A. Lazar, Nikul H. Ukani, and Yiyin Zhou. The Cartridge: A Canonical Neural Circuit Abstraction of the Lamina Neuropil - Construction and Composition Rules, January 2014. NK RFC #2, http://dx.doi.org/10.5281/zenodo.11856.

[81] Aurel A. Lazar and Yiyin Zhou. Volterra Dendritic Stimulus Processors and Biophysical Spike Generators with Intrinsic Noise Sources. *Frontiers in Computational Neuroscience*, 8, 2014. http://dx.doi.org/10.3389/fncom.2014.00095.

[82] Chih-Yung Lin, Chao-Chun Chuang, Tzu-En Hua, Chun-Chao Chen, Barry J. Dickson, Ralph J. Greenspan, and Ann-Shyn Chiang. A comprehensive wiring diagram of the protocerebral bridge for visual information processing in the Drosophila brain. *Cell Reports*, 3(5):1739–1753, May 2013. http://dx.doi.org/10.1016/j.celrep.2013.04.022.

[83] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, December 1997. http://dx.doi.org/10.1016/S0893-6080(97)00011-7.

[84] Gaby Maimon, Andrew D. Straw, and Michael H. Dickinson. A simple vision-based algorithm for decision making in flying Drosophila. *Current Biology*, 18(6):464–470, March 2008. http://dx.doi.org/10.1016/j.cub.2008.02.054.

[85] Matthew S. Maisak, Juergen Haag, Georg Ammer, Etienne Serbe, Matthias Meier, Aljoscha Leonhardt, Tabea Schilling, Armin Bahl, Gerald M. Rubin, Aljoscha Nern, Barry J. Dickson, Dierk F. Reiff, Elisabeth Hopp, and Alexander Borst. A directional tuning map of Drosophila elementary motion detectors. *Nature*, 500(7461):212–216, August 2013. http://dx.doi.org/10.1038/nature12320.

[86] Zhengmei Mao and Ronald L. Davis. Eight Different Types of Dopaminergic Neurons Innervate the Drosophila Mushroom Body Neuropil: Anatomical and Physiological Heterogeneity. *Frontiers in Neural Circuits*, 3, July 2009. http://dx.doi.org/10.3389/neuro.04.005.2009.

[87] L. Marenco, P. Nadkarni, E. Skoufos, G. Shepherd, and P. Miller. Neuronal database integration: the Senselab EAV data model. *Proceedings of the AMIA Symposium*, pages 102–106, 1999. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2232788/.

[88] Alfonso Martín-Peña, Angel Acebes, José-Rodrigo Rodríguez, Valerie Chevalier, Sergio Casas-Tinto, Tilman Triphan, Roland Strauss, and Alberto Ferrús. Cell types and coincident synapses in the ellipsoid body of Drosophila. *European Journal of Neuroscience*, 39(10):1586–1601, May 2014. http://dx.doi.org/10.1111/ejn.12537.

[89] 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.

[90] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, August 2014. http://science.sciencemag.org/content/345/6197/668.

[91] 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.

[92] K. Minkovich, C.M. Thibeault, M.J. O'Brien, A. Nogin, Y. Cho, and N. Srinivasa. HRLSim: a high performance spiking neural network simulator for GPGPU clusters. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):316–331, 2014. http://dx.doi.org/10.1109/TNNLS.2013.2276056.

[93] Javier Morante and Claude Desplan. The color-vision circuit in the medulla of Drosophila. *Current Biology*, 18(8):553–565, April 2008. http://dx.doi.org/10.1016/j.cub.2008.02.075.

[94] Laiyong Mu, Kei Ito, Jonathan P. Bacon, and Nicholas J. Strausfeld. Optic glomeruli and their inputs in Drosophila share an organizational ground pattern with the antennal lobes. *The Journal of Neuroscience*, 32(18):6061–6071, May 2012. http://dx.doi.org/10.1523/JNEUROSCI.0221-12.2012.

[95] U. Müller. The nitric oxide system in insects. *Progress in Neurobiology*, 51(3):363–381, February 1997.

[96] Jim Mutch, Ulf Knoblich, and Tomaso Poggio. CNS: a GPU-based framework for simulating cortically-organized networks. Technical Report MIT-CSAIL-TR-2010-013, MIT, 2010. http://gpucomputing.net/?q=node/429.

[97] Jayram Moorkanikara Nageswaran, Nikil Dutt, Jeffrey L. Krichmar, Alex Nicolau, and Alexander V. Veidenbaum. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5-6):791–800, July 2009. http://dx.doi.org/10.1016/j.neunet.2009.06.028.

[98] D. R. Nässel. Histamine in the brain of insects: a review. *Microscopy Research and Technique*, 44(2-3):121–136, February 1999. http://dx.doi.org/10.1002/(SICI)1097-0029(19990115/01)44:2/3%3C121::AID-JEMT6%3E3.0.CO;2-F.

[99] Eilen Nordlie, Marc-Oliver Gewaltig, and Hans Ekkehard Plesser. Towards Reproducible Descriptions of Neuronal Network Models. *PLoS Comput Biol*, 5(8):e1000456, 2009. http://dx.doi.org/10.1371/journal.pcbi.1000456.

[100] NVIDIA. CUDA Toolkit 4.0 Readiness for CUDA Applications, March 2011.

[101] NVIDIA. Kepler GK110 whitepaper, 2012.

[102] Hideo Otsuna and Kei Ito. Systematic analysis of the visual projection neurons of Drosophila melanogaster. I. Lobula-specific pathways. *The Journal of Comparative Neurology*, 497(6):928–958, August 2006.

[103] Hideo Otsuna, Kazunori Shinomiya, and Kei Ito. Parallel neural pathways in higher visual centers of the Drosophila brain that mediate wavelength-specific behavior. *Frontiers in Neural Circuits*, 8, February 2014. http://dx.doi.org/10.3389/fncir.2014.00008.

[104] Andrey Palyanov, Sergey Khayrulin, and Mike Vella. Sibernetic fluid mechanics simulator [Internet], 2015. http://openworm.github.io/sibernetic/.

[105] 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.

[106] Dejan Pecevski, Thomas Natschläger, and Klaus Schuch. PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. *Frontiers in Neuroinformatics*, 3:11, 2009. http://dx.doi.org/10.3389/neuro.11.011.2009.

[107] Hanchuan Peng, Alessandro Bria, Zhi Zhou, Giulio Iannello, and Fuhui Long. Extensible visualization and analysis for multidimensional images using Vaa3d. *Nature Protocols*, 9(1):193–208, January 2014. http://dx.doi.org/10.1038/nprot.2014.011.

[108] Hanchuan Peng, Zongcai Ruan, Fuhui Long, Julie H. Simpson, and Eugene W. Myers. V3D enables real-time 3d visualization and quantitative analysis of large-scale biological image data sets. *Nature Biotechnology*, 28(4):348–353, April 2010. http://dx.doi.org/10.1038/nbt.1612.

[109] Hanchuan Peng, Jianyong Tang, Hang Xiao, Alessandro Bria, Jianlong Zhou, Victoria Butler, Zhi Zhou, Paloma T. Gonzalez-Bellido, Seung W. Oh, Jichao Chen, Ananya Mitra, Richard W. Tsien, Hongkui Zeng, Giorgio A. Ascoli, Giulio Iannello, Michael Hawrylycz, Eugene Myers, and Fuhui Long. Virtual finger boosts three-dimensional imaging and microsurgery as well as terabyte volume image visualization and analysis. *Nature Communications*, 5:4342, 2014. http://dx.doi.org/10.1038/ncomms5342.

[110] Wayne Pereanu, Abilasha Kumar, Arnim Jennett, Heinrich Reichert, and Volker Hartenstein. http://dx.doi.org/10.1002/cne.22376.

[111] F. Perez and B.E. Granger. IPython: a system for interactive scientific computing. *Computing in Science Engineering*, 9(3):21 –29, June 2007. http://dx.doi.org/10.1109/MCSE.2007.53.

[112] Keram Pfeiffer and Uwe Homberg. Organization and Functional Roles of the Central Complex in the Insect Brain. *Annual Review of Entomology*, 59(1):165–184, 2014. http://dx.doi.org/10.1146/annurev-ento-011613-162031.

[113] James Phillips-Portillo. The Central Complex of the Flesh Fly, Neobellieria bullata: Recordings and Morphologies of Protocerebral Inputs and Small-Field Neurons. *The Journal of Comparative Neurology*, 520(14):3088–3104, October 2012. http://dx.doi.org/10.1002/cne.23134.

[114] 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.

[115] Alexander D. Rast, Xin Jin, Francesco Galluppi, Luis A. Plana, Cameron Patterson, and Steve Furber. Scalable event-driven native parallel processing: the SpiNNaker neuromimetic system. In *Proceedings of the 7th ACM international conference on Computing frontiers*, CF '10, page 21–30, New York, NY, USA, 2010. ACM. ACM ID: 1787279.

[116] Subhasis Ray and Upinder S. Bhalla. PyMOOSE: Interoperable Scripting in Python for MOOSE. *Frontiers in Neuroinformatics*, 2, December 2008. http://dx.doi.org/10.3389/neuro.11.006.2008.

[117] Micah Richert, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L. Krichmar. An efficient simulation environment for modeling large-scale cortical processing. *Frontiers in Neuroinformatics*, 5:19, 2011. http://dx.doi.org/10.3389/fninf.2011.00019.

[118] 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.

[119] Jens Rister, Dennis Pauls, Bettina Schnell, Chun-Yuan Ting, Chi-Hon Lee, Irina Sinakevitch, Javier Morante, Nicholas J. Strausfeld, Kei Ito, and Martin Heisenberg. Dissection of the peripheral motion channel in the visual system of Drosophila melanogaster. *Neuron*, 56(1):155–170, October 2007. http://dx.doi.org/10.1016/j.neuron.2007.09.014.

[120] Pedro Rittner and Thomas A. Cleland. Myriad: a transparently parallel GPU - based simulator for densely integrated biophysical models. In *Society of Neuroscience Abstracts*, page 187.02, 2014.

[121] Pedro Rittner and Thomas A. Cleland. The Myriad simulator : densely coupled realistic neural networks on GPU. page P4169, 2014.

[122] Pedro Rittner, Andrew J. Davies, and Thomas A. Cleland. The Myriad simulator: parallel computation for densely integrated models. In *Society of Neuroscience Abstracts*, page 451.12, 2015.

[123] 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`.

[124] Joshua R. Sanes and S. Lawrence Zipursky. Design principles of insect and vertebrate visual systems. *Neuron*, 66(1):15–36, April 2010. `http://dx.doi.org/10.1016/j.neuron.2010.01.018`.

[125] Johannes D. Seelig and Vivek Jayaraman. Feature detection and orientation tuning in the Drosophila central complex. *Nature*, advance online publication, October 2013. `http://dx.doi.org/10.1038/nature12601`.

[126] Johannes D. Seelig and Vivek Jayaraman. Neural dynamics for landmark orientation and angular path integration. *Nature*, 521(7551):186–191, May 2015. `http://dx.doi.org/10.1038/nature14446`.

[127] Chi-Tin Shih, Olaf Sporns, Shou-Li Yuan, Ta-Shun Su, Yen-Jen Lin, Chao-Chun Chuang, Ting-Yuan Wang, Chung-Chuang Lo, Ralph J. Greenspan, and Ann-Shyn Chiang. Connectomics-Based Analysis of Information Flow in the Drosophila Brain. *Current Biology*, 25(10):1249–1258, May 2015. `http://dx.doi.org/10.1016/j.cub.2015.03.021`.

[128] Tripathy Shreejoy, Richard Gerkin, Judy Savitskaya, and Nathaniel Urban. NeuroElectro.org: a community database on the electrophysiological diversity of mammalian neuron types. *Frontiers in Neuroinformatics*, 7, 2013.

[129] Zhuoyi Song, Marten Postma, Stephen A. Billings, Daniel Coca, Roger C. Hardie, and Mikko Juusola. Stochastic, adaptive sampling of information by microvilli in fly photoreceptors. *Current Biology*, 22(15):1371–1380, June 2012. `http://dx.doi.org/10.1016/j.cub.2012.05.047`.

[130] Marcel Stimberg, Dan F. M. Goodman, Victor Benichoux, and Romain Brette. Equation-oriented specification of neural models for simulations. *Frontiers in Neuroinformatics*, 8:6, 2014. `http://dx.doi.org/10.3389/fninf.2014.00006`.

[131] Nicholas J. Strausfeld and Frank Hirth. Deep Homology of Arthropod Central Complex and Vertebrate Basal Ganglia. *Science*, 340(6129):157–161, April 2013. http://dx.doi.org/10.1126/science.1231828.

[132] R. Strauss. Neurobiological Models of the Central Complex and the Mushroom Bodies. In Paolo Arena and Luca Patanè, editors, *Spatial Temporal Patterns for Action-Oriented Perception in Roving Robots II*, number 21 in Cognitive Systems Monographs, pages 3–41. Springer International Publishing, January 2014. http://link.springer.com/chapter/10.1007/978-3-319-02362-5_1.

[133] Balazs Szigeti, Padraig Gleeson, Michael Vella, Sergey Khayrulin, Andrey Palyanov, Jim Hokanson, Michael Currie, Matteo Cantarelli, Giovanni Idili, and Stephen Larson. OpenWorm: an open-science approach to modelling Caenorhabditis elegans. *Frontiers in Computational Neuroscience*, 8:137, 2014. http://dx.doi.org/10.3389/fncom.2014.00137.

[134] Shin-Ya Takemura, Arjun Bharioke, Zhiyuan Lu, Aljoscha Nern, Shiv Vitaladevuni, Patricia K. Rivlin, William T. Katz, Donald J. Olbris, Stephen M. Plaza, Philip Winston, Ting Zhao, Jane Anne Horne, Richard D. Fetter, Satoko Takemura, Katerina Blazek, Lei-Ann Chang, Omotara Ogundeyi, Mathew A. Saunders, Victor Shapiro, Christopher Sigmund, Gerald M. Rubin, Louis K. Scheffer, Ian A. Meinertzhagen, and Dmitri B. Chklovskii. A visual motion detection circuit suggested by Drosophila connectomics. *Nature*, 500(7461):175–181, August 2013. http://dx.doi.org/10.1038/nature12450.

[135] A. M. Vallés and K. White. Serotonin-containing neurons in Drosophila melanogaster: development and distribution. *The Journal of Comparative Neurology*, 268(3):414–428, February 1988. http://dx.doi.org/10.1002/cne.902680310.

[136] S. van der Walt, S.C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2):22–30, March 2011. http://dx.doi.org/10.1109/MCSE.2011.37.

[137] Trevor J. Wardill, Olivier List, Xiaofeng Li, Sidhartha Dongre, Marie McCulloch, Chun-Yuan Ting, Cahir J. O'Kane, Shiming Tang, Chi-Hon Lee, Roger C. Hardie, and Mikko Juusola. Multiple spectral inputs improve motion discrimination in the Drosophila visual system. *Science*, 336(6083):925–931, May 2012. http://dx.doi.org/10.1126/science.1215317.

[138] Peter T. Weir, Bettina Schnell, and Michael H Dickinson. Central complex neurons exhibit behaviorally gated responses to visual motion in Drosophila. *Journal of Neurophysiology*, 111(1):62–71, January 2014. http://dx.doi.org/10.1152/jn.00593.2013.

[139] J.G. White, E. Southgate, J.N. Thomson, and S. Brenner. The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the*

Royal Society of London. B, Biological Sciences, 314(1165):1–340, November 1986. PMID: 22462104.

[140] Rachel I. Wilson. Understanding the functional consequences of synaptic specialization: insight from the Drosophila antennal lobe. *Current Opinion in Neurobiology*, 21(2):254–260, April 2011. http://dx.doi.org/10.1016/j.conb.2011.03.002.

[141] Tanya Wolff, Nirmala A. Iyer, and Gerald M. Rubin. Neuroarchitecture and neuroanatomy of the Drosophila central complex: A GAL4-based dissection of protocerebral bridge neurons and circuits. *The Journal of Comparative Neurology*, 523(7):997–1037, May 2015. http://dx.doi.org/10.1002/cne.23705.

[142] Esin Yavuz, Pascale Maul, and Thomas Nowotny. Spiking neural network model of reinforcement learning in the honeybee implemented on the GPU. In *BMC Neuroscience 2015*, volume 16, supp. 1, page 181, Prague, 2015.

[143] Esin Yavuz and Thomas Nowotny. A modelling framework for the olfactory system of the honeybee using GeNN (GPU enhanced Neuronal Network simulation environment). *Flavour*, 3(S1):1–1, April 2014. http://dx.doi.org/10.1186/2044-7248-3-S1-P23.

[144] Esin Yavuz, James Turner, and Thomas Nowotny. GeNN: a code generation framework for accelerated brain simulations. *Scientific Reports*, 6:18854, January 2016. http://www.nature.com/articles/srep18854.

[145] J.M. Young and J.D. Armstrong. Building the central complex in Drosophila: The generation and development of distinct neural subsets. *The Journal of Comparative Neurology*, 518(9):1525–1541, 2010. http://dx.doi.org/10.1002/cne.22285.

[146] J.M. Young and J.D. Armstrong. Structure of the adult central complex in Drosophila: organization of distinct neuronal subsets. *The Journal of Comparative Neurology*, 518(9):1500–1524, 2010. http://dx.doi.org/10.1002/cne.22284.

Appendix

Neurotransmitters	Neuropil	Cell Type	References
Acetylcholine	PB, FB, EB, NO	?	[65]
Glutamate	PB, FB, EB, NO	Col, Tan	[29, 65]
GABA	FB, EB	Tan	[54]
Dopamine	PB, FB, EB, NO	Tan	[86]
Histamine	?	?	[98]
Octopamine	PB, FB	Asc, Tan	[17]
Serotonin	FB, EB, NO	Tan	[135]
Nitric oxide	FB, EB	?	[95]

Table 1: Neurotransmitters in the fruit fly CX (adapted from [112]). Columnar neurons include those that connect PB to other neuropils or connect FB and EB, NO, LAL, or other neuropils. Tangential neurons include PB local neurons, F neurons in FB, and BU-EB neurons in EB. Ascending neurons connect the subesophageal ganglion to FB.

Neuron Family	Neurotransmitter
AL-PB	?
BU-EB	Acetylcholine [88, p.1598], Glutamate, GABA
BU-LAL	?
DAL	?
EB-FB	?
EB-FB-LAL-SMP	Dopamine
EB-LAL-PB	Acetylcholine
EB-NO	?
F	Dopamine
FB local	Acetylcholine
FB-BU-LAL	?
FB-EB	?
FB-NO	?
IB-LAL-PS-PB	Dopamine
PB local	Acetylcholine, Glutamate
PB-EB	?
PB-EB-BU	?
PB-EB-LAL	Acetylcholine
PB-EB-NO	Acetylcholine
PB-FB	?
PB-FB-CRE	?
PB-FB-LAL	Acetylcholine
PB-FB-NO	Acetylcholine
PS-IB-PB	?
PS-PB	?
WED-PS-PB	Acetylcholine

Table 2: Neurotransmitter profiles of specific neural pathways in the fruit fly CX (adapted from [82, Fig. 7c] and [88]).

	Label
1	PB/R8|R9/b-PB/R[4-8]/s
2	PB/L8|L9/b-PB/L[4-8]/s
3	PB/R7|L2/b-PB/R[1-9]|L[1-9]/s
4	PB/L7|R2/b-PB/L[1-9]|R[1-9]/s
5	PB/R6|L3/b-PB/R[1-9]|L[1-9]/s
6	PB/L6|R3/b-PB/L[1-9]|R[1-9]/s
7	PB/R5|L4/b-PB/R[1-9]|L[1-9]/s
8	PB/L5|R4/b-PB/L[1-9]|R[1-9]/s
9	PB/R8|L1|L9/b-PB/R[1-9]|L[1-9]/s
10	PB/L8|R1|R9/b-PB/L[1-9]|R[1-9]/s

Table 3: PB local neurons.

	Label
1	EB/([L7,R8],[P,M],[1-4])/s-EB/(L8,[P,M],[1-4])/b-LAL/LDG/b-PB/R1|L1/b
2	EB/([R7,L8],[P,M],[1-4])/s-EB/(R8,[P,M],[1-4])/b-lal/RDG/b-PB/L1|R1/b
3	EB/([R5,R7],[P,M],[1-4])/s-EB/(R6,[P,M],[1-4])/b-lal/RVG/b-PB/L2/b
4	EB/([R3,R5],[P,M],[1-4])/s-EB/(R4,[P,M],[1-4])/b-lal/RDG/b-PB/L3/b
5	EB/([R1,R3],[P,M],[1-4])/s-EB/(R2,[P,M],[1-4])/b-lal/RVG/b-PB/L4/b
6	EB/([L2,R1],[P,M],[1-4])/s-EB/(L1,[P,M],[1-4])/b-lal/RDG/b-PB/L5/b
7	EB/([L4,L2],[P,M],[1-4])/s-EB/(L3,[P,M],[1-4])/b-lal/RVG/b-PB/L6/b
8	EB/([L6,L4],[P,M],[1-4])/s-EB/(L5,[P,M],[1-4])/b-lal/RDG/b-PB/L7/b
9	EB/([L8,L6],[P,M],[1-4])/s-EB/(L7,[P,M],[1-4])/b-lal/RVG/b-PB/L8/b
10	EB/(L8,P,[1-4])/s-EB/(L8,P,[1-4])/b-lal/RDG/b-PB/L9/b
11	EB/([L5,L7],[P,M],[1-4])/s-EB/(L6,[P,M],[1-4])/b-LAL/LVG/b-PB/R2/b
12	EB/([L3,L5],[P,M],[1-4])/s-EB/(L4,[P,M],[1-4])/b-LAL/LDG/b-PB/R3/b
13	EB/([L1,L3],[P,M],[1-4])/s-EB/(L2,[P,M],[1-4])/b-LAL/LVG/b-PB/R4/b
14	EB/([R2,L1],[P,M],[1-4])/s-EB/(R1,[P,M],[1-4])/b-LAL/LDG/b-PB/R5/b
15	EB/([R4,R2],[P,M],[1-4])/s-EB/(R3,[P,M],[1-4])/b-LAL/LVG/b-PB/R6/b
16	EB/([R6,R4],[P,M],[1-4])/s-EB/(R5,[P,M],[1-4])/b-LAL/LDG/b-PB/R7/b
17	EB/([R8,R6],[P,M],[1-4])/s-EB/(R7,[P,M],[1-4])/b-LAL/LVG/b-PB/R8/b
18	EB/(R8,P,[1-4])/s-EB/(R8,P,[1-4])/b-LAL/LDG/b-PB/R9/b

Table 4: EB-LAL-PB neurons.

	Label
1	FB/(3,L4)/s-FB/(3,R1)/b
2	FB/(3,L3)/s-FB/(3,R2)/b
3	FB/(3,L2)/s-FB/(3,R3)/b
4	FB/(3,L1)/s-FB/(3,R4)/b
5	FB/(3,R4)/s-FB/(3,L1)/b
6	FB/(3,R3)/s-FB/(3,L2)/b
7	FB/(3,R2)/s-FB/(3,L3)/b
8	FB/(3,R1)/s-FB/(3,L4)/b

Table 5: One identified class of FB local neurons.

	Label
1	IB/L/s-LAL/LHB/s-PS/L/s-PB/L[2-9]|R[2-9]/b
2	ib/R/s-lal/RHB/s-ps/R/s-PB/L[2-9]|R[2-9]/b

Table 6: IB-LAL-PS-PB neurons.

	Label
1	PB/L1/s-EB/5/b-lal/RDG/b
2	PB/L2/s-EB/4/b-lal/RVG/b
3	PB/L3/s-EB/3/b-lal/RDG/b
4	PB/L4/s-EB/2/b-lal/RVG/b
5	PB/L5/s-EB/1/b-lal/RDG/b
6	PB/L6/s-EB/8/b-lal/RVG/b
7	PB/L7/s-EB/7/b-lal/RDG/b
8	PB/L8/s-EB/6/b-lal/RVG/b
9	PB/R1/s-EB/5/b-LAL/LDG/b
10	PB/R2/s-EB/4/b-LAL/LVG/b
11	PB/R3/s-EB/3/b-LAL/LDG/b
12	PB/R4/s-EB/2/b-LAL/LVG/b
13	PB/R5/s-EB/1/b-LAL/LDG/b
14	PB/R6/s-EB/8/b-LAL/LVG/b
15	PB/R7/s-EB/7/b-LAL/LDG/b
16	PB/R8/s-EB/6/b-LAL/LVG/b

Table 7: PB-EB-LAL neurons.

	Label
1	PB/L9/s-EB/6/b-no/(1,R)/b
2	PB/L8/s-EB/7/b-no/(1,R)/b
3	PB/L7/s-EB/8/b-no/(1,R)/b
4	PB/L6/s-EB/1/b-no/(1,R)/b
5	PB/L5/s-EB/2/b-no/(1,R)/b
6	PB/L4/s-EB/3/b-no/(1,R)/b
7	PB/L3/s-EB/4/b-no/(1,R)/b
8	PB/L2/s-EB/5/b-no/(1,R)/b
9	PB/R2/s-EB/5/b-NO/(1,L)/b
10	PB/R3/s-EB/6/b-NO/(1,L)/b
11	PB/R4/s-EB/7/b-NO/(1,L)/b
12	PB/R5/s-EB/8/b-NO/(1,L)/b
13	PB/R6/s-EB/1/b-NO/(1,L)/b
14	PB/R7/s-EB/2/b-NO/(1,L)/b
15	PB/R8/s-EB/3/b-NO/(1,L)/b
16	PB/R9/s-EB/4/b-NO/(1,L)/b

Table 8: PB-EB-NO neurons.

	Label
1	PB/L1/s-FB/([3-4],L4)/s-CRE/LRB/b
2	PB/R1/s-FB/([3-4],L4)/s-CRE/LRB/b
3	PB/R2/s-FB/([3-4],L3)/s-CRE/LRB/b
4	PB/R3/s-FB/([3-4],L2)/s-CRE/LRB/b
5	PB/R4/s-FB/([3-4],L1)/s-CRE/LRB/b
6	PB/R4/s-FB/([3-4],R1)/s-CRE/LRB/b
7	PB/R5/s-FB/([3-4],R2)/s-CRE/LRB/b
8	PB/R6/s-FB/([3-4],R3)/s-CRE/LRB/b
9	PB/R7/s-FB/([3-4],R4)/s-CRE/LRB/b
10	PB/L7/s-FB/([3-4],L4)/s-cre/RRB/b
11	PB/L6/s-FB/([3-4],L3)/s-cre/RRB/b
12	PB/L5/s-FB/([3-4],L2)/s-cre/RRB/b
13	PB/L4/s-FB/([3-4],L1)/s-cre/RRB/b
14	PB/L4/s-FB/([3-4],R1)/s-cre/RRB/b
15	PB/L3/s-FB/([3-4],R2)/s-cre/RRB/b
16	PB/L2/s-FB/([3-4],R3)/s-cre/RRB/b
17	PB/L1/s-FB/([3-4],R4)/s-cre/RRB/b
18	PB/R1/s-FB/([3-4],R4)/s-cre/RRB/b

Table 9: PB-FB-CRE neurons.

	Label
1	PB/L2/s-FB/(1,R4)/b-no/(3,RP)/b
2	PB/L3/s-FB/(1,R4)/b-no/(3,RP)/b
3	PB/L4/s-FB/(1,R3)/b-no/(3,RP)/b
4	PB/L5/s-FB/(1,R2)/b-no/(3,RP)/b
5	PB/L6/s-FB/(1,R1)|(1,L1)/b-no/(3,RP)/b
6	PB/L7/s-FB/(1,L2)/b-no/(3,RP)/b
7	PB/L8/s-FB/(1,L3)/b-no/(3,RP)/b
8	PB/L9/s-FB/(1,L4)/b-no/(3,RP)/b
9	PB/R2/s-FB/(1,L4)/b-NO/(3,LP)/b
10	PB/R3/s-FB/(1,L4)/b-NO/(3,LP)/b
11	PB/R4/s-FB/(1,L3)/b-NO/(3,LP)/b
12	PB/R5/s-FB/(1,L2)/b-NO/(3,LP)/b
13	PB/R6/s-FB/(1,L1)|(1,R1)/b-NO/(3,LP)/b
14	PB/R7/s-FB/(1,R2)/b-NO/(3,LP)/b
15	PB/R8/s-FB/(1,R3)/b-NO/(3,LP)/b
16	PB/R9/s-FB/(1,R4)/b-NO/(3,LP)/b

Table 10: PB-FB-NO neurons innervating region (3,P) of NO.

	Label
1	PB/L2/s-FB/(1,R4)/b-no/(3,RM)/b
2	PB/L3/s-FB/(1,R4)/b-no/(3,RM)/b
3	PB/L4/s-FB/(1,R3)/b-no/(3,RM)/b
4	PB/L5/s-FB/(1,R2)/b-no/(3,RM)/b
5	PB/L6/s-FB/(1,R1)|(1,L1)/b-no/(3,RM)/b
6	PB/L7/s-FB/(1,L2)/b-no/(3,RM)/b
7	PB/L8/s-FB/(1,L3)/b-no/(3,RM)/b
8	PB/L9/s-FB/(1,L4)/b-no/(3,RM)/b
9	PB/R2/s-FB/(1,L4)/b-NO/(3,LM)/b
10	PB/R3/s-FB/(1,L4)/b-NO/(3,LM)/b
11	PB/R4/s-FB/(1,L3)/b-NO/(3,LM)/b
12	PB/R5/s-FB/(1,L2)/b-NO/(3,LM)/b
13	PB/R6/s-FB/(1,L1)|(1,R1)/b-NO/(3,LM)/b
14	PB/R7/s-FB/(1,R2)/b-NO/(3,LM)/b
15	PB/R8/s-FB/(1,R3)/b-NO/(3,LM)/b
16	PB/R9/s-FB/(1,R4)/b-NO/(3,LM)/b

Table 11: PB-FB-NO neurons innervating region (3,M) of NO.

	Label
1	PB/L2/s-FB/(2,R4)/b-no/(3,RA)/b
2	PB/L3/s-FB/(2,R4)/b-no/(3,RA)/b
3	PB/L4/s-FB/(2,R3)/b-no/(3,RA)/b
4	PB/L5/s-FB/(2,R2)/b-no/(3,RA)/b
5	PB/L6/s-FB/(2,R1)|(1,L1)/b-no/(3,RA)/b
6	PB/L7/s-FB/(2,L2)/b-no/(3,RA)/b
7	PB/L8/s-FB/(2,L3)/b-no/(3,RA)/b
8	PB/L9/s-FB/(2,L4)/b-no/(3,RA)/b
9	PB/R2/s-FB/(2,L4)/b-NO/(3,LA)/b
10	PB/R3/s-FB/(2,L4)/b-NO/(3,LA)/b
11	PB/R4/s-FB/(2,L3)/b-NO/(3,LA)/b
12	PB/R5/s-FB/(2,L2)/b-NO/(3,LA)/b
13	PB/R6/s-FB/(2,L1)|(1,R1)/b-NO/(3,LA)/b
14	PB/R7/s-FB/(2,R2)/b-NO/(3,LA)/b
15	PB/R8/s-FB/(2,R3)/b-NO/(3,LA)/b
16	PB/R9/s-FB/(2,R4)/b-NO/(3,LA)/b

Table 12: PB-FB-NO neurons innervating region (3,A) of NO.

	Label
1	PB/L2/s-FB/(3,R4)/b-no/(2,RD)/b
2	PB/L3/s-FB/(3,R4)/b-no/(2,RD)/b
3	PB/L4/s-FB/(3,R3)/b-no/(2,RD)/b
4	PB/L5/s-FB/(3,R2)/b-no/(2,RD)/b
5	PB/L6/s-FB/(3,R1)|(1,L1)/b-no/(2,RD)/b
6	PB/L7/s-FB/(3,L2)/b-no/(2,RD)/b
7	PB/L8/s-FB/(3,L3)/b-no/(2,RD)/b
8	PB/L9/s-FB/(3,L4)/b-no/(2,RD)/b
9	PB/R2/s-FB/(3,L4)/b-NO/(2,LD)/b
10	PB/R3/s-FB/(3,L4)/b-NO/(2,LD)/b
11	PB/R4/s-FB/(3,L3)/b-NO/(2,LD)/b
12	PB/R5/s-FB/(3,L2)/b-NO/(2,LD)/b
13	PB/R6/s-FB/(3,L1)|(1,R1)/b-NO/(2,LD)/b
14	PB/R7/s-FB/(3,R2)/b-NO/(2,LD)/b
15	PB/R8/s-FB/(3,R3)/b-NO/(2,LD)/b
16	PB/R9/s-FB/(3,R4)/b-NO/(2,LD)/b

Table 13: PB-FB-NO neurons innervating region (2,D) of NO.

	Label
1	PB/L2/s-FB/(3,R4)/b-no/(2,RV)/b
2	PB/L3/s-FB/(3,R4)/b-no/(2,RV)/b
3	PB/L4/s-FB/(3,R3)/b-no/(2,RV)/b
4	PB/L5/s-FB/(3,R2)/b-no/(2,RV)/b
5	PB/L6/s-FB/(3,R1)|(1,L1)/b-no/(2,RV)/b
6	PB/L7/s-FB/(3,L2)/b-no/(2,RV)/b
7	PB/L8/s-FB/(3,L3)/b-no/(2,RV)/b
8	PB/L9/s-FB/(3,L4)/b-no/(2,RV)/b
9	PB/R2/s-FB/(3,L4)/b-NO/(2,LV)/b
10	PB/R3/s-FB/(3,L4)/b-NO/(2,LV)/b
11	PB/R4/s-FB/(3,L3)/b-NO/(2,LV)/b
12	PB/R5/s-FB/(3,L2)/b-NO/(2,LV)/b
13	PB/R6/s-FB/(3,L1)|(1,R1)/b-NO/(2,LV)/b
14	PB/R7/s-FB/(3,R2)/b-NO/(2,LV)/b
15	PB/R8/s-FB/(3,R3)/b-NO/(2,LV)/b
16	PB/R9/s-FB/(3,R4)/b-NO/(2,LV)/b

Table 14: PB-FB-NO neurons innervating region (2,V) of NO.

	Label
1	PB/L1/s-FB/(2,L[3-4])/s-LAL/LHB/b
2	PB/R1|L1/s-FB/(2,L[3-4])/s-LAL/LHB/b
3	PB/R1/s-FB/(2,L[2-3])/s-LAL/LHB/b
4	PB/R2/s-FB/(2,L[1-2])/s-LAL/LHB/b
5	PB/R3/s-FB/(2,[L1,R1])/s-LAL/LHB/b
6	PB/R4/s-FB/(2,R[1-2])/s-LAL/LHB/b
7	PB/R5/s-FB/(2,R[2-3])/s-LAL/LHB/b
8	PB/R6/s-FB/(2,R[3-4])/s-LAL/LHB/b
9	PB/R7/s-FB/(2,R[3-4])/s-LAL/LHB/b
10	PB/L7/s-FB/(2,L[3-4])/s-lal/RHB/b
11	PB/L6/s-FB/(2,L[3-4])/s-lal/RHB/b
12	PB/L5/s-FB/(2,L[2-3])/s-lal/RHB/b
13	PB/L4/s-FB/(2,L[1-2])/s-lal/RHB/b
14	PB/L3/s-FB/(2,[R1,L1])/s-lal/RHB/b
15	PB/L2/s-FB/(2,R[1-2])/s-lal/RHB/b
16	PB/L1/s-FB/(2,R[2-3])/s-lal/RHB/b
17	PB/L1|R1/s-FB/(2,R[3-4])/s-lal/RHB/b
18	PB/R1/s-FB/(2,R[3-4])/s-lal/RHB/b

Table 15: PB-FB-LAL neurons.

	Label
1	PB/L1/s-FB/([1-4],L[3-4])/s-LAL/LHB/b
2	PB/L1|R1/s-FB/([1-4],L[2-3])/s-LAL/LHB/b
3	PB/R1/s-FB/([1-4],L[1-2])/s-LAL/LHB/b
4	PB/R2/s-FB/([1-4],[L1,R1])/s-LAL/LHB/b
5	PB/R3/s-FB/([1-4],R[1-2])/s-LAL/LHB/b
6	PB/R4/s-FB/([1-4],R[2-3])/s-LAL/LHB/b
7	PB/R5/s-FB/([1-4],R[3-4])/s-LAL/LHB/b
8	PB/R6/s-FB/([1-4],R[3-4])/s-LAL/LHB/b
9	PB/L6/s-FB/([1-4],L[3-4])/s-lal/RHB/b
10	PB/L5/s-FB/([1-4],L[3-4])/s-lal/RHB/b
11	PB/L4/s-FB/([1-4],L[2-3])/s-lal/RHB/b
12	PB/L3/s-FB/([1-4],L[1-2])/s-lal/RHB/b
13	PB/L2/s-FB/([1-4],[L1,R1])/s-lal/RHB/b
14	PB/L1/s-FB/([1-4],R[1-2])/s-lal/RHB/b
15	PB/L1|R1/s-FB/([1-4],R[2-3])/s-lal/RHB/b
16	PB/R1/s-FB/([1-4],R[3-4])/s-lal/RHB/b

Table 16: PB-FB-LAL neurons.

	Label
1	PB/L3/s-FB/([1-4],L[3-4])/s-LAL/LHB/b-lal/RHB/b
2	PB/L1/s-FB/([1-4],L[1-2])/s-LAL/LHB/b-lal/RHB/b
3	PB/R1/s-FB/([1-4],R[1-2])/s-LAL/LHB/b-lal/RHB/b
4	PB/R3/s-FB/([1-4],R[3-4])/s-LAL/LHB/b-lal/RHB/b

Table 17: PB-FB-LAL neurons.

	Label
1	WED/L/s-PS/L/s-PB/L2|L3/b
2	WED/L/s-PS/L/s-PB/L3|L4/b
3	WED/L/s-PS/L/s-PB/L4|L5/b
4	WED/L/s-PS/L/s-PB/L5|L6/b
5	WED/L/s-PS/L/s-PB/L6|L7/b
6	WED/L/s-PS/L/s-PB/L7|L8/b
7	WED/L/s-PS/L/s-PB/L8|L9/b
8	wed/R/s-ps/R/s-PB/L2|L3/b
9	wed/R/s-ps/R/s-PB/L3|L4/b
10	wed/R/s-ps/R/s-PB/L4|L5/b
11	wed/R/s-ps/R/s-PB/L5|L6/b
12	wed/R/s-ps/R/s-PB/L6|L7/b
13	wed/R/s-ps/R/s-PB/L7|L8/b
14	wed/R/s-ps/R/s-PB/L8|L9/b
15	WED/L/s-PS/L/s-PB/R2|R3/b
16	WED/L/s-PS/L/s-PB/R3|R4/b
17	WED/L/s-PS/L/s-PB/R4|R5/b
18	WED/L/s-PS/L/s-PB/R5|R6/b
19	WED/L/s-PS/L/s-PB/R6|R7/b
20	WED/L/s-PS/L/s-PB/R7|R8/b
21	WED/L/s-PS/L/s-PB/R8|R9/b
22	wed/R/s-ps/R/s-PB/R2|R3/b
23	wed/R/s-ps/R/s-PB/R3|R4/b
24	wed/R/s-ps/R/s-PB/R4|R5/b
25	wed/R/s-ps/R/s-PB/R5|R6/b
26	wed/R/s-ps/R/s-PB/R6|R7/b
27	wed/R/s-ps/R/s-PB/R7|R8/b
28	wed/R/s-ps/R/s-PB/R8|R9/b

Table 18: WED-PS-PB neurons.

Ring Neuron Name	**Microglomerulus Range**	**Label**
R1	1..16	BU/Lx/s-EB/(LR[1-8],[M,A],1)/b
		bu/Rx/s-EB/(LR[1-8],[M,A],1)/b
R2	33..48	BU/Lx/s-EB/(LR[1-8],A,[3,4])/b
		bu/Rx/s-EB/(LR[1-8],A,[3,4])/b
R3	17..32	BU/Lx/s-EB/(LR[1-8],A,[1,2])/b
		bu/Rx/s-EB/(LR[1-8],A,[1,2])/b
R4m	49..64	BU/Lx/s-EB/(LR[1-8],A,4)/b
		bu/Rx/s-EB/(LR[1-8],A,4)/b
R4d	65..80	BU/Lx/s-EB/(LR[1-8],M,4)/b
		bu/Rx/s-EB/(LR[1-8],M,4)/b

Table 19: Hypothesized arborizations of BU-EB neurons. Each microglomerulus corresponds to a single neuron, where the "x" character in the neuron label is replaced with the microglomerulus number.

	Label
1	FB/(1,L4)/s-FB/(1,R1)/b
2	FB/(1,L3)/s-FB/(1,R2)/b
3	FB/(1,L2)/s-FB/(1,R3)/b
4	FB/(1,L1)/s-FB/(1,R4)/b
5	FB/(1,R4)/s-FB/(1,L1)/b
6	FB/(1,R3)/s-FB/(1,L2)/b
7	FB/(1,R2)/s-FB/(1,L3)/b
8	FB/(1,R1)/s-FB/(1,L4)/b
9	FB/(2,L4)/s-FB/(2,R1)/b
10	FB/(2,L3)/s-FB/(2,R2)/b
11	FB/(2,L2)/s-FB/(2,R3)/b
12	FB/(2,L1)/s-FB/(2,R4)/b
13	FB/(2,R4)/s-FB/(2,L1)/b
14	FB/(2,R3)/s-FB/(2,L2)/b
15	FB/(2,R2)/s-FB/(2,L3)/b
16	FB/(2,R1)/s-FB/(2,L4)/b
17	FB/(4,L4)/s-FB/(4,R1)/b
18	FB/(4,L3)/s-FB/(4,R2)/b
19	FB/(4,L2)/s-FB/(4,R3)/b
20	FB/(4,L1)/s-FB/(4,R4)/b
21	FB/(4,R4)/s-FB/(4,L1)/b
22	FB/(4,R3)/s-FB/(4,L2)/b
23	FB/(4,R2)/s-FB/(4,L3)/b
24	FB/(4,R1)/s-FB/(4,L4)/b
25	FB/(5,L4)/s-FB/(5,R1)/b
26	FB/(5,L3)/s-FB/(5,R2)/b
27	FB/(5,L2)/s-FB/(5,R3)/b
28	FB/(5,L1)/s-FB/(5,R4)/b
29	FB/(5,R4)/s-FB/(5,L1)/b
30	FB/(5,R3)/s-FB/(5,L2)/b
31	FB/(5,R2)/s-FB/(5,L3)/b
32	FB/(5,R1)/s-FB/(5,L4)/b

Table 20: Hypothesized FB local neurons linking segments in layers 1, 2, 4, and 5, respectively.

	Label
1	FB/(1,L4)/s-FB/(1,L3)/b
2	FB/(1,L3)/s-FB/(1,L2)/b
3	FB/(1,L2)/s-FB/(1,L1)/b
4	FB/(1,L1)/s-FB/(1,R1)/b
5	FB/(1,R1)/s-FB/(1,R2)/b
6	FB/(1,R2)/s-FB/(1,R3)/b
7	FB/(1,R3)/s-FB/(1,R4)/b
8	FB/(2,L4)/s-FB/(2,L3)/b
9	FB/(2,L3)/s-FB/(2,L2)/b
10	FB/(2,L2)/s-FB/(2,L1)/b
11	FB/(2,L1)/s-FB/(2,R1)/b
12	FB/(2,R1)/s-FB/(2,R2)/b
13	FB/(2,R2)/s-FB/(2,R3)/b
14	FB/(2,R3)/s-FB/(2,R4)/b
15	FB/(3,L4)/s-FB/(3,L3)/b
16	FB/(3,L3)/s-FB/(3,L2)/b
17	FB/(3,L2)/s-FB/(3,L1)/b
18	FB/(3,L1)/s-FB/(3,R1)/b
19	FB/(3,R1)/s-FB/(3,R2)/b
20	FB/(3,R2)/s-FB/(3,R3)/b
21	FB/(3,R3)/s-FB/(3,R4)/b
22	FB/(4,L4)/s-FB/(4,L3)/b
23	FB/(4,L3)/s-FB/(4,L2)/b
24	FB/(4,L2)/s-FB/(4,L1)/b
25	FB/(4,L1)/s-FB/(4,R1)/b
26	FB/(4,R1)/s-FB/(4,R2)/b
27	FB/(4,R2)/s-FB/(4,R3)/b
28	FB/(4,R3)/s-FB/(4,R4)/b
29	FB/(5,L4)/s-FB/(5,L3)/b
30	FB/(5,L3)/s-FB/(5,L2)/b
31	FB/(5,L2)/s-FB/(5,L1)/b
32	FB/(5,L1)/s-FB/(5,R1)/b
33	FB/(5,R1)/s-FB/(5,R2)/b
34	FB/(5,R2)/s-FB/(5,R3)/b
35	FB/(5,R3)/s-FB/(5,R4)/b

Table 21: Hypothesized FB local neurons linking adjacent segments within the same layer in layers 1-5.

	Label
1	FB/(5,L1)/s-FB/(4,L1)/b
2	FB/(4,L1)/s-FB/(3,L1)/b
3	FB/(3,L1)/s-FB/(2,L1)/b
4	FB/(2,L1)/s-FB/(1,L1)/b
5	FB/(5,L2)/s-FB/(4,L2)/b
6	FB/(4,L2)/s-FB/(3,L2)/b
7	FB/(3,L2)/s-FB/(2,L2)/b
8	FB/(2,L2)/s-FB/(1,L2)/b
9	FB/(5,L3)/s-FB/(4,L3)/b
10	FB/(4,L3)/s-FB/(3,L3)/b
11	FB/(3,L3)/s-FB/(2,L3)/b
12	FB/(2,L3)/s-FB/(1,L3)/b
13	FB/(5,L4)/s-FB/(4,L4)/b
14	FB/(4,L4)/s-FB/(3,L4)/b
15	FB/(3,L4)/s-FB/(2,L4)/b
16	FB/(2,L4)/s-FB/(1,L4)/b
17	FB/(5,R1)/s-FB/(4,R1)/b
18	FB/(4,R1)/s-FB/(3,R1)/b
19	FB/(3,R1)/s-FB/(2,R1)/b
20	FB/(2,R1)/s-FB/(1,R1)/b
21	FB/(5,R2)/s-FB/(4,R2)/b
22	FB/(4,R2)/s-FB/(3,R2)/b
23	FB/(3,R2)/s-FB/(2,R2)/b
24	FB/(2,R2)/s-FB/(1,R2)/b
25	FB/(5,R3)/s-FB/(4,R3)/b
26	FB/(4,R3)/s-FB/(3,R3)/b
27	FB/(3,R3)/s-FB/(2,R3)/b
28	FB/(2,R3)/s-FB/(1,R3)/b
29	FB/(5,R4)/s-FB/(4,R4)/b
30	FB/(4,R4)/s-FB/(3,R4)/b
31	FB/(3,R4)/s-FB/(2,R4)/b
32	FB/(2,R4)/s-FB/(1,R4)/b

Table 22: Hypothesized FB local neurons linking adjacent layers within the same segment for layers 1-5.

	Label
1	FB/(1,L1)/sb-FB/(8,L1)/sb
2	FB/(1,L2)/sb-FB/(8,L2)/sb
3	FB/(1,L3)/sb-FB/(8,L3)/sb
4	FB/(1,L4)/sb-FB/(8,L4)/sb
5	FB/(1,R1)/sb-FB/(8,R1)/sb
6	FB/(1,R2)/sb-FB/(8,R2)/sb
7	FB/(1,R3)/sb-FB/(8,R3)/sb
8	FB/(1,R4)/sb-FB/(8,R4)/sb
9	FB/(2,L1)/sb-FB/(7,L1)/sb
10	FB/(2,L2)/sb-FB/(7,L2)/sb
11	FB/(2,L3)/sb-FB/(7,L3)/sb
12	FB/(2,L4)/sb-FB/(7,L4)/sb
13	FB/(2,R1)/sb-FB/(7,R1)/sb
14	FB/(2,R2)/sb-FB/(7,R2)/sb
15	FB/(2,R3)/sb-FB/(7,R3)/sb
16	FB/(2,R4)/sb-FB/(7,R4)/sb

Table 23: Hypothesized FB local neurons linking nonadjacent layers within the same segment.

	Label
1	PB/R8|R9/b-PB/R[4-8]/s
2	PB/L8|L9/b-PB/L[4-8]/s
3	PB/L2/b-PB/L[1-9]/s
4	PB/R7/b-PB/R[1-9]/s
5	PB/L7/b-PB/L[1-9]/s
6	PB/R2/b-PB/R[1-9]/s
7	PB/L3/b-PB/L[1-9]/s
8	PB/R6/b-PB/R[1-9]/s
9	PB/L6/b-PB/L[1-9]/s
10	PB/R3/b-PB/R[1-9]/s
11	PB/L4/b-PB/L[1-9]/s
12	PB/R5/b-PB/R[1-9]/s
13	PB/L5/b-PB/L[1-9]/s
14	PB/R4/b-PB/R[1-9]/s
15	PB/R8/b-PB/R[1-9]/s
16	PB/L1|L9/b-PB/L[1-9]/s
17	PB/L8/b-PB/L[1-9]/s
18	PB/R1|R9/b-PB/R[1-9]/s

Table 24: Hypothesized PB local neurons in the *no bridge* mutant.