Analysis of Computational Tools for Applications in the Fruit Fly Brain Observatory

Noah Parker

Bionet Lab, Fu Foundation, Electrical Engineering, Columbia University, NYC

Abstract

Motivation: To create methods to transfer morphology information from FBL to modeling and simulation tools, and to analyze the effectiveness of those tools.
Results: Brian2 provides a more accessible and easy-to-use interface for defining models and is more memory-efficient, but NetPyNE is preferred for situations which call for greater simulation power and speed.
Availability: https://github.com/NJParker415/NeuroNLP_to_Brian_Netpyne

1. Introduction

Computational neuroscience has emerged as a vital field for understanding the complex workings of the brain. With advancements in computing power and modeling techniques, neural simulators have become essential tools in simulating and analyzing neural activity. These simulators provide researchers with a computational framework to investigate the dynamics of neural networks, study brain disorders, and explore potential therapeutic interventions.

Among the diverse range of neural simulators available, Brian2 and NetPyNE have gained significant attention for their capabilities and user-friendly interfaces. Brian2 is a popular Python-based simulator known for its simplicity and flexibility, allowing users to describe and simulate neural models in a concise and intuitive manner. On the other hand, NetPyNE, built on the NEURON simulator, provides a powerful platform for simulating large-scale neural networks, with a focus on network architecture and connectivity.

The purpose of this report is to conduct a comparative analysis of Brian2 and NetPyNE, aiming to provide insights into their features, performance, and community support. By evaluating these simulators side by side, I aim to assist researchers, computational neuroscientists, and modelers in selecting the most suitable simulator for their specific research needs. In particular, I seek to explore how these tools might best be used in the context of simulating neural circuits generated from NeuroNLP queries.

To accomplish this, I will follow a systematic methodology involving the installation and setup of both simulators, the design of a comprehensive comparative analysis, and the selection of benchmark models and datasets. I will evaluate key features such as neuron and synapse models, network connectivity options, input stimulus generation, parallel simulation support, and integration with other tools and libraries.

Additionally, I will perform performance evaluations, including execution time, memory usage, and scalability analysis.

Furthermore, the user experience and documentation of both simulators will be examined, considering factors such as user interface design, ease of use, and availability of documentation resources. Additionally, I will analyze the active user communities and support channels associated with Brian2 and NetPyNE, highlighting the benefits and challenges of each community.

It is important to acknowledge that both simulators have their limitations and face common challenges in modeling complex neural systems. Therefore, I will also discuss these limitations, challenges, and future research directions for both Brian2 and NetPyNE.

2. Methodology

2.1 Installation and Setup

All code was executed in a custom FBL Anaconda environment, with the following packages and their associated dependencies installed:

- Brian2, for building models and running simulations
- NetPyNE, for building models and running simulations
- pyNeuroML, to facilitate transfer of models from NetPyNE to other modeling tools

The technical specifications of the device used to run all tests, and the full list of dependencies can be found in Appendix 3.

2.2 Design of Comparative Analysis

Three benchmarks were considered when comparing the functionality of Brian2 and NetPyNE, being execution time, memory usage, and scalability.

2.3 Selection of Benchmark Model and Dataset

For this project, Drosophila visual columns were selected as the model benchmark due to their ready availability from FBL queries, options for scalability, and relative simplicity compared to simulating the entire Drosophila medulla.

Since the visual columns of Drosophila exhibit modular organization, where each column represents a functional unit responsible for processing visual information, it is easier to design benchmark models of varying sizes and complexity, allowing for scalability analysis of the simulators. By increasing the number of columns simulated, the scalability of the simulators can be assessed, providing insights into their performance under different computational demands.

Additionally, due to computational limits imposed on this project caused by a lack of sufficiently powerful hardware, the comparatively few number of neurons in each visual column allow for computational experiments to be executed without demanding an excessive amount of time.

While the ability to directly draw neuron morphologies from NeuroNLP was implemented in this project's codebase, this level of granularity was not simulated due to

missing information in the database's morphology data. Specifically, many components of neuron morphology are not categorized as belonging to the axon, soma, etc. which prevented importing these models into NetPyNE. Instead, all neurons are defined as identical point-hodgkin-huxley neurons, with all synapses defined by identical differential equations. Though this does not create an accurate representation of the functionality of Drosophila visual columns, it is sufficient for the purposes of evaluating the performance metrics of different simulators.

To test these parameters, each simulator was tasked with running a simulation of a variable number of visual columns, ranging from 1 to 6. In each case, simulation time and memory usage were measured in order to determine simulator performance, with increasing computational and time costs determining scalability factor. Each network simulated a single second of neural activity, with a time step of 0.05 seconds. In each case, the Mi4 neuron in the "C" column was stimulated with an excitatory synaptic mechanism, with a firing rate of 10 Hz and 50% noise. Network diagrams visualizing each simulation can be seen in Figures 9.1-9.6 in Appendix 2.

3. Features and Functionality

3.1 Brian2 Overview

Brian2 is a highly flexible and adaptable simulation framework, specifically designed for the creation and simulation of neuron models. This framework excels in its ability to replicate intricate neural dynamics, making it particularly useful for accurately simulating the complex neuronal connections found in Drosophila. At the core of Brian2's functionality is its capacity to define neuronal models using differential equations, providing a significant level of customization that is crucial for detailed and precise simulations (Figure 1).

The framework's flexibility, supporting various numerical methods like the Euler method and the fourth-order Runge-Kutta method, is another critical advantage. This versatility enables Brian2 to model the complex interactions within Drosophila's neuronal circuits with enhanced accuracy. Such precision in simulation is vital for capturing the subtle nuances of neural behavior and interactions. Additionally, the visualization tools available in Brian2, including raster plots, voltage traces, and connectivity graphs, provide researchers with comprehensive insights into both the macro and micro dynamics of neural networks. These tools could be useful in determining the behavior of individual neurons and synapses, offering a deeper understanding of the overall behavior.

Figure 2 showcases a voltage trace of a Hodgkin-Huxley neuron using Brian2, representing the Mi4-C neuron stimulated during the performance evaluation test of a single visual column as described in Section 2.3.

```
cell_vars = {'area': -20000*umetre**2,
             'Cm': 1*ufarad*cm**-2 * 20000*umetre**2,
             'gl': 5e-5*siemens*cm**-2 * 20000*umetre**2,
             'El': -65*mV,
             'EK': -90*mV,
             'ENa': 50*mV,
             'g_na': 100*msiemens*cm**-2 * 20000*umetre**2,
             'g_kd': 30*msiemens*cm**-2 * 20000*umetre**2,
             'VT': -63*mV,
             'tau1': 0.1*ms,
             'tau2': 0.2*ms,
             'e': 0}

# Typical equations
neuron_eqs = '''
dv/dt = (gl*(El-v) - g_na*(m*m*m)*h*(v-ENa) - g_kd*(n*n*n*n)*(v-EK) + I)/Cm : volt
dm/dt = 0.32*(mV**-1)*4*mV/exprel((13.*mV-v+VT)/(4*mV))/ms*(1-m)-0.28*(mV**-1)*5*mV/exprel((v-VT-40.*mV)/(5*mV))/
dn/dt = 0.032*(mV**-1)*5*mV/exprel((15.*mV-v+VT)/(5*mV))/ms*(1.-n)-.5*exp((10.*mV-v+VT)/(40.*mV))/ms*n : 1
dh/dt = 0.128*exp((17.*mV-v+VT)/(18.*mV))/ms*(1.-h)-4./(1+exp((40.*mV-v+VT)/(5.*mV)))/ms*h : 1
I : amp
'''

syn_onpre = '''
v += 10*mV
'''

stim_dict = {'model': 'rates = 10*Hz : Hz', 'threshold': 'rand()<rates*dt'}
```

```
stim_mech = {'bkg': {'type': 'NetStim', 'rate': 10, 'noise': 0}} # Stimulation mechanism
default_syn = {'mod': 'Exp2Syn', 'tau1': 0.1, 'tau2': 5.0, 'e': 0} # Excitatory synapse mechanism

default_cell = {'secs': {}}
default_cell['secs']['soma'] = {'geom': {}, 'mechs': {}}
default_cell['secs']['soma']['geom'] = {'diam': 18.8, 'L': 18.8, 'Ra': 123.0}
default_cell['secs']['soma']['mechs']['hh'] = {'gnabar': 0.12, 'gkbar': 0.036, 'gl': 0.003, 'el': -70}
```

Figure 1: Example Definitions of Hodgkin-Huxley Neurons and Excitatory Mechanisms, Brian2 (Above) and NetPyNE (Below)

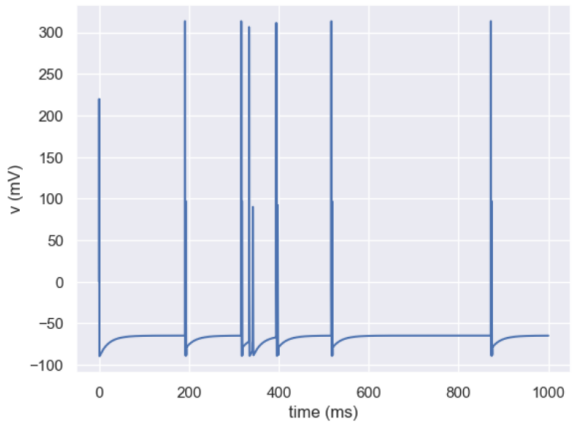

Figure 2: Sample Brian2 Voltage Trace

The integration of optimization tools in Brian2, which includes techniques like automatic differentiation and Bayesian optimization, presents a significant opportunity to elevate the accuracy and efficiency of neuronal models. This capability would allow for more precise tuning of models to mirror the real-world behaviors of Drosophila neurons. The support for integrating custom mechanisms into neuron models through custom equations is another potentially useful feature. By enabling the simulation of a broader range of biophysical processes, such as synaptic plasticity or dendritic integration, Brian2 could enrich the complexity and variability of the models, thereby capturing a more holistic view of neural function and interaction.

Another significant aspect of Brian2 is its potential to support hybrid models, combining different types of neuron models.

This capability could significantly broaden the scope of simulations, allowing for a more comprehensive exploration of neural network dynamics. Furthermore, the availability of tools for dynamic visualizations, which go beyond static representations, opens up new avenues for real-time exploration and interaction with neural models. Such interactive visualizations could provide a more immersive and intuitive experience in understanding and manipulating neural simulations, offering novel ways to study the complex neural networks present in Drosophila.

3.2 NetPyNE Overview

NetPyNE is a Python package for simulating and analyzing large-scale neural networks. It is designed to be flexible, easy to use, and extensible, allowing researchers to explore a wide range of neural network models and analyze their behavior.

In NetPyNE, users define neuronal models by specifying various parameters such as cell types, membrane properties, ion channel dynamics, and synaptic mechanisms. This is achieved through a combination of predefined templates and custom specifications, allowing for a high degree of precision and customization. The platform's flexibility extends to supporting a range of neuronal model types, from simple point neurons to more complex models with intricate dendritic and axonal structures. Figure 3 displays an example of this functionality, showcasing the voltage trace of a neuron using NEURON's Izhi2007a model, simulated using the same framework as in the performance evaluation tests.

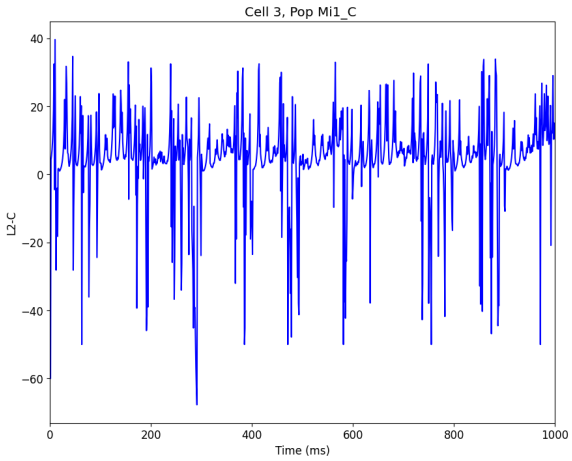

Figure 3: Izhi2007a Voltage Trace

One of the advantages of NetPyNE lies in its ability to interface with popular neuronal network simulators like NEURON, NEST, and Brian2. This compatibility could be extremely beneficial as a support tool when combined with FBL, offering the flexibility to choose a simulator that best matches the specific needs of network simulations. Such versatility in simulation environments can lead to more accurate and efficient research outcomes.

NetPyNE's user-friendly GUI (Figure 3) is another feature that could greatly aid research into Drosophila neuronal networks. The GUI's intuitive drag-and-drop interface might simplify the process of constructing and configuring intricate neural networks, which has the potential to streamline further research. This ease of model building, coupled with the GUI's capabilities for real-time visualization of network connectivity and activity, could provide researchers with a more dynamic and

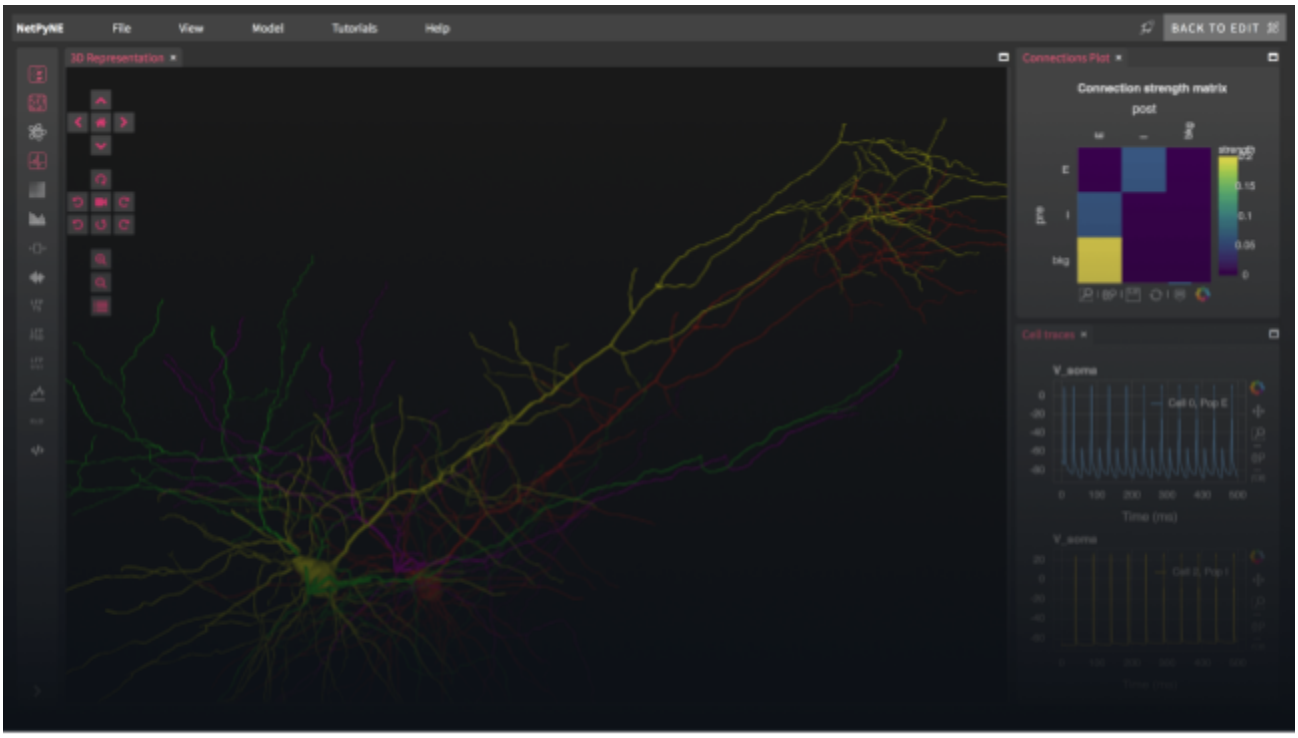

Figure 4: NetPyNE GUI View [2]

interactive approach to studying simulations of neuronal networks obtained from FBL. NetPyNE also supports a scripting interface separate from the GUI. By enabling the programmatic creation of network models and simulations, as displayed in this project, this feature could cater to the specific needs of researchers by allowing for a higher degree of control and customization in modeling.

Moreover, the scalability and efficiency offered by NetPyNE's support for parallel processing would be a significant advantage for simulating Drosophila neuronal networks. This feature would allow for the handling of large-scale simulations more efficiently, reducing computational times and enabling more extensive and complex studies.

Like Brian2, NetPyNE's comprehensive suite of built-in analysis tools, like spike raster plots, population activity plots, and connectivity matrices, would also be useful when analyzing Drosophila simulations. These tools would provide a broad spectrum of options for analyzing and visualizing the intricate dynamics within the fruit fly brain.

Figures 5-8 display sample visualizations from the single column performance test as described in Section 2.3, showing a voltage trace for the hodgkin-huxley neuron representing the Mi4-C neuron being stimulated at a rate of 10 Hz , a raster plot of all spike times across the test, a connectivity diagram of the generated network, and the network’s connectivity strength matrix respectively. It should be noted that in the generated connectivity diagram, the spatial positioning of neurons is not representative of that actually found in Drosophila motor

columns. However, NetPyNE is capable of generating 2D and 3D connectivity diagrams that showcase relative spatial positioning, a feature with potential to be very useful in further understanding neuronal connectivity.

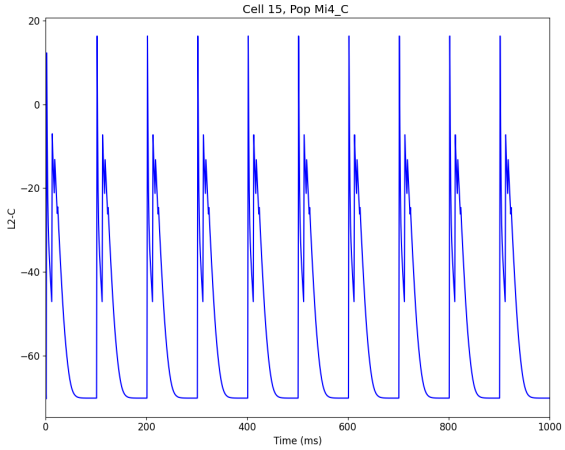

Figure 5: Sample NetPyNE Voltage Trace

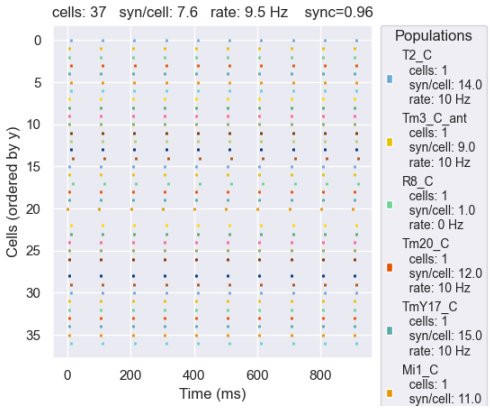

Figure 6: Sample NetPyNE Raster Plot

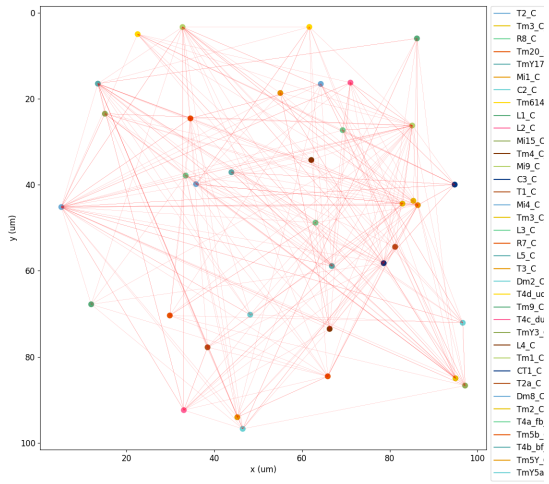

Figure 7: Sample NetPyNE Connectivity Diagram

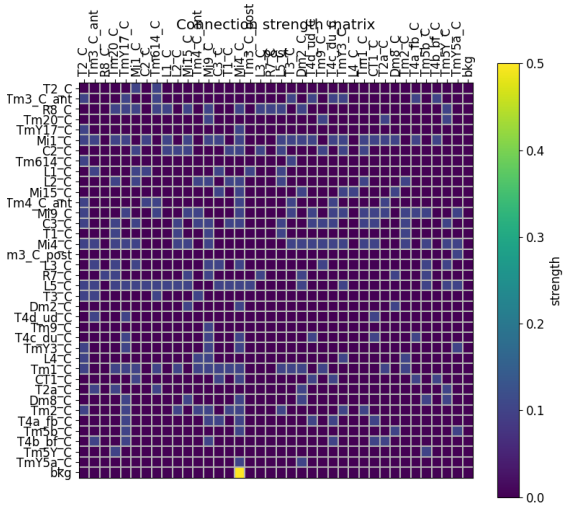

Figure 8: Sample NetPyNE Connection Strength Matrix

3.3 Features Leveraged in Testing

Not all key features were leveraged in testing the performance of each simulator. Since the primary purpose of testing was to evaluate computational efficiency, it was not

deemed necessary to include the totality of each simulator's functionality.

This project leverages Brian2's simulation framework and visualization tools in order to determine its computational ability. In order to show that Brian2 is capable of producing accurate simulation results, the user is capable of defining neuron models and connections using differential equations, and then identifying individual neurons for Brian2 to plot voltage traces of once the simulation is complete.

NetPyNE's simulation framework and analysis tools were leveraged in much the same way as they were for Brian2, with the addition of also plotting connectivity graphs and raster plots. The user is able to freely define neuronal and synaptic mechanisms, either through one of NEURON's predefined models or through importing a custom-defined model. Additionally, NetPyNE's ability to export models to other popular simulators is also included for future projects seeking to explore the capabilities of other neural simulators. It should be noted that, however, due to the way in which NetPyNE models are constructed, exporting multi-neuron models built this way cannot be reliably exported to the Brian2 simulator.

Examples of voltage traces and network connectivity graphs generated by each simulator can be found in Appendix 2, with images for all tests in the attached Github repository.

Though NetPyNE's GUI was not used in the analysis conducted for this project, functionality to export models to the GUI is also included. Examples of using these import and export functions are included in the attached GitHub repository. Since the code developed for this project was done in the context of programmatically creating maps of neuronal connections directly from the FlyBrainLab interface, the GUI was not necessary in order to construct the network. The code is, however, capable of exporting generated NetPyNE models to a format that can be imported into the GUI interface.

In the future, should this project's codebase be integrated into a larger system, it is recommended that it be modified to allow for additional flexibility in generating plots for visualization and analysis.

3.4 Comparative Analysis of Key Features

3.4.1 Neuron and Synapse Models

Brian2's framework, while not possessing a predefined library of neuron models, offers the flexibility to simulate various types of neurons, such as integrate-and-fire and Hodgkin-Huxley models, through the definition of custom neuron models using differential equations. This feature is crucial for accurately representing the diverse neuronal types found in the fruit fly brain, enabling researchers to tailor models to the specific characteristics of neurons observed in Drosophila.

NetPyNE's multi-compartmental models with detailed morphology, derived from NEURON, are crucial for simulating the complex structure of neurons in the fruit fly brain. These models enable a more biologically accurate representation, aiding in understanding how individual neuron

structure contributes to overall brain function.

Both simulators provide flexibility in modeling neurons, with Brian2 being advantageous for simplicity and ease of use, and NetPyNE offering more detailed and biophysically realistic models.

3.4.2 Network Connectivity Options

Brian2 provides flexible connectivity mechanisms, including spatial connectivity rules, distance-dependent connections, and random connectivity patterns. It allows users to define connectivity using equations, functions, or probabilistic rules. The spatial connectivity rules and distance-dependent connections in Brian2 can be used to accurately simulate the physical layout and synaptic connections of neurons in the fruit fly brain. This is particularly useful for studying how spatial arrangements influence neural network function.

NetPyNE, leveraging NEURON's connectivity features, offers a wide range of connectivity options, such as distance-dependent synapses, random connectivity, and connectivity based on experimental data or connectome information. NetPyNE also supports complex network topologies and allows for the specification of connectivity rules at different spatial scales. NetPyNE's ability to model large-scale networks and incorporate experimental connectivity data is especially beneficial for neural simulations. This allows for creating models that closely mirror the actual connectivity observed in the Drosophila brain, facilitating more accurate simulations and predictions.

3.4.3 Input Stimulus Generation

The flexibility in generating various input stimuli in Brian2, such as time-varying currents and noise sources, enables researchers to simulate a wide range of environmental inputs that a fruit fly might experience. This helps in studying how different stimuli affect brain activity.

NetPyNE provides similar input generation capabilities, allowing users to specify current or conductance inputs with customizable temporal patterns. Additionally, NetPyNE's ability to integrate sensory stimuli like visual or auditory inputs is particularly relevant for simulating FlyBrainLab queries, as it allows the simulation of how sensory information is processed in the fruit fly brain. This can aid in understanding sensory processing mechanisms and neural pathways.

3.4.4 Parallel Simulation Support

Both simulators provide parallel simulation support, with Brian2 offering GPU acceleration and NetPyNE leveraging NEURON's MPI-based parallelization.

3.4.5 Integration With Other Tools and Libraries

Both Brian2 and NetPyNE support integration with other tools and libraries commonly used in computational neuroscience. Brian2 integrates with popular scientific computing libraries, such as NumPy, SciPy, and matplotlib, enabling

efficient data handling, analysis, and visualization. It also provides support for exporting simulation data to HDF5 format for further analysis.

NetPyNE benefits from NEURON's ecosystem and compatibility with numerous tools and libraries. It integrates well with NEURON's collection of analysis and visualization tools, as well as with PyNN, a simulator-independent neural network modeling interface. NetPyNE also facilitates the exchange of data and models with other simulators through standardized formats like NeuroML, which would allow for easy transfer of Drosophila models to other simulators should the need arise.

3.5 Impact of Features on Design

The framework for converting FBL to each model was fundamentally the same, parsing an FBL query into dictionaries of neurons and the connections between them, and then constructing a model using each package's tools. Differences between the way in which models are constructed in NetPyNE or Brian2 are thus a direct byproduct of how each package defines models.

3.5.1 Brian2

Brian2's design architecture relies heavily on global variables and functions; as a result, workarounds were required to implement a system to programmatically generate connectivity graphs. In particular, it required the mass-storage of floating variables in the locals() directory and hardcoding in certain variables representing the voltage or current of neurons or synapses. As a result, this project's implementation requires that both neuron voltage and synaptic weight must be represented using specific variables.

A specific challenge encountered in this project with Brian2 has been its approach to visualization and recording. The necessity to define neuron recordings during network setup has limited the ability to adaptively modify the recording parameters in response to iterative analysis findings. While not a major issue for this project, as frequent model reconstructions were not strictly required, this has the potential to significantly add to the complexity and time requirements of future work.

Furthermore, while Brian2's flexibility in defining custom mechanisms using differential equations is advantageous, the lack of built-in validation for these mechanisms requires additional layers of testing and verification.

3.5.2 NetPyNE

The architecture of NetPyNE offered a distinct approach to the design and implementation of neural network models, contrasting with the challenges encountered in Brian2, particularly in generative model creation and scalability. NetPyNE's structured framework for model definitions streamlined the building of complex neural network models, allowing the user to construct neurons and synapses from predefined models. This structured approach has been particularly beneficial when dealing with the intricate neural circuits generated from NeuroNLP queries, facilitating a more organized and intuitive

construction process compared to the more programmatic and variable-intensive approach of Brian2.

One notable aspect of generating models with NetPyNE is that, due to how connections between neuron populations are defined, each individual neuron must be defined as a separate population and synapses are defined as projections between populations. This has the side effect of requiring that all neurons and synapses, even if using identical models, must be defined individually when generating models.

4. Performance Evaluation

4.1 Brian2

Number of Columns (Number of Neurons)	Computation Time (s)	Memory Usage (Mb)
1 (37)	220.3682	16.8957
2 (72)	429.4440	19.3543
3 (109)	656.4876	29.7745
4 (141)	931.6230	38.0816
5 (178)	1345.138	53.6348
6 (213)	1743.230	60.0838

Table 1: Brian2 Performance

4.2 NetPyNE

Number of Columns (Number of Neurons)	Computation Time (s)	Memory Usage (Mb)
1 (37)	0.44	16.7040
2 (72)	0.87	22.2862
3 (109)	1.66	33.7665
4 (141)	2.21	46.4320
5 (178)	3.05	60.5445
6 (213)	3.35	73.7898

Table 2: NetPyNE Performance

4.3 Analysis

Across all simulations, NetPyNE's computation time was significantly shorter than that of Brian2, at the cost of greater peak memory usage during the simulation. Both Brian2 and NetPyNE displayed roughly linear increases in computation time and peak memory usage with the total number of neurons simulated. However, Brian2's peak memory usage increased at a slower rate with the total number of simulated neurons compared to NetPyNE. This suggests that, though notably faster in simulating neural models, NetPyNE is slightly less computationally efficient than Brian2.

Both simulators produced voltage traces consistent with that of the Hodgkin-Huxley model, as can be seen in figures 9.7-9.9 and 9.16-9.18 in Appendix 2. This suggests that both simulators are capable of producing accurate neuronal simulations, and that differences in simulation efficiency cannot be attributed to any differences in

complexity in how the Hodgkin-Huxley neuron is simulated in either simulator.

5. Limitations and Challenges

While Brian2 and NetPyNE offer powerful tools for neural modeling and simulation, there are certain limitations and challenges that should be considered:

Both simulators have a learning curve, especially for researchers who are new to computational neuroscience or simulation frameworks. Understanding the underlying concepts, syntax, and workflows may require time and effort to become proficient in utilizing the simulators effectively. Additionally, both Brian2 and NetPyNE require substantial computational resources, including memory and processing power, for simulations with a large number of neurons and synapses. Researchers should ensure they have access to adequate hardware resources to handle the computational requirements of their simulations.

In terms of documentation and support, while both simulators provide documentation and resources to assist users, the availability and comprehensiveness of documentation can vary. Researchers may encounter challenges in finding specific information or troubleshooting issues. Engaging with the respective user communities, forums, or seeking assistance from experienced users can help overcome these challenges. Although both Brian2 and NetPyNE support interoperability standards like NeuroML or PyNN, transitioning models or data between simulators may require additional effort and compatibility checks.

6. Conclusion

Brian2 stands out for its user-friendly interface, simplicity, and ease of use. It provides a wide range of built-in neuron models and offers flexibility in defining connectivity and generating input stimuli. With its object-oriented approach and support for sparse connectivity, Brian2 optimizes memory usage, making it efficient for simulations of smaller to medium-sized networks. Additionally, its integration with popular scientific computing libraries facilitates data analysis and visualization, streamlining the research workflow.

On the other hand, NetPyNE leverages the capabilities of NEURON, offering a more detailed and biophysically realistic modeling approach. It provides advanced connectivity options, allowing for the incorporation of experimental connectivity data and the modeling of large-scale networks with complex topologies. NetPyNE's integration with NEURON's parallelization capabilities enables efficient parallel simulation execution, making it suitable for simulations with a large number of neurons and synapses.

In addition, Brian2 diverges significantly from the architecture upon which many other simulators such as NetPyNE or NeuroML are built, which makes importing models between these tools difficult. Some support tools have been developed to bridge this gap, but most are not capable of creating

a 1:1 reconstruction of a model built using another tool in Brian2.

Ultimately, the choice between Brian2 and NetPyNE depends on the specific modeling requirements and priorities of the researcher. If simplicity, ease of use, and a more user-friendly interface are preferred, Brian2 is a compelling choice. On the other hand, if detailed biophysical modeling, advanced connectivity options, and scalability for large-scale networks are crucial, NetPyNE, with its integration with NEURON, provides a more comprehensive solution.

7. References

[1] Brian Authors, “Brian 2 documentation,” Brian 2 documentation - Brian 2 2.5.1 documentation, https://brian2.readthedocs.io/en/stable/ (accessed Jun. 16, 2023).

[2] NetPyNE Team, “Welcome to the Netpyne homepage!,” Welcome to the NetPyNE homepage! - NetPyNE documentation, http://www.netpyne.org/ (accessed Jun. 16, 2023).

[3] “NetPyNE GUI Tutorial.” Yale University, New Haven, 2018

[4] NeuroML Contributors, “PyNeuroML,” pyNeuroML - NeuroML Documentation, https://docs.neuroml.org/Userdocs/Software/pyNeuroML.html (accessed Jun. 16, 2023).

8. Appendix 1 - Notes on Applications of SPICE for Simulating Neural Activity

SPICE (Simulation Program with Integrated Circuit Emphasis) is a widely used circuit simulator originally designed for electronic circuit simulation, with potential applications in computational neuroscience. Although outside the scope of this project, its potential use in this field merits further interest.

At its core, SPICE excels in the design of custom circuit components. This feature is particularly critical for neural simulations, as it allows for the creation of models that accurately reflect the electrical behaviors of neurons and synapses. Utilizing SPICE's capabilities, researchers can develop models that incorporate nuanced details of neuronal activities, such as the kinetics of ion channels, the temporal dynamics of synaptic transmission, and the mechanisms underlying neural plasticity. These models can simulate the action potential generation, synaptic potentials, and other electrophysiological properties of neurons with a high degree of precision. Integrated waveform viewers and frequency domain analysis tools within SPICE also have the potential to be adapted for analyzing neuronal circuit behavior. Enhancements through third-party integrations, such as GUIs like NGSPICE or PSPICE, scripting capabilities, and data export options, significantly enhance SPICE’s usability and flexibility.

Despite these capabilities, scaling SPICE for simulations of extensive neural networks, such as those necessary to model brain-scale activities, poses significant computational challenges. The main issue lies in the exponential growth of computational demand correlating with the increase in network size, leading to severe strain on

processing power and memory resources. This is particularly evident in simulations that encompass networks ranging from thousands to millions of neurons. Additionally, SPICE's architecture, originally tailored for electronic circuits, is not optimized for processing the vast and complex datasets typical of large-scale biological simulations. This results in decreased simulation speeds, potential data bottlenecks, and challenges in maintaining data integrity during simulations.

To render SPICE a viable tool for the Fruit Fly Brain Observatory, significant enhancements are required, particularly in its data processing and simulation capabilities. There is a need to equip SPICE to efficiently handle the large volumes of data generated in neural simulations. This demands improvements in its data processing algorithms and memory management techniques. Moreover, the simulation algorithms within SPICE require optimization to accurately capture the dynamic interactions within large neural networks, including the intricate patterns of connectivity and the diverse range of neuronal firing properties.

9. Appendix 2 - Supplemental Figures

These figures are intended to showcase examples of the graphs generated during performance testing. These figures are meant to demonstrate that

All figures were generated during test execution as part of evaluating each simulator's performance metrics, via *run_simulations.ipnyb* in the attached github project, via each simulator's associated *simulate* function.

The neurons recorded (Mi4-C, C2-C, and L2-C) were chosen due to their presence in all test cases and their proximity to the point of stimulation. This was done to ensure that connectivity between neurons was correctly represented.

9.1 Network Diagrams

The following network diagrams visualize the connections between neurons for each test simulating, ranging from a single drosophila visual column to 6 columns. The spatial relationships between neurons shown here are not representative of the actual orientation between neurons obtained from FBL.

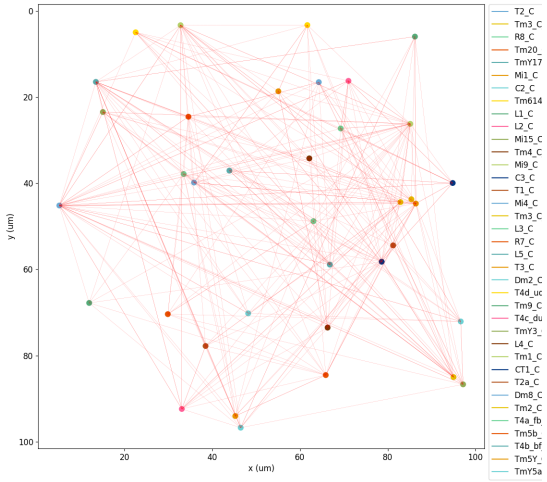

Figure 9: Network Diagram, 1 Column

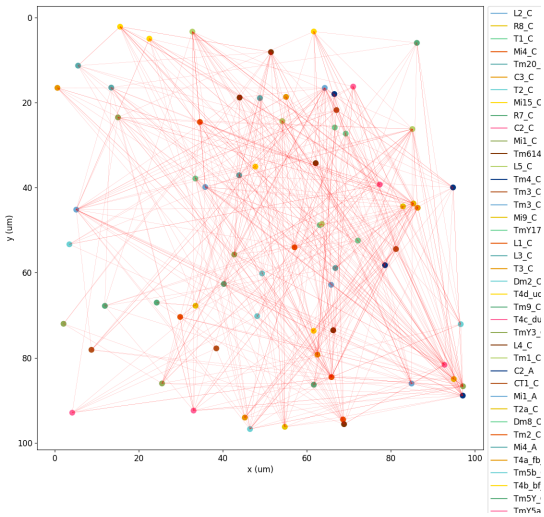

Figure 10: Network Diagram, 2 Columns

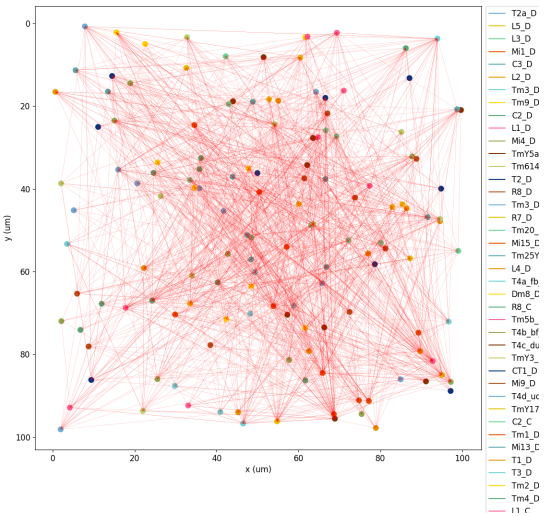

Figure 12: Network Diagram, 4 Columns

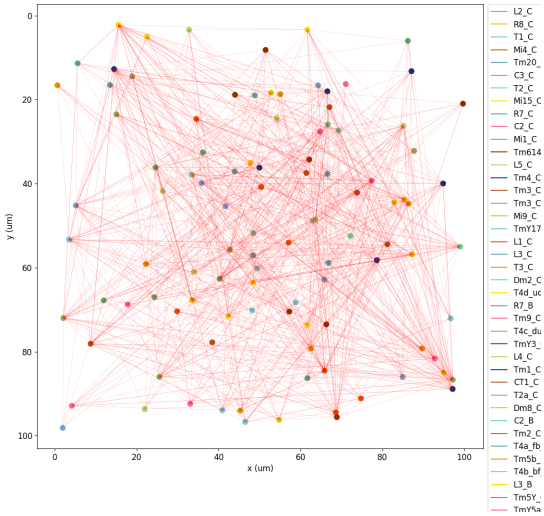

Figure 11: Network Diagram, 3 Columns

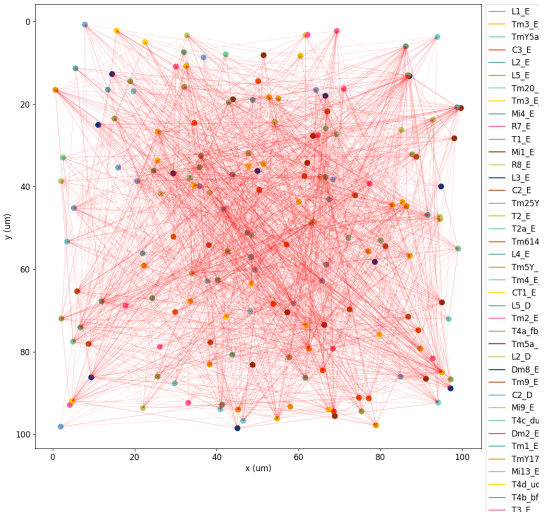

Figure 13: Network Diagram, 5 Columns

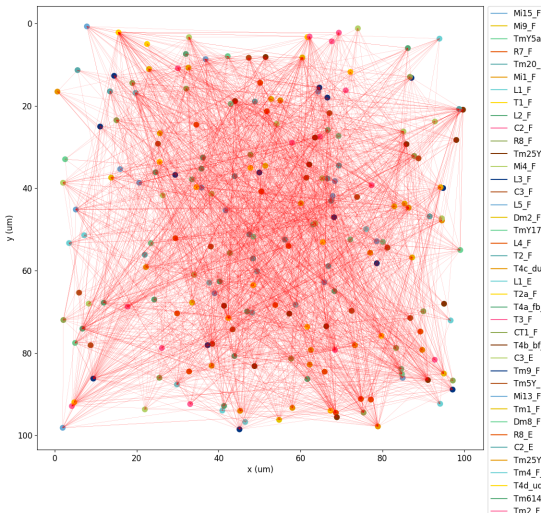

Figure 14: Network Diagram, 6 Columns

9.2 Brian2 Spike Data

The following graphs are a sample of the spike recordings obtained while evaluating Brian2's performance metrics on a single visual column. Spike recordings for all test cases are available in *run_simulations.ipnyb*.

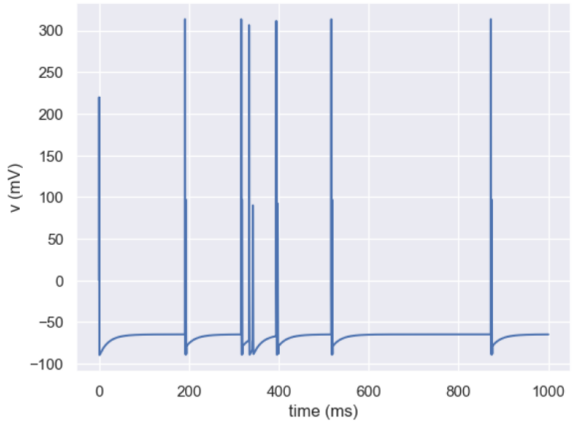

Figure 15: Mi4-C Spike Recording

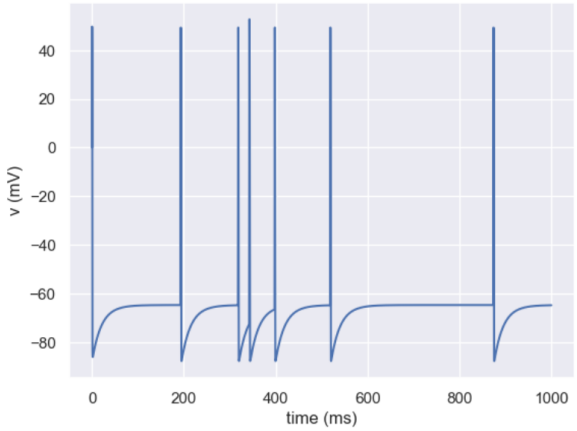

Figure 16: C2-C Spike Recording

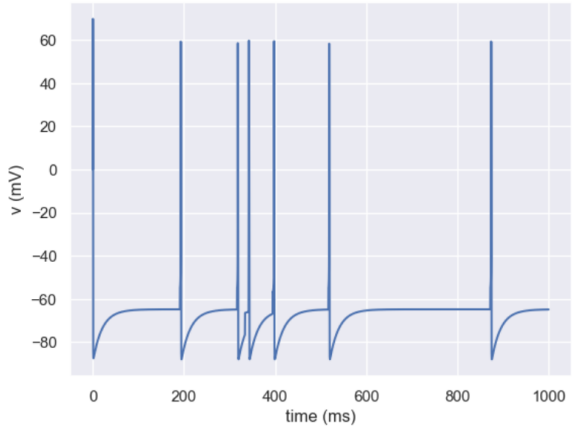

Figure 17: L2-C Spike Recording

9.3 NetPyNE Spike Data, Raster Plots

The following graphs are raster plots for neuronal firing patterns across all test cases. These images are also available in the attached github project's img directory.

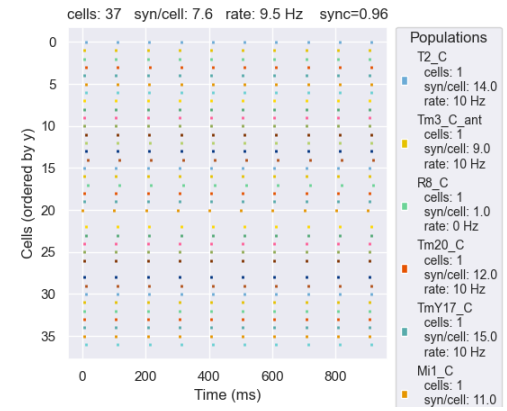

Figure 18: Raster Plot, 1 Column

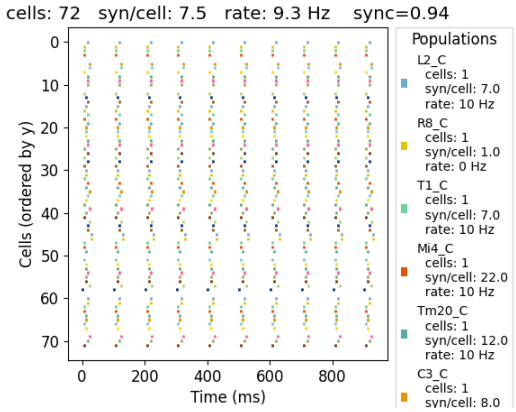

Figure 19: Raster Plot, 2 Columns

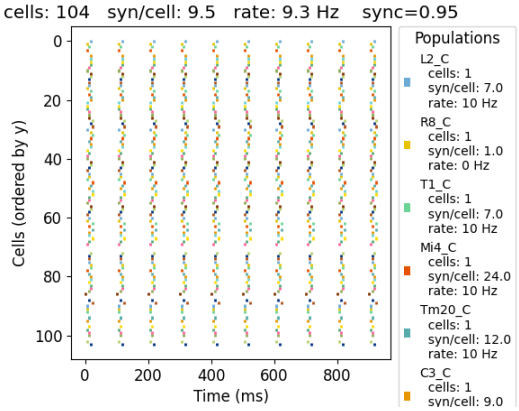

Figure 20: Raster Plot, 3 Columns

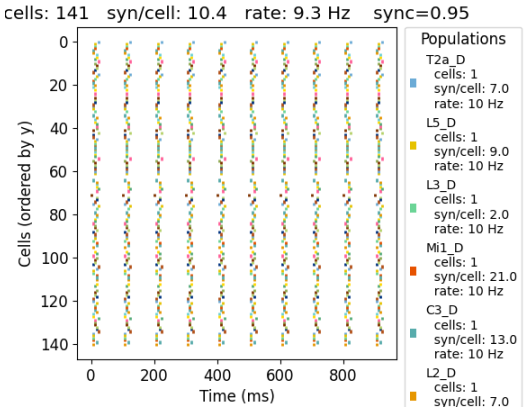

Figure 21: Raster Plot, 4 Columns

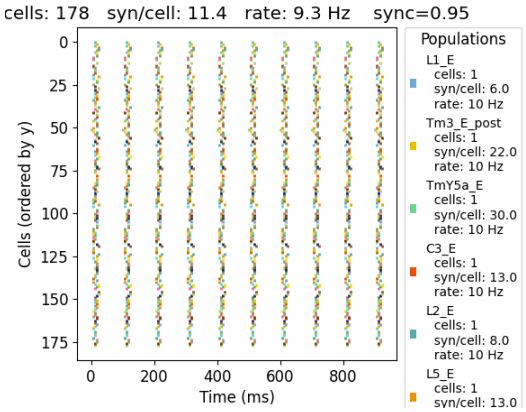

Figure 22: Raster Plot, 5 Columns

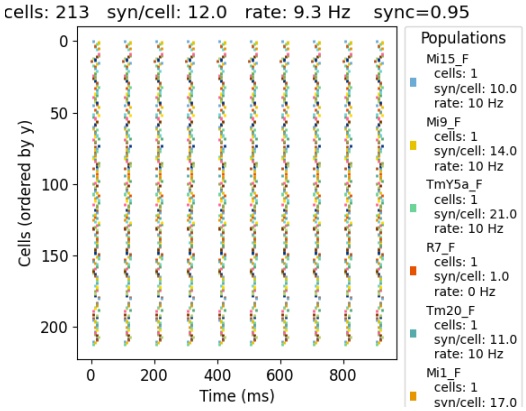

Figure 23: Raster Plot, 6 Columns

9.4 NetPyNE Spike Data, Spike Recordings

The following graphs are a sample of the spike recordings obtained while evaluating

NetPyNE's performance metrics on a single visual column. Spike recordings for all test cases are available in the attached github project's img directory.

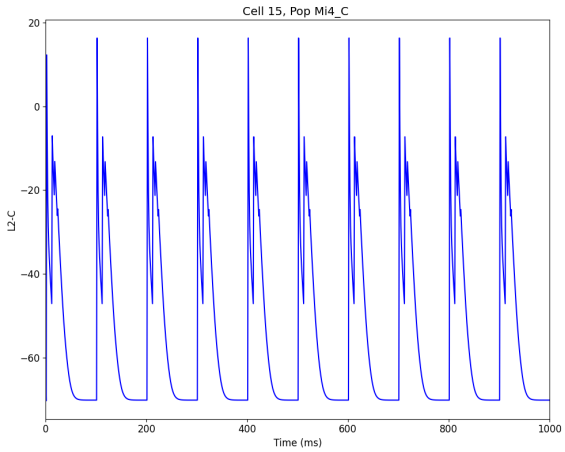

Figure 24: Mi4-C Spike Recording

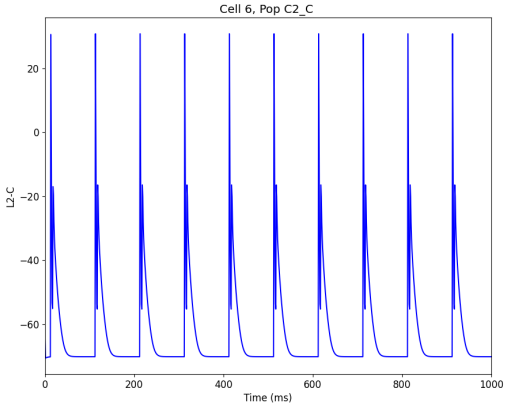

Figure 25: C2-C Spike Recording

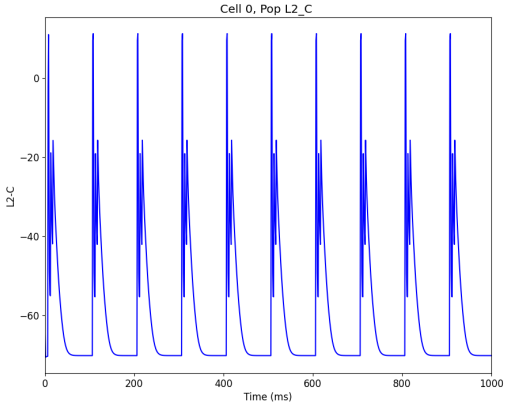

Figure 26: L2-C Spike Recording

10. Appendix 3 - Technical Details

10.1 System Specifications

The device used to compile code had the following specifications:

- Processor: AMD Ryzen 9 4900HS
- CPU Speed: 3000 MHz
- Cores: 8
- Logical Processors: 16
- RAM: 16 GB
- Virtual Memory: 32 GB

10.2 List of Packages Used

- brian2
- brian2tools
- cython
- flybrainlab
- graphviz
- gsl
- install-jdk
- lxml
- netpyne
- pyneuroml