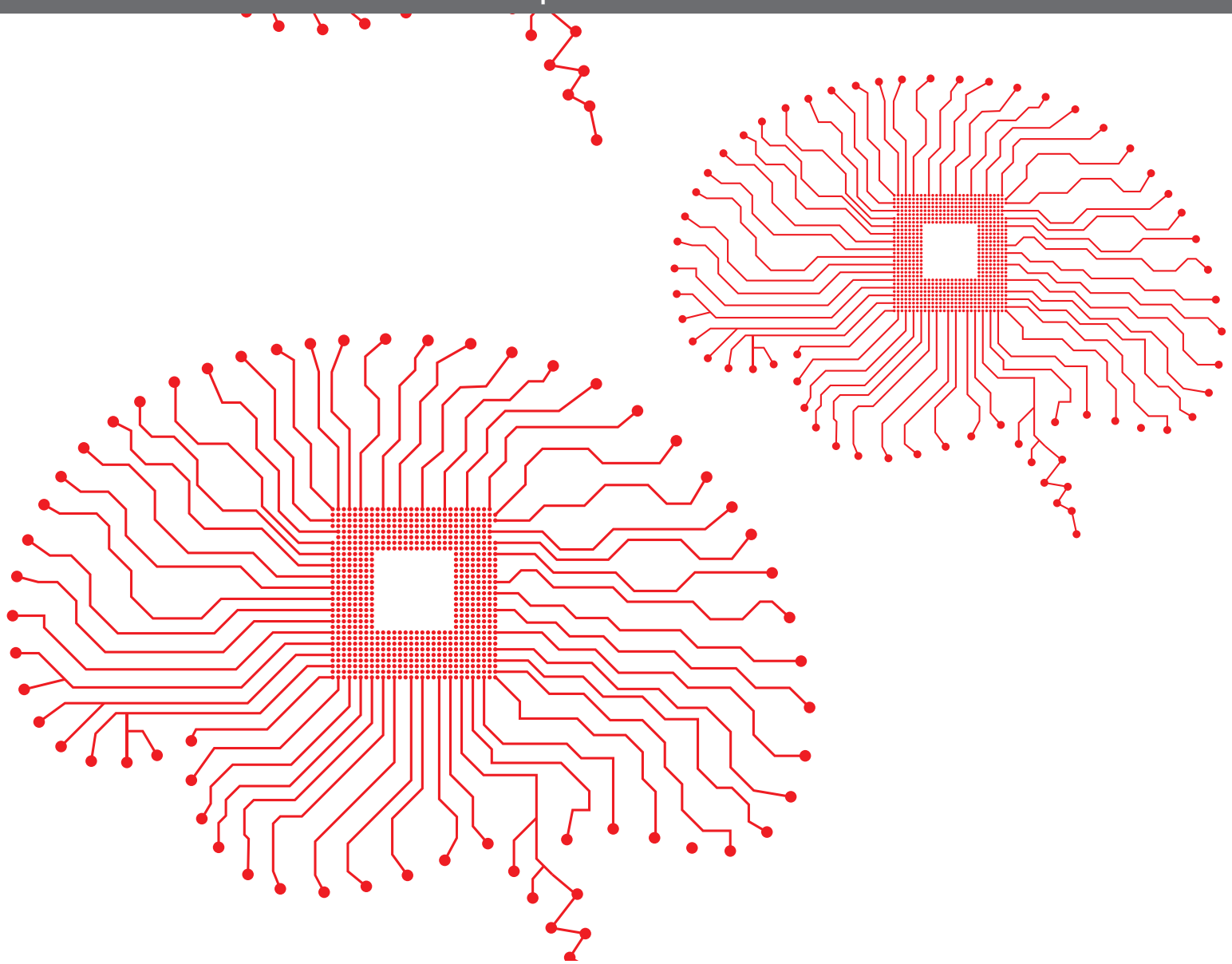
NEUROINFORMATICS OF LARGE SCALE BRAIN MODELLING

EDITED BY: John David Griffiths, Kelly Shen and Padraig Gleeson
PUBLISHED IN: Frontiers in Neuroinformatics and
Frontiers in Computational Neuroscience



frontiers Research Topics

frontiers

ISSN 1664-8714
ISBN 978-2-83250-179-5
DOI 10.3389/978-2-83250-179-5

About Frontiers

Frontiers is more than just an open-access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers Journal Series

The Frontiers Journal Series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the Frontiers Journal Series operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to Quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews.
Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the Frontiers Journals Series: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area! Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers Editorial Office: frontiersin.org/about/contact

NEUROINFORMATICS OF LARGE SCALE BRAIN MODELLING

Topic Editors:
John David Griffiths, University of Toronto, Canada
Kelly Shen, Simon Fraser University, Canada
Padraig Gleeson, University College London, United Kingdom

Citation: Griffiths, J. D., Shen, K., Gleeson, P., eds. (2022). Neuroinformatics of Large Scale Brain Modelling. Lausanne: Frontiers Media SA.
doi: 10.3389/978-2-83250-179-5

Table of Contents

frontiers | Frontiers in Neuroinformatics

TYPE Editorial
PUBLISHED 11 October 2022
DOI 10.3389/fninf.2022.1043732

OPEN ACCESS

EDITED AND REVIEWED BY
Jan G. Bjaalie,
University of Oslo, Norway

*CORRESPONDENCE
John D. Griffiths
j.davidgriffiths@gmail.com

RECEIVED 14 September 2022
ACCEPTED 26 September 2022
PUBLISHED 11 October 2022

CITATION
Griffiths JD, Shen K and Gleeson P (2022) Editorial: Neuroinformatics of large-scale brain modelling. *Front. Neuroinform.* 16:1043732. doi: 10.3389/fninf.2022.1043732

Editorial: Neuroinformatics of large-scale brain modelling

John D. Griffiths[1,2,3]*, Kelly Shen[4] and Padraig Gleeson[5]

[1]Krembil Centre for Neuroinformatics, Centre for Addiction and Mental Health, Toronto, ON, Canada, [2]Department of Psychiatry, University of Toronto, Toronto, ON, Canada, [3]Institute of Medical Sciences, University of Toronto, Toronto, ON, Canada, [4]Institute for Neuroscience and Neurotechnology, Simon Fraser University, Burnaby, BC, Canada, [5]Department of Neuroscience, Physiology, and Pharmacology, University College London, London, United Kingdom

KEYWORDS

large-scale brain modeling, computational neuroscience, neuroinformatics, neuroimaging, simulators and models

Editorial on the Research Topic
Neuroinformatics of large scale brain modelling

A major focus in contemporary neuroscience research is the mapping and modeling of connectivity and activity dynamics in large-scale brain networks. As the resolution, coverage, and availability of neural data increase rapidly, neuroinformatics techniques are playing an increasingly important role in this scientific enterprise. *Large-scale brain modeling* is the methodologically-defined sub-field of computational neuroscience that is focused on simulations of either whole-brain activity at a coarse-grained (meso/macro) spatial scale, or activity in select neural subsystems at a fine-grained (micro) spatial scale and high level of detail. Neuroinformatics tools employed in large-scale brain modeling come in the form of software infrastructure, database resources, and practical implementations of mathematical and algorithmic techniques that facilitate these core research goals.

In many cases the neuroinformatics and architectural solutions developed as part of this work are in themselves of general methodological interest to researchers, but are often communicated secondarily to the principal neuroscientific research questions. This joint Frontiers in Neuroinformatics and Frontiers in Computational Neuroscience Research Topic was therefore conceived by the Editorial Team as a venue to highlight exciting recent developments in the field, as well as to demonstrate the broad range of innovative work taking place. It features a collection of 11 original research articles describing new advances in the neuroinformatics of large-scale brain modeling. These span a diverse range of computational methods and neuroscientific applications, from cell and microcircuit dynamics to macro-scale neuroanatomy and neuroimaging. In addition to the stand-alone value of the various individual contributions, we believe strongly that the shared focus on computational methodologies across the articles in this collection brings an important additional benefit—to facilitate dialogue, exposure, and cross-pollination across neuroscience sub-fields.

Two common themes across the included articles are (i) improving the scale, speed, accuracy, and resolution of modeling and data analysis pipelines, and (ii) improving connections between micro-meso-macro levels of analysis. We will discuss contributions from each of these themes in turn. Several of the featured papers describe new or improved simulator software. One example is the article by Panagiotou et al., which introduces a novel and high-performance neural simulator named EDEN ("Extensible Dynamics Engine for Networks"). By heavily basing EDEN around the model specification language NeuroML (Gleeson et al., 2010), EDEN achieves an impressive combination of flexibility and ease-of-use. More impressively however, the authors also demonstrate almost two orders-of-magnitude speed improvements as compared to the industry-standard tool NEURON (Hines and Carnevale, 1997) for simple single-computer usage, as well as seamless scaling over multiple CPUs and compute clusters with minimal effort and code modification. Improving computational scalability is also a major emphasis in the contributions of Florimbi et al. and Golosio et al., who describe impressive performance with new GPU-based architectures for modeling large-scale cerebellar networks (incorporating conductance-based neuronal models) and spiking network models, respectively. A common context where high-performance implementations are particularly needed and useful is in parameter space exploration and parameter optimization problems. On this topic, Yegenoglu et al. present a novel genetic algorithm-based approach, drawing on the concept of "learning to learn" (L2L), with worked examples for multiple simulators at multiple scales.

Complementing these contributions focused on the specification and execution of neural simulations, several articles in this collection address another major topic in the field of neuroinformatics, namely the systematic and efficient analysis of multimodal structural and functional brain data, at various spatial scales, as a critical first step in the development of large-scale computational models of brain activity. Moon et al. offer suggestions for improving the scalability of tractography analyses, now commonly used for reconstructing white matter fiber projections and anatomical connectivity patterns from individual human diffusion-weighted MRI scans. Their article is based around the interesting observation that probabilistic tractography reconstructions, which conventionally make use of thousands or millions of samples per seed location, can be reduced down to a handful of samples with comparable results in terms of identifiability and connectome matrix quality. Building on tools such as these, Frazier-Logue et al. describe a new improved neuroimaging pipeline for preprocessing brain connectivity using structural and functional MRI scans in preparation for whole-brain simulations with TheVirtualBrain library (https://thevirtualbrain.org; Sanz Leon et al., 2013).

Moving another level up, two articles in this collection explore the question of data organization *via* systematic ontologies. Gutierrez et al. describe a new tool for collaborative data-driven development of spiking network models, including structured management of the various entities used to specify physiological parameters and state variable dynamics, as well as code generation functionality that allows full specification of NEST-based network models (Gewaltig and Diesmann, 2007). At a slightly higher level of abstraction, Lazar et al. offer a novel "programmable logic" schema for describing the functional organization of the *Drosophila* brain, including a web application ("NeuroNLP++") allowing natural language querying of published literature. These authors demonstrate usage of their new tool with examples exploring the functional logic of feedback loops in the *Drosophila* antennal lobe.

The second main theme in this Research Topic is, as noted, improving connections between micro, meso, and macro levels of analysis, which is represented by the articles from Siu et al., Layer et al., and Maith et al.. In the first of these, Maith et al. introduce a blood oxygenation-dependent (BOLD) "monitor" (i.e., empirical measurement process simulator) into the ANNarchy (Vitay et al., 2015) spiking neuron modeling library. This work can be seen as part of the major recent and growing trend in computational neuroscience toward "true" multiscale neural models that simultaneously capture experimentally observed patterns across several qualitatively different data types (D'Angelo and Jirsa, 2022). The contribution by Siu et al. also has a focus on modeling of hemodynamic activity patterns, in this case resting-state and stimulus-evoked fMRI activity patterns in anesthetized mice. Impressively, these authors combine theoretical analyses of bifurcation behavior in neural mass models of mouse cortex with neuroinformatics database-driven spatial variations in dynamical parameters to study resting-state and stimulus-driven activity patterns in (mouse) whole-brain functional MRI data. Finally, the paper by Layer et al. articulates a general approach for mean-field model derivation/reduction that is both theoretically powerful and practically useful, the latter in particular due to their development of the new open-source "Neuronal Network Mean-Field Toolbox" (NNMT) Python library (github.com/INM-6/nnmt). A particular strength of NNMT is its treatment of mean-field behavior for leaky integrate and fire model neuron models. The toolbox also provides functionality to estimate various properties of large neuronal networks, such as firing rates, power spectra, and dynamical stability in mean-field and linear response approximations, based entirely on well-developed mathematical theory and without the need for running computationally expensive numerical simulations.

We thank the authors for their excellent contributions to this Research Topic on the Neuroinformatics of Large-Scale Brain

Modeling. Our hope and aim is that the collection assembled here provides an interesting and representative overview of the impressive recent work on this topic from around the world, and we look forward to continued discussions on new developments in this exciting field.

Author contributions

All authors contributed equally to the editing of the Research Topic and conceptualization of the manuscript. JG wrote the first draft of the manuscript and all authors contributed to revisions and approved the submitted version.

Funding

JG acknowledges research funding support from the Krembil Foundation, CAMH Discovery Fund, Labbatt Foundation, and Tri-Council UK-Canada AI Initiative. PG acknowledges research funding support from Wellcome (212941).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

D'Angelo, E., and Jirsa, V. (2022). The quest for multiscale brain modeling. *Trends Neurosci*. 45, P777–790. doi: 10.1016/j.tins.2022.06.007

Gewaltig, M.-O., and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia J*. 2, 1430. doi: 10.4249/scholarpedia.1430

Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., et al. (2010). NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput. Biol.* 6, e1000815. doi: 10.1371/journal.pcbi.1000815

Hines, M. L., and Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Comput*. 9, 1179–1209. doi: 10.1162/neco.1997.9.6.1179

Sanz Leon, P., Knock, S. A., Woodman, M. M., Domide, L., Mersmann, J., McIntosh, A. R., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinform*. 7, 10. doi: 10.3389/fninf.2013.00010

Vitay, J., Dinkelbach, H. Ü., and Hamker, F. H. (2015). ANNarchy: a code generation approach to neural simulations on parallel hardware. *Front. Neuroinform*. 9, 19. doi: 10.3389/fninf.2015.00019

frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 17 February 2021
doi: 10.3389/fncom.2021.627620

Fast Simulations of Highly-Connected Spiking Cortical Models Using GPUs

Bruno Golosio[1,2]*, *Gianmarco Tiddia*[1,2], *Chiara De Luca*[3,4], *Elena Pastorelli*[3,4], *Francesco Simula*[4] and *Pier Stanislao Paolucci*[4]

[1] Department of Physics, University of Cagliari, Cagliari, Italy, [2] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Cagliari, Cagliari, Italy, [3] Ph.D. Program in Behavioral Neuroscience, "Sapienza" University of Rome, Rome, Italy, [4] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Roma, Rome, Italy

Over the past decade there has been a growing interest in the development of parallel hardware systems for simulating large-scale networks of spiking neurons. Compared to other highly-parallel systems, GPU-accelerated solutions have the advantage of a relatively low cost and a great versatility, thanks also to the possibility of using the CUDA-C/C++ programming languages. NeuronGPU is a GPU library for large-scale simulations of spiking neural network models, written in the C++ and CUDA-C++ programming languages, based on a novel spike-delivery algorithm. This library includes simple LIF (leaky-integrate-and-fire) neuron models as well as several multisynapse AdEx (adaptive-exponential-integrate-and-fire) neuron models with current or conductance based synapses, different types of spike generators, tools for recording spikes, state variables and parameters, and it supports user-definable models. The numerical solution of the differential equations of the dynamics of the AdEx models is performed through a parallel implementation, written in CUDA-C++, of the fifth-order Runge-Kutta method with adaptive step-size control. In this work we evaluate the performance of this library on the simulation of a cortical microcircuit model, based on LIF neurons and current-based synapses, and on balanced networks of excitatory and inhibitory neurons, using AdEx or Izhikevich neuron models and conductance-based or current-based synapses. On these models, we will show that the proposed library achieves state-of-the-art performance in terms of simulation time per second of biological activity. In particular, using a single NVIDIA GeForce RTX 2080 Ti GPU board, the full-scale cortical-microcircuit model, which includes about 77,000 neurons and $3 \cdot 10^8$ connections, can be simulated at a speed very close to real time, while the simulation time of a balanced network of 1,000,000 AdEx neurons with 1,000 connections per neuron was about 70 s per second of biological activity.

Keywords: spiking neural network simulator, cortical microcircuits, adaptive exponential integrate-and-fire neuron model, conductance-based synapses, GPU

OPEN ACCESS

Edited by:
Padraig Gleeson,
University College London,
United Kingdom

Reviewed by:
Jiang Wang,
Tianjin University, China
Marcel Stimberg,
Université de la Sorbonne, France

***Correspondence:**
Bruno Golosio
golosio@unica.it

Received: *09 November 2020*
Accepted: *26 January 2021*
Published: *17 February 2021*

Citation:
Golosio B, Tiddia G, De Luca C, Pastorelli E, Simula F and Paolucci PS (2021) Fast Simulations of Highly-Connected Spiking Cortical Models Using GPUs. Front. Comput. Neurosci. 15:627620. doi: 10.3389/fncom.2021.627620

1. INTRODUCTION

The human brain is an extremely complex system, with a number of neurons in the order of 100 billions, an average number of connections per neuron in the order of 10 thousands, hundreds of different neuron types, several types of neurotransmitters and receptors. Because of this complexity, the simulation of brain activity at the level of signals produced by individual neurons is extremely demanding, even if it is limited to relatively small regions of the brain. Therefore, there is a growing interest in the development of high-performance hardware and software tools for efficient simulations of large-scale networks of spiking neuron models. Some simulators, as for instance NEST (Fardet et al., 2020), NEURON (Carnevale and Hines, 2006), and Brian (Goodman and Brette, 2008), combine flexibility and simplicity of use with the possibility to simulate a wide range of spiking neuron and synaptic models. All three of these simulators offer support for multithread parallel computation for parallelization on a single computer. NEST and NEURON also support distributed simulations on computer clusters through MPI. On the other hand, a fertile field of research in recent decades has investigated the use of highly parallel hardware systems for simulating large-scale networks of spiking neurons. Such systems include custom made neuromorphic very-large-scale-integration (VLSI) circuits (Indiveri et al., 2011), field programmable gate arrays (FPGAs) (Wang et al., 2018), and systems based on graphical processing units (GPUs) (Sanders and Kandrot, 2010; Garrido et al., 2011; Brette and Goodman, 2012; Vitay et al., 2015; Yavuz et al., 2016; Chou et al., 2018). Compared to other highly-parallel systems, the latter have the advantages of a relatively low cost, a sustained technological development driven by the consumer market and a great versatility, thanks also to the possibility of using CUDA (Compute Unified Device Architecture), a parallel computing platform and programming model that has been created by NVIDIA to allow software developers to take full advantage of the GPU capabilities (Sanders and Kandrot, 2010). General purpose computing on graphical processing units (GPGPU) is widely employed for massively parallel computing. GPGPUs can significantly reduce the processing time compared to multi-core CPU systems for tasks that require a high degree of parallelism, because a single GPU can perform thousands of core computations in parallel. However, in order to derive maximum benefit from GPGPU, the applications must be carefully designed taking into account the hardware architecture. Over the past decade, several GPU-based spiking neural network simulators have been developed (see Brette and Goodman, 2012 for a review). EDLUT (Garrido et al., 2011) is a hybrid CPU/GPU spiking neural network simulator which combines time-driven (in GPU) and event-driven (in CPU) simulation methods to achieve real-time simulation of medium-size networks, which can be exploited in real-time experiments as for instance the control of a robotic arm. ANNarchy (Vitay et al., 2015) is a simulator for distributed rate-coded or spiking neural networks, which provides a Python interface for the definition of the networks and generates optimized C++ code to actually run the simulation in parallel, using either OpenMP on CPU architectures or CUDA on GPUs. CARLsim (Chou et al., 2018) is a GPU-accelerated library for simulating large-scale spiking neural network (SNN), which includes different neuron models and provides programming interfaces in C/C++ and in Python. Recently, the GeNN simulator (Yavuz et al., 2016; Knight and Nowotny, 2018) achieved cutting edge performance in GPU-based simulation of spiking neural networks, achieving better performance than CPU-based clusters and neuromorphic systems in the simulation of the full-scale cortical microcircuit model proposed by Potjans and Diesmann (2014). In this work we present a comprehensive GPU library for fast simulation of large-scale networks of spiking neurons, called NeuronGPU, which uses a novel GPU-optimized algorithm for spike delivery. This library can be used either in Python or in C/C++. The Python interface is very similar to that of the NEST simulator and allows interactive use of the library. Having an interface similar to that of NEST is an advantage in view of a possible integration of this library with the NEST simulator, which is currently in progress (Golosio et al., 2020). In the following sections, after a general description of the library and of the spike-delivery algorithm, we will evaluate the library on three types of spiking neural network models:

- The Potjans-Diesmann cortical microcircuit model (Potjans and Diesmann, 2014), based on the leaky-integrate-and-fire (LIF) neuron model, which describes the behavior of a region of the cerebral cortex having a surface of 1 mm^2 and includes about 77,000 neurons and $3 \cdot 10^8$ connections;
- A balanced network of excitatory and inhibitory neurons (Brunel, 2000), based on the adaptive-exponential-integrate-and-fire (AdEx) neuron model (Brette and Gerstner, 2005), with up to 1,000,000 neurons and 10^9 connections;
- A balanced network of excitatory and inhibitory neurons, based on the Izhikevich neuron model (Izhikevich, 2003) and STDP synapses, with up to 1,000,000 neurons and 10^8 connections.

We will show that, although the building time is larger compared to other simulators, NeuronGPU achieves state-of-the-art performance in terms of simulation time per unit time of biological activity.

2. MATERIALS AND METHODS

2.1. The NeuronGPU Library

NeuronGPU is a GPU library for simulation of large-scale networks of spiking neurons, written in the C++ and CUDA-C++ programming languages. Currently it can simulate LIF models, different multisynapse AdEx models with current or conductance based synapses as well as user definable neuron models. The LIF model subthreshold dynamics is integrated by the *exact integration* scheme described in Rotter and Diesmann (1999) on the time grid given by the simulation time resolution. On the other hand, the numerical solution of the differential equations of the AdEx dynamics is performed through a parallel implementation, written in CUDA C++, of the fifth-order Runge-Kutta method with adaptive control of the step size (Press and Teukolsky, 1992). NeuronGPU can simulate networks of any

neuron and synaptic current models whose dynamics can be described by a system of ordinary differential equations (ODEs), although currently it does not provide a dedicated interface for defining new models; the definition of a new model involves changes in specific parts of the code. However, such changes do not require experience with programming languages. In the simplest approach, the user has to modify the list of state variables and parameters, their initial values, and the differential equations that describe the neuron dynamics. With this approach the number of user-defined neuron models that can be used in a simulation together with the pre-defined models is limited to two. A more advanced approach allows to use an arbitrary number of new models in the same simulation and greater flexibility in the model definition. Detailed instructions on different approaches for the implementation of new models can be found in https://github.com/golosio/NeuronGPU/wiki/How-to-implement-new-neuron-models. The computations are carried out using mainly 32-bit floating point numerical precision, with the exception of some parts of the code for which double precision calculations are more appropriate, e.g., those in which a very large number of terms can be added. Neuron parameters and connection weights and delays can be initialized either using fixed values or through arrays or probability distributions. Neuron groups can be connected either using predefined connection rules (one-to-one, all-to-all, fixed indegree, fixed outdegree, fixed total number) or by user-defined connections. In addition to the standard synapse model, nearest-neighbor spike-timing-dependent-plasticity (STDP) is also available (Morrison et al., 2008; Sboev et al., 2016). In the STDP model, the weight that characterizes the strength of a synapse changes when the presynaptic and postsynaptic neurons emit spikes that are close in time. More specifically, the weight change depends on the time difference: $\Delta t = t_{\text{post}} - t_{\text{pre}} = t_{\text{spike\_post}} + \tau_{\text{dendritic}} - (t_{\text{spike\_pre}} + \tau_{\text{axon}})$ where $t_{\text{spike\_pre}}$ is the time the presynaptic neuron emits the spike, τ_{axon} is the axonal delay, t_{pre} is the time the presynaptic spike reaches the synapse, $t_{\text{spike\_post}}$ is the time the postsynaptic neuron emits the spike, $\tau_{\text{dendritic}}$ is the dendritic backpropagation delay, i.e., the time between the emission of the postsynaptic spike and the time in which it affects the synapse, t_{post} is the time in which the postsynaptic spike affects the synapse. NeuronGPU uses a symmetric-nearest-neighbor spike pairing scheme (Morrison et al., 2008). A weight change can be triggered either by the postsynaptic or by the presynaptic spike buffer. The first case occurs when the time associated with a spike stored in the postsynaptic spike buffer becomes equal to the dendritic delay. In this case Δt is equal to the difference between the current time and the time in which the last presynaptic spike reached the synapse. The second case occurs when the time associated with a spike stored in the presynaptic spike buffer becomes equal to the axonal delay. In this second case, Δt is equal to the difference between the time in which the last postsynaptic spike reached the input synapse and the current time. In both cases, the weight change is computed using the formula (Sboev et al., 2016):

$$\Delta w = \begin{cases} -\lambda \alpha w_{\max} \cdot \left(\frac{w}{w_{\max}}\right)^{\mu_-} \cdot e^{\left(\frac{\Delta t}{\tau_-}\right)} & \text{if } \Delta t = t_{\text{post}} - t_{\text{pre}} < 0 \\ \lambda w_{\max} \cdot \left(1 - \frac{w}{w_{\max}}\right)^{\mu_+} \cdot e^{\left(-\frac{\Delta t}{\tau_+}\right)} & \text{if } \Delta t = t_{\text{post}} - t_{\text{pre}} > 0 \end{cases} \quad (1)$$

If $\mu_+ = \mu_- = 0$, the rule is called additive, while if $\mu_+ = \mu_- = 1$ the rule is called multiplicative, and intermediate values are also possible. Different types of spike generators and recording devices can be simulated, including Poisson generators, spike recorders, and multimeters. NeuronGPU includes an efficient implementation of GPU-MPI communication among different nodes of a GPU cluster, however the performance of the proposed library on GPU clusters has not yet been thoroughly evaluated, therefore this feature is not described in the present work. The Python interface is very similar to that of NEST in main commands, use of dictionaries, connection rules, model names, and parameters. The following Python code sample illustrates this strong similarity.

```
import neurongpu as ngpu
# create Poisson generator with rate
poiss_rate
pg = ngpu.Create(''poisson_generator'')
poiss_rate = 12000. 0
ngpu. SetStatus(pg, ''rate,'' poiss_rate)
# Create n_neurons neurons with n_receptor
receptor ports
# neuron model is multisynapse AdEx (aeif)
with conductance-based synapse
# described by the beta function
n_neurons = 10
n_receptor = 2
neuron = ngpu.Create(''aeif_cond_beta,''
n_neurons, n_receptors)
# Initialize receptor parameters
E_rev = [0.0, -85.0]
tau_decay = [1.0, 1.0]
tau_rise = [1.0, 1.0]
ngpu.SetStatus(neuron,
{"E_rev":E_rev, "tau_decay":tau_decay,
"tau_rise":tau_rise})
# Connect Poisson generator to neurons
poiss_weight = 0.05
poiss_delay = 2.0
conn_dict={"rule": "all_to_all"}
syn_dict={"weight": poiss_weight, "delay":
poiss_delay, "receptor":0}
ngpu.Connect(poiss_gen, neuron, conn_dict,
syn_dict)
```

About 30 test scripts and C++ programs have been designed to check the correctness of neuron model dynamics, spike generators, recording tools, spike delivery, connection rules. Many of such tests use similar NEST simulations as reference. Several examples in C++ and in Python are also available. NeuronGPU is an open-source library, freely available on GitHub from the web address https://github.com/golosio/NeuronGPU under the terms of the GNU General Public License v3.0.

2.2. The Spike-Delivery Algorithm

A crucial issue that must be addressed in the design of spiking neural network simulators is the choice of the algorithms to store

the spikes and to propagate and deliver them after proper delays. In particular, two important aspects can significantly affect the performance of different approaches: the way they account for the delays associated with connections and the representation used to index connections and to retrieve them when they must be used for spike delivery. A common approach for handling delays consists in using a circular event queue (see for instance Brette et al., 2007). Each element of this queue corresponds to a time index, and points to a list of synaptic spikes that are scheduled for that time. When a neuron i fires a spike, for each target neuron j a synaptic event $i \rightarrow j$ is scheduled to be delivered at a time $t + d$, where d is the synaptic delay. The computational cost per time step of managing delays with this approach is (Brette et al., 2007)

$$c_d \times N \times F \times C \times dt \tag{2}$$

where c_d is the cost of one store and retrieve operation in the circular queue, N is the number of neurons or other spiking devices, F is the average firing rate, C is the number of output connections per neuron and dt it the simulation time step. The computational cost per time step for propagating the spikes is

$$c_p \times N \times F \times C \times dt \tag{3}$$

where c_p is the cost of one spike propagation. In CPU implementations of this approach, c_d is usually small compared to c_p, therefore handling delays through the circular queue increases the cost of spike propagation by a small factor. On the other hand, in a GPU implementation c_d may not be small compared to c_p, because the insertion and retrieval operations in the circular queue would require access to the GPU global memory. This type of access is relatively slow, and represents in many cases one of the main bottlenecks of GPU codes. For this reason, many GPU-based simulators use different methods. Nageswaran et al. (2009) propose an approach for handling spikes and synaptic delays in GPU architectures based on two tables: a firing count table and a firing address table. The firing count table stores the cumulative count of neurons that emitted a spike in each time step of the last second. The firing address table holds the indexes of the neurons that emitted a spike in the last second. The firing count table is used to retrieve from the firing address table the list of all the neurons that fired in each time step t', with $t - \texttt{max\_delay} \leq t' \leq t$, where t is the current time step, and `max_delay` is the maximum delay of all synaptic connections of the network, expressed in time step units. The computational cost per time step for retrieving the spikes emitted in that interval is $O(N \times F \times \texttt{max\_delay} \times dt)$. The spikes emitted in the time step t' are sent to the neurons' outgoing synaptic connections having a delay equal to $t - t'$. Synaptic connections are represented through a sparse representation similar to adjacency lists for directed graphs. Each neuron has a list of output connections, identified by the index of the target neuron and by the index of the synapse in that neuron. The connections in the list are sorted based on their delays. Two arrays, `delay start` and `delay count`, are used to retrieve the connections corresponding to a given delay: `delay start`$[k]$ is the index of the first connection in the list with a delay of k ms, and `delay count`$[k]$ is the number of connections having that delay. A drawback of this approach is that spikes produced by neurons that have outgoing connections with a maximum delay much less than `max_delay` remain in the firing address table and are retrieved for a number of time steps equal to `max_delay`.

Yavuz et al. (2016) propose an algorithm for handling spikes and synaptic delays based on a circular queue array structure, with $N \times m$ elements, where $m = \text{delay}/dt$. An index p points to the slots of the queue, and is increased by 1 at every time step. A spike of the ith neuron is stored in the slot (i, p) of the queue, and spikes to be delivered are retrieved from the slots $[i, (p - m) \bmod m]$. This approach is very efficient, with a computational cost $O(N)$, however it has the limitation that delays have to be identical across the synapses of each synapse population. In order to use different delays, a synapse population has to be defined for each delay, with its own circular queue structure. In particular, this approach would not be efficient in realistic conditions where the delays vary according to some probability distribution. The spikes retrieved from the queue are delivered to the target neurons through a connection matrix, either an all-to-all connection matrix in case of dense connections, or based on the YALE sparse matrix format (Eisenstat et al., 1982) in case of sparse connectivity.

NeuronGPU uses one (output) spike buffer per neuron, which holds the spikes that have been fired by the neuron. The output connections of each neuron are organized in groups, all connections in the same group having the same delay (see **Figure 1**). Only three values per spike are stored in the buffer: a multiplicity, a time index t_s, which starts from 0 and is incremented by 1 at every time step, and a connection-group index i_g, which also starts from zero and is incremented by 1 every time the spike reaches a connection group, i.e., when the time index t_s matches the connection-group delay. **Figure 1A** represents the structure of the spike buffer and illustrates an example of how the spike is delivered from the neuron that fired it to the target neurons of different connection groups. Keeping a connection-group index and having output-connection groups ordered according to their delays is useful for reducing the computational cost, because with this approach there is no need for a nested loop for comparing the time index of the spike with the connection delays. When the time index of a spike t_s matches a connection-group delay, the spike is sent to the spike array, as shown in **Figure 1B**. Finally, spikes are sent from this array to the target neurons. This final delivery is done directly by a CUDA kernel, so no additional memory is required. The maximum size of the global spike array is equal to the number of nodes (i.e., neurons and other spiking devices), so the maximum GPU memory required by this algorithm is well-defined.

In MPI connections, when a source node (a neuron or another spiking device) is connected to target nodes of another host, a spike buffer, similar to the local one, is created in the remote host. When the source node fires a spike, this is sent to its spike buffer of the remote host, which delivers the spike to all target neurons after proper delays.

The computational cost per time step of the spike-buffer update algorithm is $c_s \times N \times B$, where c_s is the cost of a single

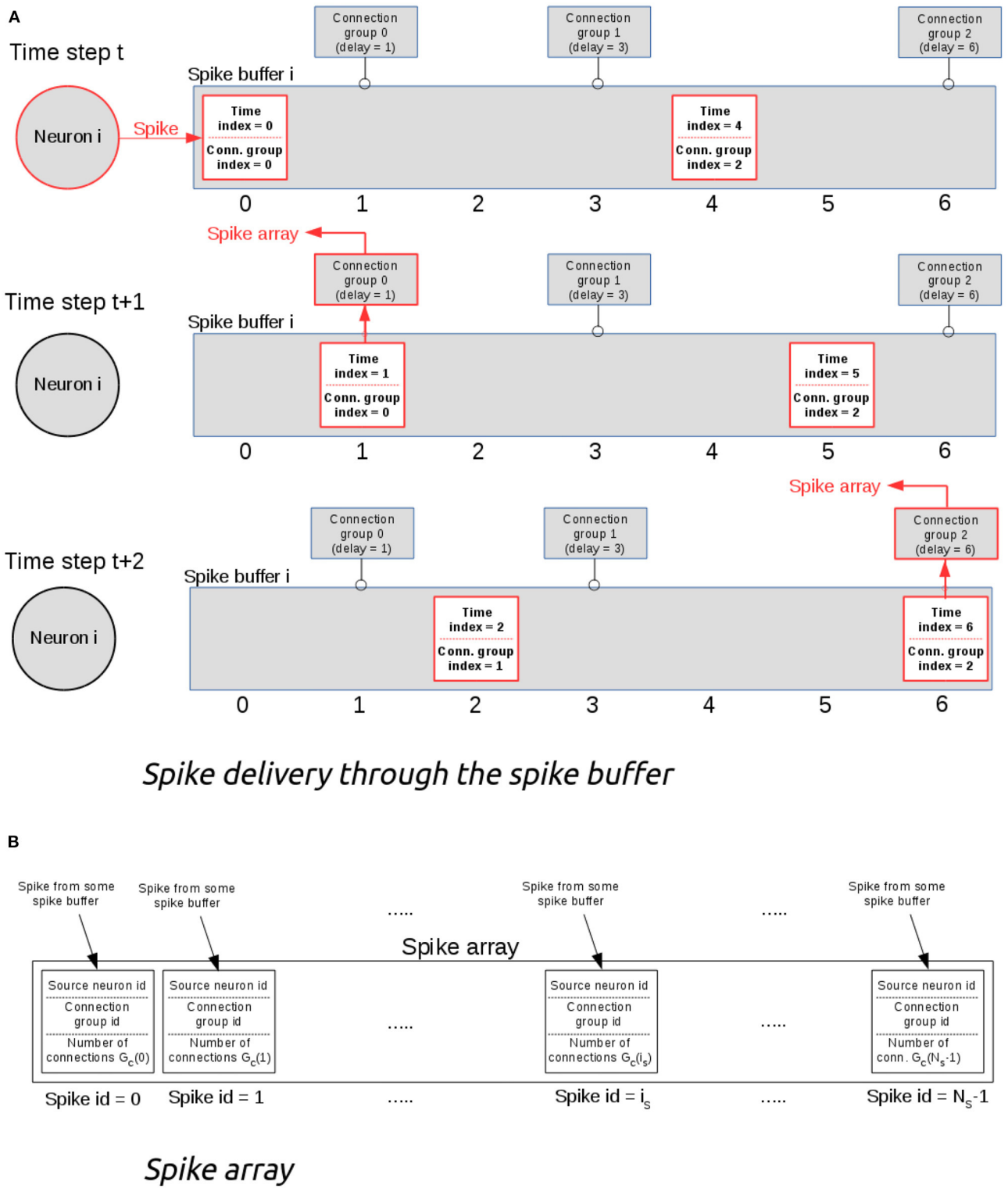


FIGURE 1 | (A) Example of spike delivery through the spike buffer. At time t, the i-th neuron emits a spike which is inserted in the spike buffer. In this example, the buffer contains also another spike emitted previously. At each time step, the spike time index is incremented by 1. When it becomes equal to the delay of some connection group, the spike is delivered to that group and its connection group index is incremented by 1. **(B)** The spike array. When the time index of a spike matches the delay of a connection group, the spike is sent to the spike array, which is used for delivering the spike to all neurons of the connection group.

spike update and B is the average number of spikes stored in a spike buffer. If we call $d_{\max}(i)$ the maximum delay, expressed in time step units, of the outgoing synaptic connections of the ith neuron, and $\langle d_{\max}(i)\rangle$ its average over all the neurons, B can be expressed as

$$B = F \times \langle d_{\max}(i)\rangle \times dt \tag{4}$$

and therefore the cost of the spike buffer update is

$$c_s \times N \times F \times \langle d_{\max}(i)\rangle \times dt \tag{5}$$

It should be observed that $\langle d_{\max}(i)\rangle$ is less than or equal to `max_delay`, which is the maximum delay of all synaptic connections of the network and can be expressed as `max_delay` $= \max_i\{d_{\max}(i)\}$, therefore the order of the computational cost of the proposed approach is smaller than or equal to that proposed by Nageswaran et al.

The computational cost per simulation time step for writing and reading the spikes to and from the spike array is $O(N \times F \times dt)$. This contribution is usually much smaller than the cost of neuron dynamics update, which is $O(N)$, because in realistic conditions $F \times dt \ll 1$. The computational cost per simulation time step for delivering the spikes from the spike array to the target neurons is

$$c_d \times N \times F \times C \times dt \tag{6}$$

where c_d is the cost for delivering a single spike. By comparing this cost with that of the spike buffer update, it can be observed that when

$$C \gg \langle d_{\max}(i)\rangle \times c_s/c_d \tag{7}$$

the delivery of the spikes to the target neurons gives the main contribution to the computational cost. This is usually the case when the number of connections per neuron is of the order of hundreds or more. An advantage of the proposed approach is that the delivery of the spikes from the spike array to the target neurons requires a small number of global memory accesses per delivery, therefore c_d is relatively small.

2.3. The Potjans-Diesmann Cortical Microcircuit Model

The cortical microcircuit model used in this work was developed in 2014 by Potjans and Diesmann (2014) and describes a portion of 1 mm^2 of sensory cortex, comprising approximately 77,000 LIF neurons organized into layers 2/3, 4, 5, and 6. Each layer contains an excitatory and an inhibitory population of LIF neurons with current-based synapses, for a total of eight populations: 2/3I, 2/3E, 4I, 4E, 5I, 5E, 6I, and 6E. The number of neurons in each population, the connection probability matrix and the rates of the external Poisson inputs are based on the integration of anatomical and physiological data mainly from cat V1 and rat S1. The total number of connections is about $3 \cdot 10^8$. **Figure 2** shows a diagram of the model with a schematic representation of the connections having probabilities >0.04.

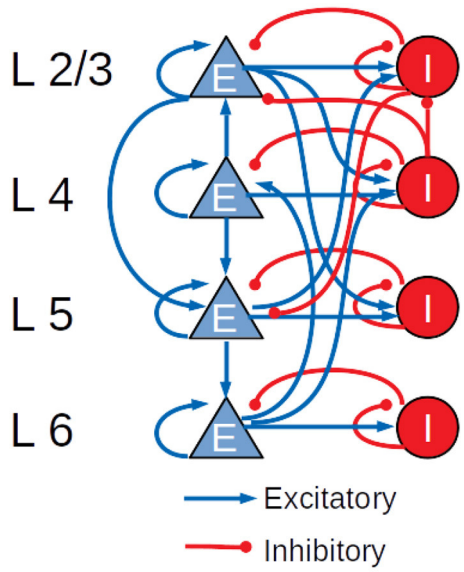


FIGURE 2 | Schematic diagram of the Potjans-Diesmann cortical microcircuit model.

The LIF neuron model, used in the cortical microcircuit, is one of the simplest spiking neuron models. The neuron dynamics is modeled by the following differential equation

$$\tau_m \frac{dV_i}{dt} = -(V_i - E_L) + R_m I_{\text{syn},i} \tag{8}$$

where $V_i(t)$ represents the membrane potential of neuron i at time t, τ_m is the membrane time constant, E_L is the resting membrane potential, R_m is the membrane resistance and $I_{\text{syn},i}$ is the synaptic input current. In the exponential shaped postsynaptic currents (PSCs) model, which will be used to simulate the Potjans-Diesmann cortical microcircuit model, the input current is described by the following equation

$$\tau_{\text{syn}} \frac{dI_{\text{syn},i}}{dt} = -I_{\text{syn},i} + \sum_j w_{ij} \sum_{t_j^f} \delta(t - t_j^f) \tag{9}$$

where τ_{syn} is the synaptic time constant, w_{ij} are the connection weights and t_j^f are the spike times from presynaptic neuron j. The simulation time step is set to 0.1 ms.

2.4. The AdEx-Neurons Balanced Network Model

The performance of the library was also assessed on a balanced network of sparsely connected excitatory and inhibitory neurons (Brunel, 2000), using the AdEx neuron model with conductance-based synapses and synaptic conductance modeled by an alpha function (Roth and van Rossum, 2013). The differential equations underlying the neuron dynamics are solved using the fifth-order Runge Kutta method with adaptive step size. To our knowledge, other GPU simulators of large scale spiking neural networks do not support this method. For this reason, the results of

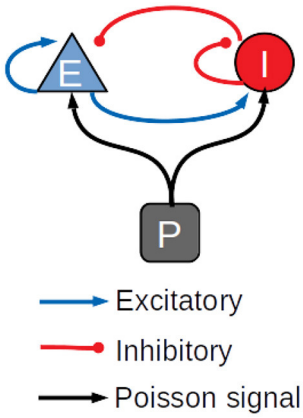


FIGURE 3 | Schematic diagram of the balanced network used in the simulations.

TABLE 1 | Values of the parameters used for the balanced network simulations.

Parameter	Value
N_{ex} (n. of excitatory neurons)	Variable
N_{in} (n. of inhibitory neurons)	$N_{ex}/4$
CE (n. of input excitatory synapses per neuron)	Variable
CI (n. of input inhibitory synapses per neuron)	$CE/4$
W_{ex} (excitatory connection weight)	0.05
W_{in} (inhibitory connection weight)	0.35
Mean delay	0.5 ms
Delay STD	0.25 ms
$W_{poisson}$ (Poisson signal weight)	0.37
$Rate_{poisson}$ (Poisson signal rate)	20,000 Hz
Neuron average firing rate	30.7 Hz

the simulations of the AdEx-neurons balanced network model are compared only with the CPU-based simulator NEST, which supports the same method. In general, GPU simulations work more efficiently with fixed step size; the adaptive step size is challenging and it was not obvious a priori that a GPU simulator could be faster than multi-core CPU systems with this kind of methods. Both populations of excitatory and inhibitory neurons are stimulated by an external Poissonian signal, as shown in **Figure 3**. Simulations have been made with a variable number of neurons and connections, with up to 1,000,000 neurons and 10^9 connections. **Table 1** represents the parameters used for the balanced network simulations.

The AdEx model represents an attractive neuron model for use in large-scale network simulations, because it is relatively simple compared to biologically detailed spiking neuron models, nonetheless it provides a good level of realism in representing the spiking behavior of biological neurons in many conditions, in the sense that it fits well the response of neurons as measured from electrophysiological recordings (Brette and Gerstner, 2005). This model is described by a system of two differential equations. The first equation describes the dynamics of the membrane potential

TABLE 2 | Values of the AdEx parameters used in the balanced network simulations.

Parameter	Value
C (Membrane capacitance)	281 pF
g_L (leak conductance)	30 nS
E_L (leak reversal potential)	−70.6 mV
V_T (spike initiation threshold)	−50.4 mV
Δ_T (slope factor)	2 mV
τ_w (adaptation time constant)	144 ms
a (subthreshold adaptation)	4 nS
b (spike-triggered adaptation)	80.5 pA
V_r (reset value of V_m after a spike)	−60 mV
E_{ex} (excitatory reversal potential)	0 mV
E_{in} (inhibitory reversal potential)	−85 mV
τ_{syn} (synaptic time constant)	1 ms

V(t) and includes an activation term with an exponential voltage dependence

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T e^{\frac{V-V_T}{\Delta_T}} + I_{syn}(V,t) - \omega + I_e \quad (10)$$

where the synaptic current is

$$I_{\text{syn}}(V,t) = \sum_i g_i(t)(V - E_{\text{rev},i}) \quad (11)$$

C is the membrane capacitance, g_L is the leak conductance, E_L is the leak reversal potential, Δ_T is a slope factor, V_T is the spike initiation threshold, ω is the spike-adaptation current, I_e is an external input current, $g_i(t)$ are the synaptic conductances and $E_{\text{rev},i}$ are the reversal potentials. The voltage is coupled to a second equation which describes adaptation

$$\tau_\omega \frac{d\omega}{dt} = a(V - E_L) - \omega \quad (12)$$

where τ_ω is the adaptation time-constant and a is the subthreshold adaptation parameter. When the neuron fires a spike, the adaptation current ω changes into $\omega \rightarrow \omega + b$, where b is a spike-triggered adaptation parameter, while the membrane potential changes into $V \rightarrow V_r$. **Table 2** reports the AdEx parameter values that have been used for the balanced network simulations. The time step for spike communication is set to 0.1 ms.

2.5. The Izhikevich-Neurons Balanced Network With STDP Synapses

The architecture of this model is still that shown in **Figure 3** and the ratio of excitatory to inhibitory neurons is the same as the model presented in the previous section. The other features of the model are listed below:

- Time step of 1 ms;
- 4-parameters Izhikevich neuron model (Izhikevich, 2003);

TABLE 3 | Values of the STDP parameters used in the Izhikevich-neurons balanced network simulations.

Parameter	Value
τ_+	20.0 ms
τ_-	20.0 ms
λ	0.001
α	1.0
μ_+	1.0
μ_-	1.0
w_{max}	10.0

- Current-based synapses described by an exponential-decay function;
- Euler forward integration method with two integration steps per simulation time step;
- 100 connections per neuron;
- Excitatory synapses change their weights according to the STDP rule, while inhibitory synapses have fixed weights;
- Average firing rate of 16 Hz for both excitatory and inhibitory populations.

The values of the Izhikevich-neuron parameters are $a = 0.02$, $b = 0.2$, $c = -65$, and $d = 8$. The synaptic decay time is $\tau_{decay} = 2$ ms. **Table 3** reports the values of the STDP parameters. The value of λ is small so that the weights do not change significantly during the simulation. It should be considered that the simulation time overhead due to STDP synapses depends only on the spike time distributions and not on the values of the STDP parameters if λ is sufficiently small.

3. RESULTS

The cortical microcircuit model and the balanced network described in the previous section were used both to verify the correctness of the simulations performed using NeuronGPU and to compare the performance of the proposed library with those of NEST version 2.20.0 (Fardet et al., 2020) and GeNN version 3.2.0 (neworderofjamie et al., 2018). For this purpose, we used a PC with a CPU Intel Core i9-9900 K with a frequency of 3.6 GHz and 8 cores featuring hyperthreading with two threads per core, for a total number of 16 hardware threads, 64 GB RAM, and a GPU card NVIDIA GeForce RTX 2080 Ti with 11 GB of GDDR6 VRAM, 4,352 CUDA cores, and a boost clock of 1,635 MHz. NeuronGPU and GeNN simulations were also performed on a system equipped with an NVIDIA Tesla V100 GPU with 16 GB GPU memory and 5,120 CUDA cores.

3.1. Simulation of the Cortical Microcircuit Model

Following the procedure proposed by van Albada et al. (2018) and by Knight and Nowotny (2018), in this section we will verify the correctness of the simulations by comparing some relevant statistical distributions extracted from the simulations of the Potjans-Diesmann cortical microcircuit model made using NeuronGPU with the analogous distributions obtained using the NEST simulator. Subsequently, still following the same line of van Albada et al. (2018) and Knight and Nowotny (2018), the cortical microcircuit model will be used as a benchmark to evaluate the performance of NeuronGPU in terms of building time and simulation time per unit time of biological activity.

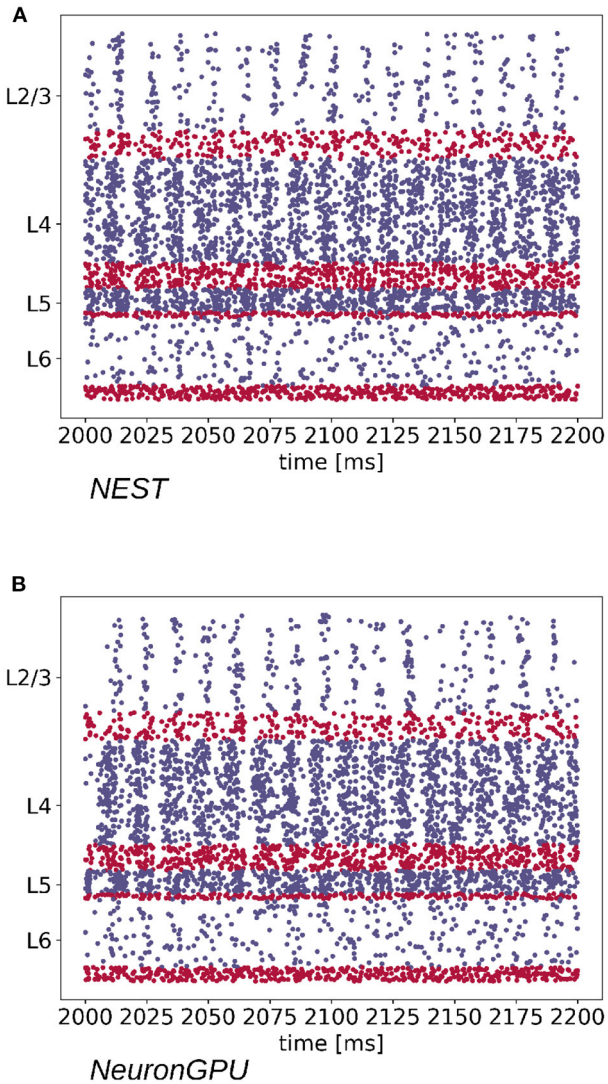


FIGURE 4 | Raster plot showing spike times (dots) of neurons from each population of the cortical microcircuit model, simulated using **(A)** NEST and **(B)** NeuronGPU, in a time window of 200 ms (in blue the excitatory and in red the inhibitory). Due to the high number of neurons in the model, only the spikes of one neuron out of ten are shown.

The Python code used for simulations, available in https://github.com/golosio/NeuronGPU/tree/master/python/Potjans_2014, is almost identical to the NEST implementation (https://nest-simulator.readthedocs.io/en/stable/microcircuit/).

Figure 4 shows a raster plot of the spike times of neurons from each population of the model, simulated using NEST and NeuronGPU, in a time window of 200 ms.

In order to verify the correctness of the simulations, we simulated 11 s of biological activity of the full-scale

Potjans-Diesmann model with both NeuronGPU and NEST, with a time step of 0.1 ms. For both simulators we performed 10 simulations, distinct from each other only for the initial seed for random number generation. As in van Albada et al. (2018) and Knight and Nowotny (2018), the first second was discarded in order to eliminate transient trends. The spike times of all neurons have been recorded during the simulations, and subsequently they have been used to extract three distributions for each population, namely:

- The average firing rate of the single neuron;
- The coefficient of variation of the inter-spike time interval (CV ISI), defined as the ratio between the standard deviation and the average of the inter-spike time intervals;
- The Pearson correlation between the spike trains.

The latter has been computed on a subset of 200 neurons for each population, as in van Albada et al. (2018) and Knight and Nowotny (2018). This number represents a compromise between statistical precision and computation time. The spike trains of those neurons have first been rebinned to a time step of 2 ms, equal to the refractory time. Denoting the

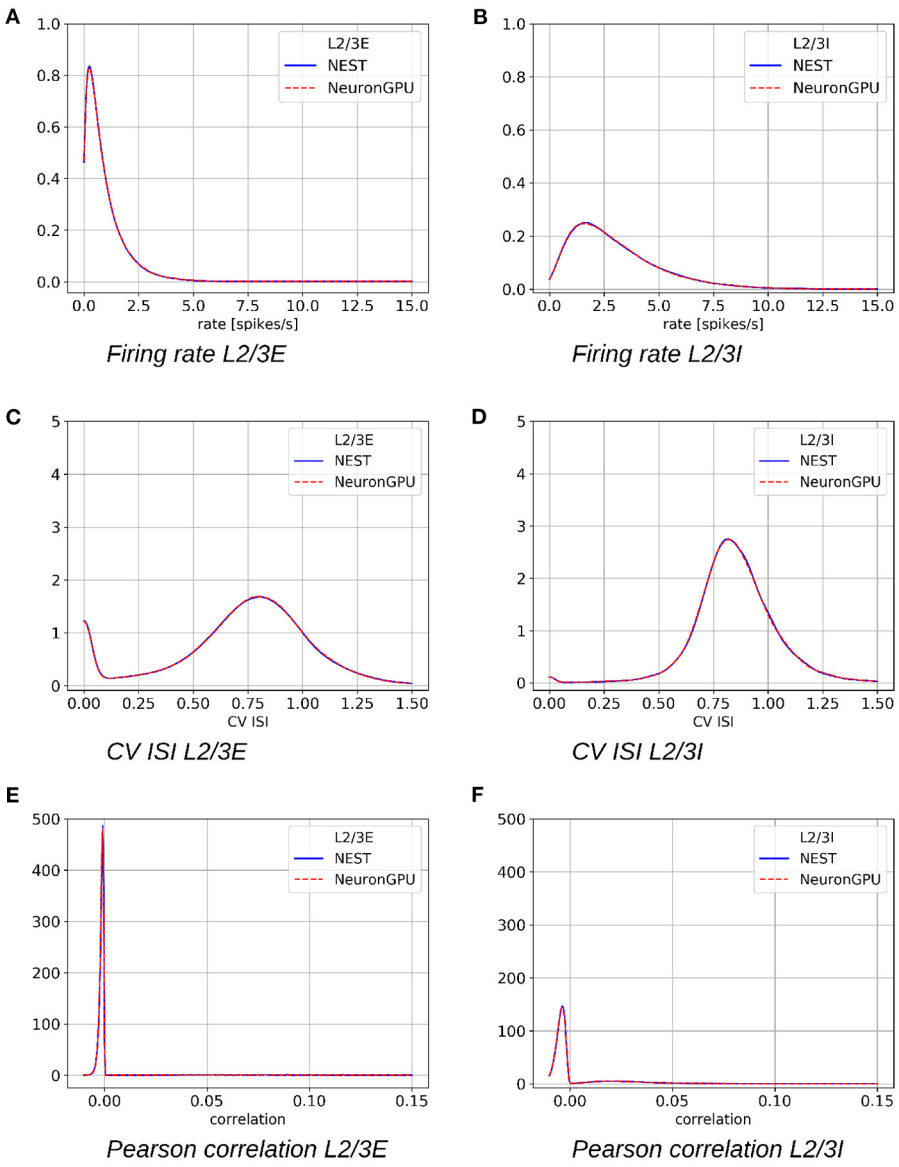


FIGURE 5 | Distribution of the firing rates, coefficient of variation of interspike intervals (CV ISI) and Pearson correlation coefficient of the spike trains for the populations L2/3E and L2/3I of the cortical microcircuit model, averaged over 10 simulations, made using NEST (blue) or NeuronGPU (red). **(A)** Firing rate L2/3E, **(B)** firing rate L2/3I, **(C)** CV ISI L2/3E, **(D)** CV ISI L2/3I, **(E)** Pearson correlation L2/3E, **(F)** Pearson correlation L2/3I.

binned spike trains as b_i and their mean value as μ_i, the correlation coefficient between the spike trains b_i and b_j is defined as

$$C[i,j] = < b_i - \mu_i, b_j - \mu_j > / \sqrt{< b_i - \mu_i, b_i - \mu_i > \cdot < b_j - \mu_j, b_j - \mu_j >}$$

where $<,>$ represents the scalar product. For 200 spike trains, a 200x200 correlation matrix is returned. The Pearson correlation distribution is evaluated as the distribution of the off-diagonal elements of this matrix. All distributions have been evaluated from the spike time recordings using the Elephant (Electrophysiology Analysis Toolkit) package (Denker et al., 2018), dedicated to the analysis of electrophysiological data in the Python environment. The distributions have been smoothed using the KDE (Kernel Density Estimation) method (Rosenblatt, 1956; Parzen, 1962), available in the *scikit-learn* Python library (Pedregosa et al., 2011) through the function `sklearn.neighbors.KernelDensity`. The KDE method allows to estimate the probability density of a random variable with a reduced dependence on random fluctuations linked to individual simulations. In particular, each of the N points belonging to a sample is represented by a Gaussian function of suitable width, called kernel bandwidth. The integral of each of these functions is normalized to $1/N$; the overall distribution is therefore estimated as the sum of all these Gaussians, and obviously it has an integral normalized to one. The kernel bandwidth has been optimized using the so-called Silverman's rule (Silverman, 1986), which prescribes a bandwidth value of

$$b = 0.9 \cdot \min\left(\hat{\sigma}, \frac{\text{IQR}}{1.349}\right) \cdot N^{-\frac{1}{5}} \tag{13}$$

where $\hat{\sigma}$ is the standard deviation of the samples, N is the sample size and IQR is the interquartile range. It should be observed that the distributions obtained through the KDE method are continuous

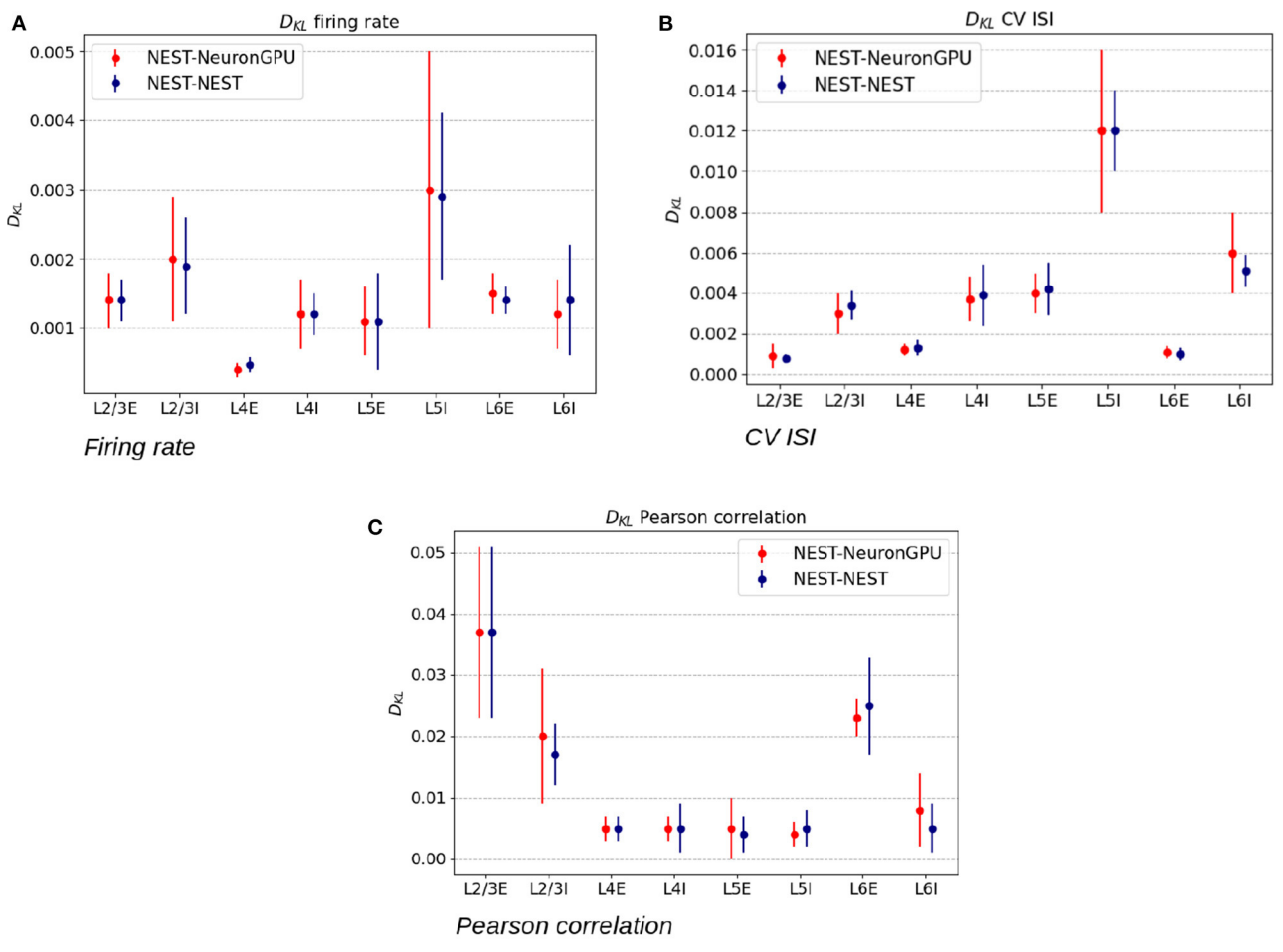


FIGURE 6 | Kullback-Leibler divergence between the distributions of the firing rate **(A)**, coefficient of variation of interspike intervals **(B)**, and Pearson correlation coefficient **(C)**, extracted from NEST and NeuronGPU simulations. The red error bars represent the average values and the standard deviations of the divergence between NEST and NeuronGPU, while the blue ones represent the same values for NEST simulations with different seeds.

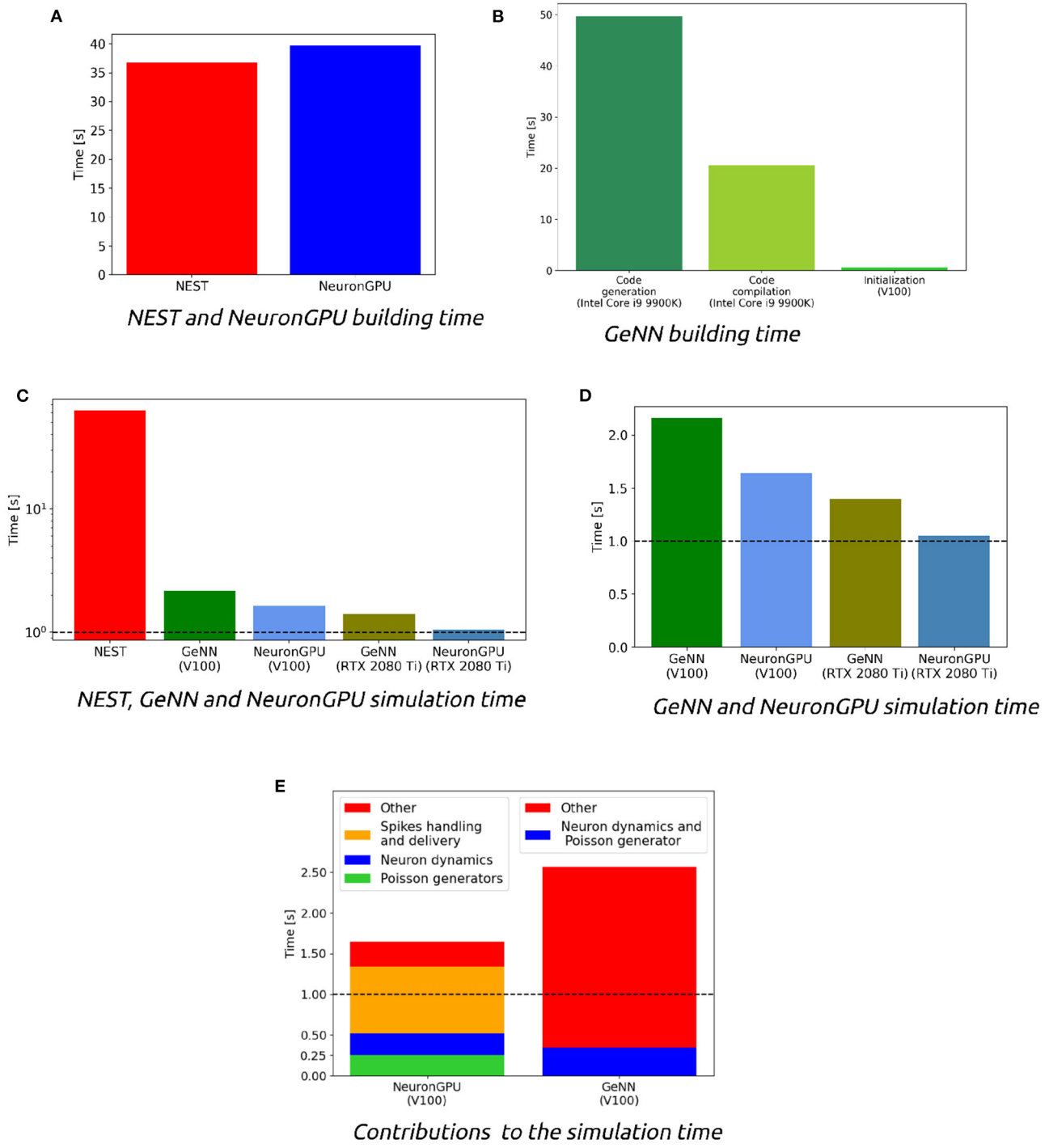


FIGURE 7 | (A) Building time of the cortical microcircuit model simulated with NEST and with NeuronGPU on a system equipped with an Intel Core i9-9900K CPU. **(B)** Times for code generation, compilation, and initialization of the cortical microcircuit model for GeNN. The first two phases are performed by the CPU, and the times refer to a system equipped with an Intel Core i9-9900K. The third phase is mainly performed by the GPU, and the figure shows the time for an NVIDIA Tesla V100. **(C,D)** Simulation times per second of biological time of the cortical microcircuit model simulated with NEST, NeuronGPU, and GeNN on various CPU and GPU hardware. **(E)** Contributions of neuron dynamic update time, Poisson generator time, and spike handling and delivery time to the total simulation time of the Potjans-Diesmann model, simulated using NeuronGPU and GeNN on a Tesla V100 GPU. In GeNN the Poissonian input signal is generated within the same code that manages the neuron's dynamics, and furthermore it was not possible to separate the time used for spike handling and delivery from the remaining contributions to the simulation time. The horizontal line represents the biological time. The simulation time step is set to 0.1 ms.

functions, since they are evaluated as the sum of a set of Gaussian functions.

Figure 5 shows the distributions of the firing rate, the CV ISI and the Pearson correlation coefficient for two populations of the Potjans-Diesmann model, averaged over 10 simulations made using NEST or NeuronGPU. As can be seen in the graphs, the distributions obtained from the two simulators are very similar to each other. This is also true for the other populations of the model. In order to compare quantitatively the distributions obtained using NeuronGPU to those obtained using NEST, we evaluated the Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951), defined as $D_{KL}(p_1, p_2) = -\sum_i p_{1,i} \log(p_{1,i}/p_{2,i})$, where p_1 and p_2 are two distributions, and the index i runs on the sampling points of the two distributions. For this purpose, we used 10 pairs of simulations (NeuronGPU-NEST and NEST-NEST) using different seeds for random number generation. The KL divergence was then calculated for each pair and its average and standard deviation were calculated on the 10 pairs. Since the KDE method provides a smooth continuous function, the result is not sensitive to the sampling step as long as this is small enough. The KL divergence was evaluated using the Python scientific library (Virtanen et al., 2020) and in particular the `scipy.stats.entropy` function. **Figure 6** shows the average and standard deviation of the KL divergences between the distributions of firing rates, CV ISI, and Pearson correlation, obtained from NEST and from NeuronGPU simulations, for the eight populations of the cortical microcircuit model. It can be observed that the KL divergence between distributions obtained from NEST and from NeuronGPU are perfectly compatible with the divergence between distributions obtained from NEST simulations with different seeds. To compare the performance of NeuronGPU with those of NEST and GeNN, we performed a series of 10 simulations of 10 s of biological activity of the cortical microcircuit with each simulator, using different seeds for random number generation. The execution time of the simulations can be divided into building time and simulation time of biological activity. The building time includes the time needed to allocate memory for connections, neurons, spike generators, and recording devices, to build connections and to initialize the values of state variables and parameters. **Figure 7A** shows the building time for NEST and NeuronGPU. On a system equipped with an Intel Core i9-9900 K CPU, the building times were 36.8 ± 0.6 and 39.7 ± 0.4 s for NEST and NeuronGPU, respectively. The building time of NeuronGPU is comparable to that of NEST. This is due to the fact that in NeuronGPU the connections are initially created in the RAM, and only immediately before the simulation they are copied from RAM to GPU memory. The times for code generation, compilation, and initialization of the cortical microcircuit model with GeNN were 49.7, 20.6, and 0.65 s, respectively, as shown in **Figure 7B**. Importantly, since GeNN uses a code-generation approach, while in NeuronGPU the models are created dynamically, the building

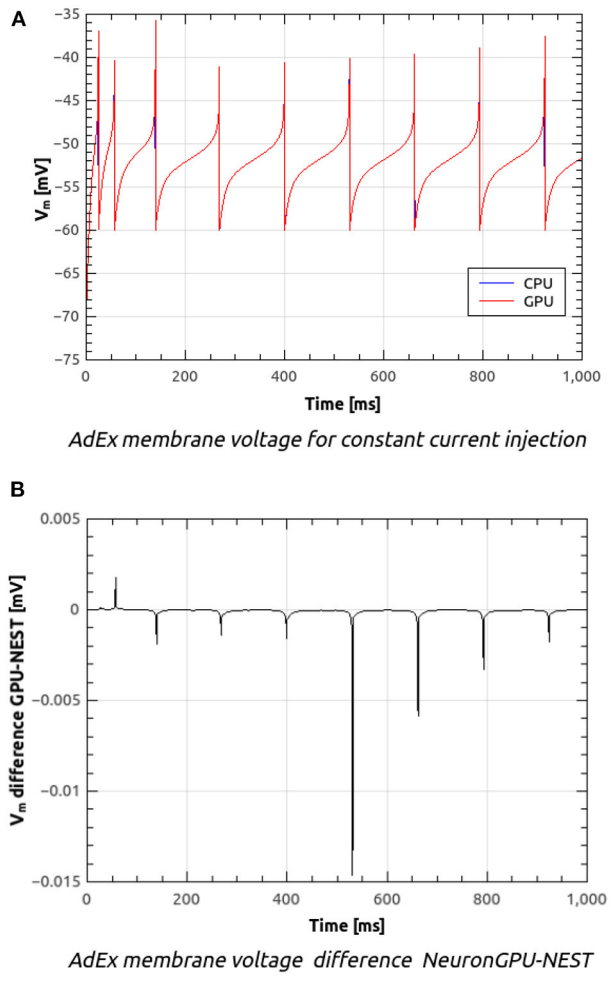


FIGURE 8 | Membrane voltage of an AdEx neuron with the parameter values reported in **Table 2** and an injected current of 700 pA, simulated with NeuronGPU and with NEST **(A)** and difference between the two simulation signals **(B)**.

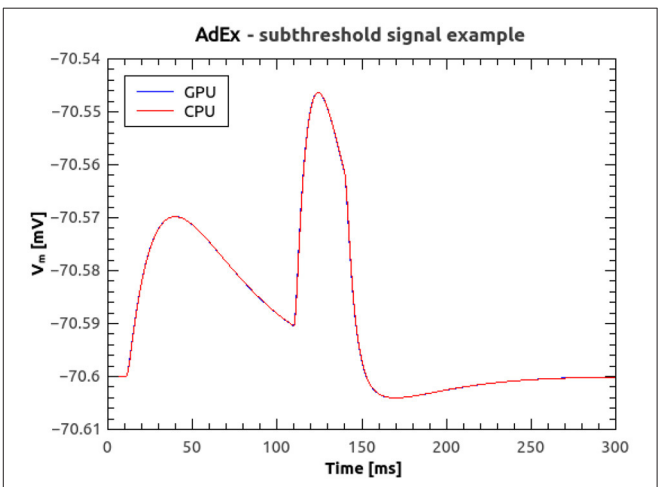


FIGURE 9 | Membrane voltage of an AdEx neuron stimulated by three input spikes in a subthreshold condition, simulated using NeuronGPU and NEST.

times of GeNN and NeuronGPU cannot be directly compared. In GeNN the code of the model is generated from C/C++-like code fragments and it must be compiled before execution. Any changes in the model parameters require a new generation and compilation of the code. Once the code is generated and compiled, the initialization is very fast. **Figures 7C,D** show the simulation times per unit time of biological activity for NeuronGPU, NEST and GeNN on different CPU and GPU platforms. The simulation time per second of biological time with NEST running on the Intel Core i9-9900K CPU was 62.7 ± 0.3 s. On a system equipped with an NVIDIA Tesla V100 GPU card, the simulation time per second of biological time with GeNN was 2.16 s. NeuronGPU was 31.6% faster than GeNN, with a simulation time of 1.641 ± 0.014 s on the same GPU. On an NVIDIA RTX 2080 Ti GPU card, the simulation time per second of biological time with GeNN was 1.398 ± 0.007 s, while NeuronGPU was 32.5% faster with a simulation time of 1.055 ± 0.004 s. **Figure 7E** shows the contributions of neuron dynamic update time, Poisson generator time and spike handling and delivery time to the total simulation time of the Potjans-Diesmann model, simulated using NeuronGPU and GeNN on a Tesla V100 GPU. It should be noted that while in the case of NeuronGPU the Poissonian input signal is generated by external Poisson spike generators connected to the neurons, in the case of GeNN this is generated within the same code that manages the neuron's dynamics. Furthermore, in the case of GeNN it was not possible to separate the time used for spike handling and delivery from the remaining contributions to the simulation time.

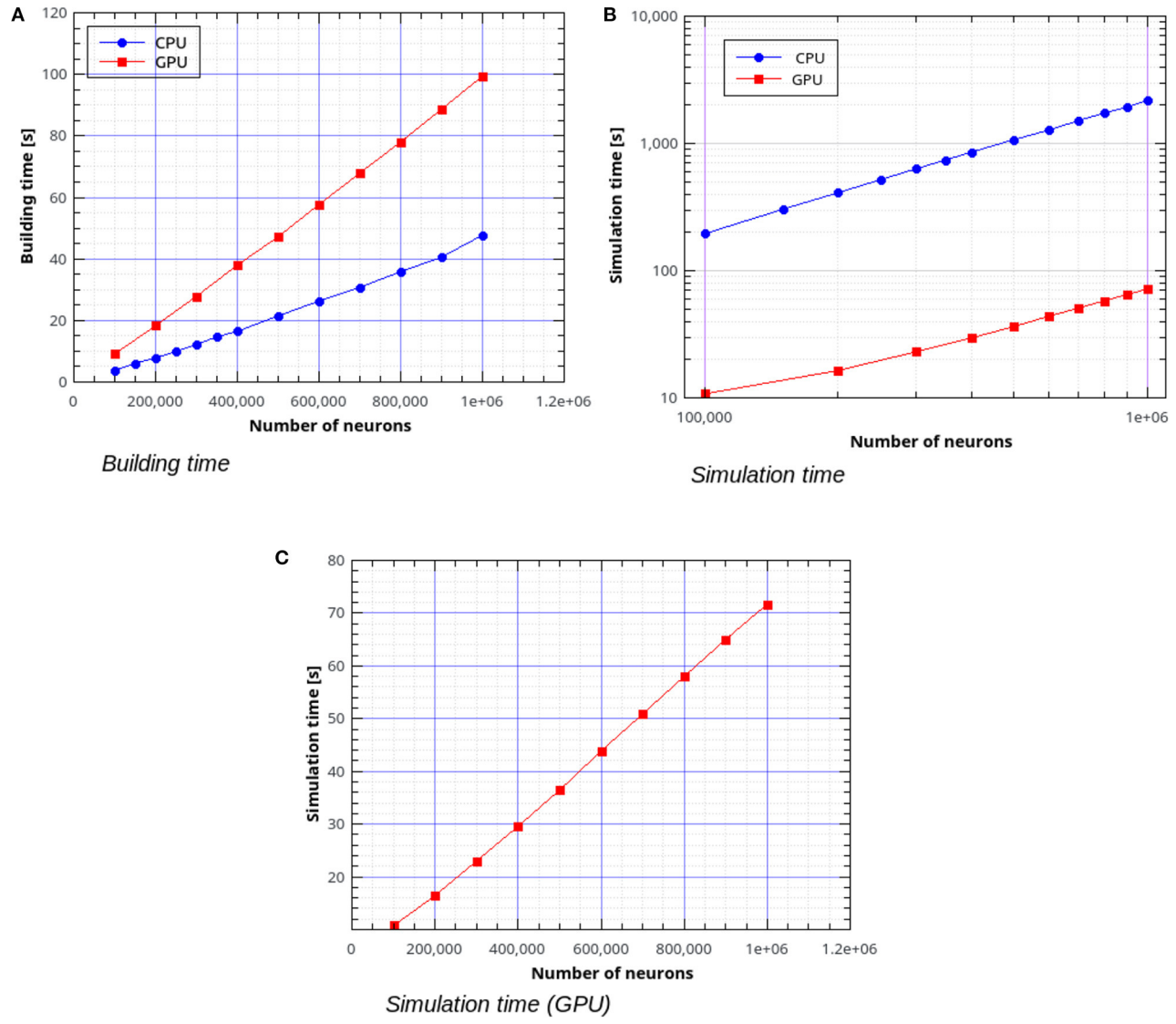


FIGURE 10 | Building time **(A)** and simulation time **(B)** for the balanced network simulations with a variable number of neurons and a fixed number of 1,000 input connections per neuron, simulated using NeuronGPU and NEST, and simulation time for NeuronGPU shown on a different scale **(C)**. The time step for spike communication is set to 0.1 ms.

In the case of NeuronGPU, excluding the neuron dynamic update time and the Poisson generator time, most of the remaining simulation time is spent on handling and delivering the spikes. Assuming this is also the case with GeNN, the improvement in the simulation time of NeuronGPU over GeNN would be mainly due to a more efficient approach in spikes handling and delivery.

3.2. Simulation of the AdEx-Neurons Balanced Network Model

Figure 8A shows the time course of the membrane voltage of an AdEx neuron with the parameter values reported in **Table 2** and an injected current of 700 pA, simulated with NeuronGPU and with NEST. With the exception of the peaks, the two plots appear to be perfectly superimposed on this scale. **Figure 8B** represents the difference between the two signals simulated with NEST and with NeuronGPU. Apart from the peaks, the difference is in the order of a few 10^{-4} mV. **Figure 9** shows the time course of the membrane voltage of an AdEx neuron stimulated by three input spikes on three different receptor ports in a subthreshold condition, simulated with NeuronGPU and with NEST.

In the remaining part of this section we present the results of simulations of the AdEx-neurons balanced network with the parameters shown in **Tables 1**, **2**. **Figure 10A** shows the building time for the balanced network simulations as a function of the number of neurons, for a fixed number of 1,000 input connections per neuron. **Figures 10B,C** represent the simulation time per second of biological activity of the balanced network as a function of the total number of neurons. It can be observed that the GPU simulations are faster than the CPU's by a factor ranging from about 18× for 100,000 neurons with 10^8 connections to 30.4× for 10^6 neurons with 10^9 connections.

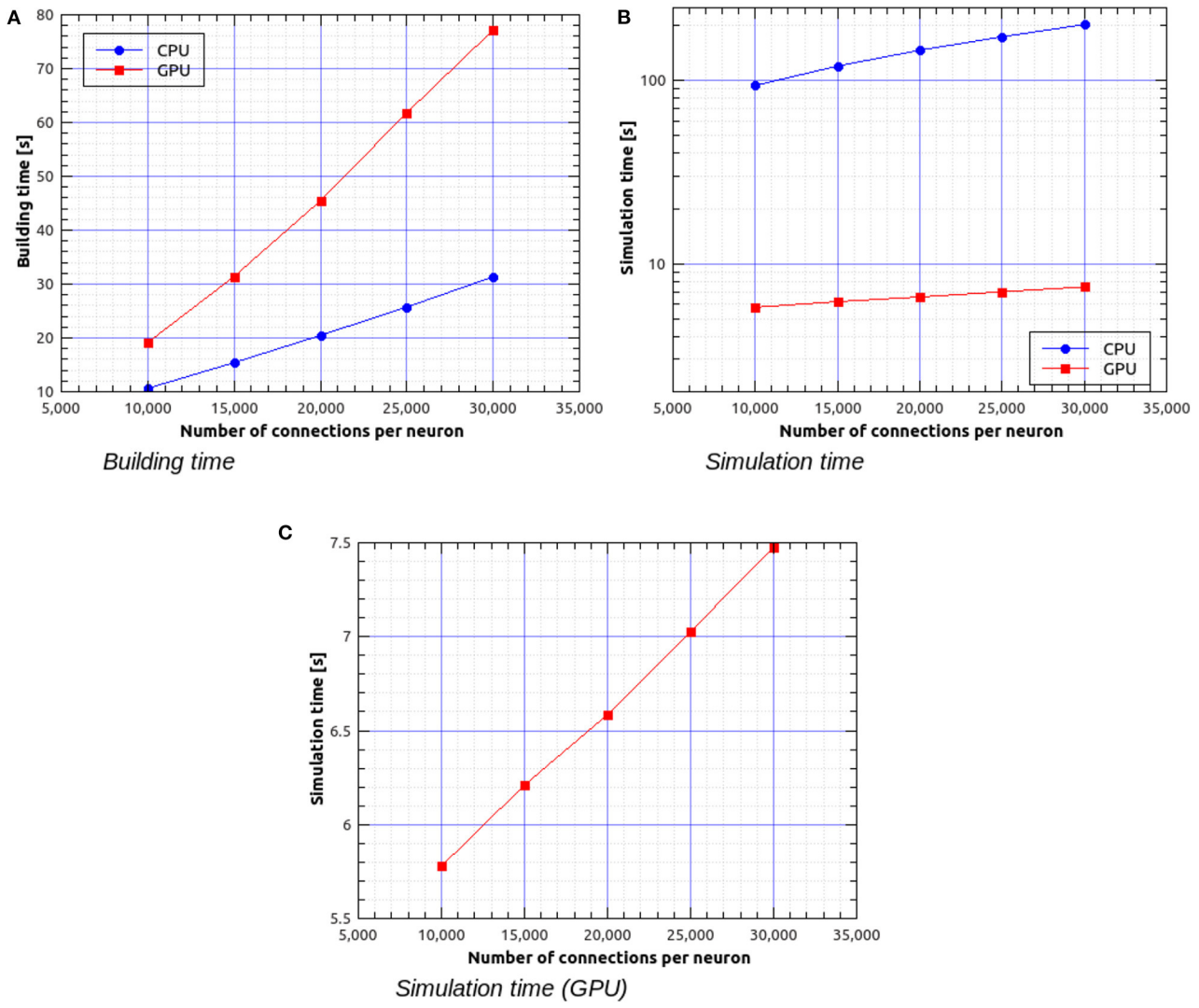


FIGURE 11 | Building time **(A)** and simulation time **(B)** for the balanced network simulations with a fixed number of 30,000 neurons and a variable number of input connections per neuron, simulated using NeuronGPU and NEST, and simulation time for NeuronGPU shown on a different scale **(C)**. The time step for spike communication is set to 0.1 ms.

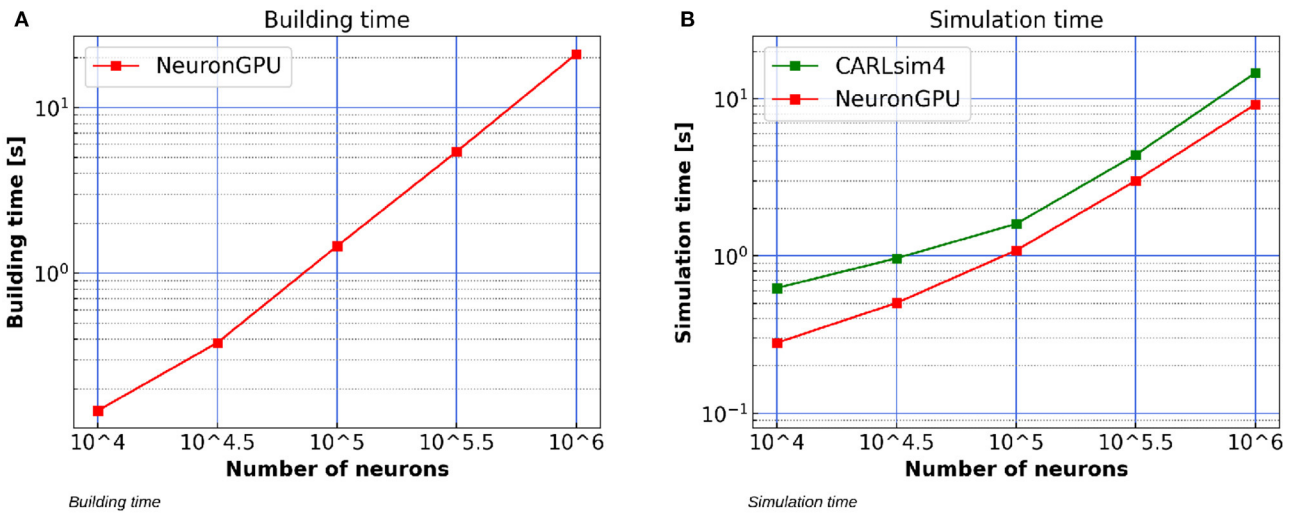


FIGURE 12 | (A) Building time of the Izhikevich-neurons balanced network model as a function of the number of neurons, simulated using NeuronGPU on a system equipped with a Tesla K80 GPU. **(B)** Simulation time per second of biological activity of the Izhikevich-neurons balanced network model simulated using NeuronGPU and CARLsim 4 on a Tesla K80 GPU. The data for the latter are taken from Chou et al. (2018). STDP plasticity is active on excitatory connections. The simulation time step is set to 1.0 ms.

Figure 11A shows the building time as a function of the number of connections per neuron for a fixed total number of neurons, which was set to 30,000. **Figures 11B,C** represent the simulation time as a function of the number of connections per neuron. It can be observed that, in this case, simulations on GPU are faster than on CPU by a factor ranging from about 16× for 30,000 neurons with $3 \cdot 10^8$ connections to about 27× for 30,000 neurons with $9 \cdot 10^8$ connections.

3.3. Simulation of the Izhikevich-Neurons Balanced Network With STDP Synapses

Figure 12A shows the building time of the Izhikevich-neurons balanced network model as a function of the number of neurons, simulated using NeuronGPU on a system equipped with an Intel Xeon E5-2686 v4 processor, 64 GB RAM and a Tesla K80 GPU. **Figure 12B** compares the simulation time per second of biological activity of the model simulated using NeuronGPU with that of CARLsim 4. The simulation times for the latter are taken from Chou et al. (2018), which reports that the simulations were performed on a system that was also equipped with a Tesla K80 GPU card, while the CPU model and the amount of RAM of the system are not specified. It can be observed that the simulation time of NeuronGPU is lower than that of CARLsim 4 in the considered interval. In particular, for 10^6 neurons and 10^8 connections NeuronGPU is about 59% faster than CARLsim 4.

4. DISCUSSION

As it can be observed in **Figure 7**, the building time of the cortical microcircuit model simulated using NeuronGPU is comparable to that of NEST, mainly because in NeuronGPU the connections are created in the RAM and only immediately before the simulation loop they are copied to the GPU memory. Compared to most GPU-based simulators, NeuronGPU offers a wide range of choices for connection rules and connection parameter distributions, which can be exploited at runtime and interactively through the Python interface. It is easier to manage these connection rules and these distributions on the CPU side, also thanks to the functions provided by the standard C++ library. In both NEST and NeuronGPU the model parameters, the neuron populations and the network architecture are defined at runtime and the memory they need is allocated dynamically. On the other hand, GeNN uses a code-generation approach. The model parameters, neuron populations and architecture are defined using code fragments similar to C/C++, from which the CUDA/C ++ code of the model is generated. This code must be compiled before execution. Any changes in the parameters, neuron populations or network architecture require a new generation and compilation of the code. Once the code is generated and compiled, the initialization is very fast as it is carried out directly by the GPU with parallel computing algorithms. On the other hand, NeuronGPU achieved a simulation time per second of biological activity of 1.64 s on an NVIDIA Tesla V100 GPU and of 1.055 s on an NVIDIA RTX 2080 Ti GPU, about 32% faster than GeNN, 59x faster than NEST and very close to biological time. Moreover, NeuronGPU was about 59% faster than CARLsim 4 in terms of simulation time per second of biological activity in the simulation of the Izhikevich-neurons balanced network with 10^6 neurons and 10^8 STDP synaptic connections. The building time of the AdEx-neurons balanced network simulated using NeuronGPU was about twice as large as that of NEST. However, NeuronGPU was faster than NEST in terms of simulation time per second of biological activity by a factor ranging from about 16× for smaller networks to about 30× for networks with 10^9 connections. In

future releases of the library, the building time could significantly be reduced by creating the connections directly in the GPU memory, exploiting the parallel computing capabilities of the GPU and avoiding the bottleneck of memory transfer from RAM to GPU memory. Besides the relatively long building time, NeuronGPU has other limitations compared to other GPU simulators. In particular, it currently does not include multi-compartment models. The only type of synaptic plasticity available is nearest-neighbor STDP. Neuromodulation is also not included. Multi-GPU simulations are only supported via MPI, which is yet to be evaluated. User defined models are supported, however there is currently no dedicated interface to configure them; the list of state variables and parameters and the differential equations of the dynamics must be modified directly in the code, which has to be recompiled. On the other hand, the high simulation speed demonstrated by the proposed library, significantly higher than that of other CPU and GPU based simulators, combined with the availability of a wide range of neuron models, spike generators, recording tools, and connection rules, makes this library particularly useful for simulations of large spiking neural networks over relatively long biological times. NeuronGPU was recently proposed for being integrated with the NEST neural simulator (Golosio et al., 2020). The high degree of similarity between the Python interfaces of NEST and NeuronGPU immediately simplifies porting scripts from one simulator to the other, and opens the door to integration and cosimulations between NEST and NeuronGPU.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found at: https://github.com/golosio/ngpu_cortical_circuits_paper; https://github.com/golosio/NeuronGPU.

AUTHOR CONTRIBUTIONS

BG, GT, and PP wrote the manuscript. BG is the main developer of NeuronGPU. BG and PP designed the experiments. All authors contributed to conducting the experiments, analyzing the results, and reviewed the manuscript.

FUNDING

This work has been partially supported by the European Union Horizon 2020 Research and Innovation program under the FET Flagship Human Brain Project (grant agreement SGA3 n. 945539 and grant agreement SGA2 n. 785907) and by the INFN APE Parallel/Distributed Computing laboratory. We acknowledge the use of Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the grant agreement No. 800858.

ACKNOWLEDGMENTS

We are grateful to Prof. Hans Ekkehard Plesser and to Dr. Tanguy Fardet for their revision of the aeif_cond_beta_multisynapse model in the NEST simulator, which was the basis for the implementation of the same model in NeuronGPU. We would also like to thank Prof. Plesser, Prof. Markus Diesmann, Dr. Alexander Peyser, and Dr. Wouter Klijn for the useful discussions on the dynamics of spiking neural networks, the use of CPU and GPU clusters and the spike-delivery algorithms.

REFERENCES

Brette, R., and Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.* 94, 3637–3642. doi: 10.1152/jn.00686.2005

Brette, R., and Goodman, D. F. M. (2012). Simulating spiking neural networks on GPU. *Network* 23, 167–182. doi: 10.3109/0954898X.2012.730170

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.* 23, 349–398. doi: 10.1007/s10827-007-0038-6

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci.* 8, 183–208. doi: 10.1023/A:1008925309027

Carnevale, N. T., and Hines, M. L. (2006). *The NEURON Book*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511541612

Chou, T.-S., Kashyap, H. J., Xing, J., Listopad, S., Rounds, E. L., Beyeler, M., et al. (2018). "CARLsim 4: an open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro). doi: 10.1109/IJCNN.2018.8489326

Denker, M., Yegenoglu, A., and Grün, S. (2018). "Collaborative HPC-enabled workflows on the HBP collaboratory using the elephant framework," in *Neuroinformatics 2018*, P19.

Eisenstat, S. C., Gursky, M. C., Schultz, M. H., and Sherman, A. H. (1982). Yale sparse matrix package i: the symmetric codes. *Int. J. Numer. Methods Eng.* 18, 1145–1151. doi: 10.1002/nme.1620180804

Fardet, T., Vennemo, S. B., Mitchell, J., Mørk, H., Graber, S., Hahne, J., et al. (2020). *NEST 2.20.0*. Genève.

Garrido, J. A., Carrillo, R. R., Luque, N. R., and Ros, E. (2011). "Event and time driven hybrid simulation of spiking neural networks," in *Advances in Computational Intelligence* (Berlin; Heidelberg: Springer), 554–561. doi: 10.1007/978-3-642-21501-8_69

Golosio, B., De Luca, C., Pastorelli, E., Simula, F., Tiddia, G., and Paolucci, P. S. (2020). "Toward a possible integration of NeuronGPU in NEST," in *NEST Conference 2020*, 7 (Aas). doi: 10.5281/zenodo.4501615

Goodman, D., and Brette, R. (2008). Brian: a simulator for spiking neural networks in Python. *BMC Neurosci.* 9:P92. doi: 10.1186/1471-2202-9-S1-P92

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Izhikevich, E. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440

Knight, J. C., and Nowotny, T. (2018). GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Front. Neurosci.* 12:941. doi: 10.3389/fnins.2018.00941

Kullback, S., and Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Stat.* 22, 79–86. doi: 10.1214/aoms/1177729694

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybern.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Nageswaran, J. M., Dutt, N., Krichmar, J. L., Nicolau, A., and Veidenbaum, A. V. (2009). A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Netw.* 22, 791–800. doi: 10.1016/j.neunet.2009.06.028

neworderofjamie, Nowotny, T., Turner, J. P., Yavuz, E., Zhang, M., Diamond, A., et al. (2018). *genn-team/genn: Genn 3.2.0*. Genève. doi: 10.5281/zenodo.1478540

Parzen, E. (1962). On estimation of a probability density function and mode. *Ann. Math. Stat.* 33, 1065–1076. doi: 10.1214/aoms/1177704472

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* 12, 2825–2830. Available online at: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

Potjans, T. C., and Diesmann, M. (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cereb. Cortex* 24, 785–806. doi: 10.1093/cercor/bhs358

Press, W. H., and Teukolsky, S. A. (1992). Adaptive stepsize runge-kutta integration. *Comput. Phys.* 6:188. doi: 10.1063/1.4823060

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Ann. Math. Stat.* 27, 832–837. doi: 10.1214/aoms/1177728190

Roth, A., and van Rossum, M. (2013). "Chapter 6: Modeling synapses," in *Computational Modeling Methods for Neuroscientists*, ed E. D. Schutter (Cambridge, MA: MIT Press), 266–290.

Rotter, S., and Diesmann, M. (1999). Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biol. Cybern.* 81, 381–402. doi: 10.1007/s004220050570

Sanders, J., and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Upper Saddle River, NJ: Addison-Wesley.

Sboev, A., Vlasov, D., Serenko, A., Rybka, R., and Moloshnikov, I. (2016). On the applicability of STDP-based learning mechanisms to spiking neuron network models. *AIP Adv.* 6:111305. doi: 10.1063/1.4967353

Silverman, B. W. (1986). Density estimation for statistics and data analysis. London: Chapman and Hall. doi: 10.1007/978-1-4899-3324-9

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Front. Neurosci.* 12:291. doi: 10.3389/fnins.2018.00291

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods* 17, 261–272. doi: 10.1038/s41592-019-0686-2

Vitay, J., Dinkelbach, H. U., and Hamker, F. H. (2015). ANNarchy: a code generation approach to neural simulations on parallel hardware. *Front. Neuroinform.* 9:19. doi: 10.3389/fninf.2015.00019

Wang, R. M., Thakur, C. S., and van Schaik, A. (2018). An FPGA-based massively parallel neuromorphic cortex simulator. *Front. Neurosci.* 12:213. doi: 10.3389/fnins.2018.00213

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. *Sci. Rep.* 6:18854. doi: 10.1038/srep18854

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 16 March 2021
doi: 10.3389/fncom.2021.630795

Granular layEr Simulator: Design and Multi-GPU Simulation of the Cerebellar Granular Layer

Giordana Florimbi[1], *Emanuele Torti*[1]*, *Stefano Masoli*[2], *Egidio D'Angelo*[2,3] and *Francesco Leporati*[1]

[1] Custom Computing and Programmable Systems Laboratory, Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia, Italy, [2] Neurocomputational Laboratory, Neurophysiology Unit, Department of Brain and Behavioral Sciences, University of Pavia, Pavia, Italy, [3] Istituti di Ricovero e Cura a Carattere Scientifico (IRCCS) Mondino Foundation, Pavia, Italy

In modern computational modeling, neuroscientists need to reproduce long-lasting activity of large-scale networks, where neurons are described by highly complex mathematical models. These aspects strongly increase the computational load of the simulations, which can be efficiently performed by exploiting parallel systems to reduce the processing times. Graphics Processing Unit (GPU) devices meet this need providing on desktop High Performance Computing. In this work, authors describe a novel Granular layEr Simulator development implemented on a multi-GPU system capable of reconstructing the cerebellar granular layer in a 3D space and reproducing its neuronal activity. The reconstruction is characterized by a high level of novelty and realism considering axonal/dendritic field geometries, oriented in the 3D space, and following convergence/divergence rates provided in literature. Neurons are modeled using Hodgkin and Huxley representations. The network is validated by reproducing typical behaviors which are well-documented in the literature, such as the center-surround organization. The reconstruction of a network, whose volume is 600 × 150 × 1,200 μm^3 with 432,000 granules, 972 Golgi cells, 32,399 glomeruli, and 4,051 mossy fibers, takes 235 s on an Intel i9 processor. The 10 s activity reproduction takes only 4.34 and 3.37 h exploiting a single and multi-GPU desktop system (with one or two NVIDIA RTX 2080 GPU, respectively). Moreover, the code takes only 3.52 and 2.44 h if run on one or two NVIDIA V100 GPU, respectively. The relevant speedups reached (up to ∼38× in the single-GPU version, and ∼55× in the multi-GPU) clearly demonstrate that the GPU technology is highly suitable for realistic large network simulations.

Keywords: computational modeling, neuroscience, granular layer simulator, graphics processing unit, high performance computing, parallel processing

OPEN ACCESS

Edited by:
Padraig Gleeson, University College London, United Kingdom

Reviewed by:
James Courtney Knight, University of Sussex, United Kingdom
Federico Giove, Centro Fermi-Museo Storico della Fisica e Centro Studi e Ricerche Enrico Fermi, Italy

***Correspondence:**
Emanuele Torti
emanuele.torti@unipv.it

Received: *18 November 2020*
Accepted: *17 February 2021*
Published: *16 March 2021*

Citation:
Florimbi G, Torti E, Masoli S, D'Angelo E and Leporati F (2021) Granular layEr Simulator: Design and Multi-GPU Simulation of the Cerebellar Granular Layer. Front. Comput. Neurosci. 15:630795. doi: 10.3389/fncom.2021.630795

INTRODUCTION

The challenge to understand, reproduce and simulate the human brain activities needs more and more High-Performance Computing (HPC) support, in particular, where heterogeneous elements, described by complex mathematical models, have to be simulated as fast as possible (Bouchard et al., 2016). For example, computational modeling in neuroscience has to perform large-scale simulations to reproduce complex physiological behaviors of neuronal networks. Among the

different aspects that ask for HPC in neuroscience, some have a more relevant impact as the network dimension, i.e., the number of neurons, and the connections to model. Nowadays, several research groups work on reproducing the functionalities of very large areas of the brain (Beyeler et al., 2014; Cremonesi and Schürmann, 2020). To this aim, they need multicore and/or manycore technologies capable of reducing the processing time and of ensuring the power, memory, and storage capabilities offered by HPC solutions (Fidjeland et al., 2013). Another aspect to consider is the model to use for the neuron representation and the detailed morphologies introduced in the network. Starting from the simple Leaky Integrate and Fire (LIF) model up to the more complex Hodgkin Huxley (HH) one, all the mathematical representations are characterized by a variable number of differential equations, which strongly increases the computational load of the simulations (Izhikevich, 2004). Moreover, the detailed morphologies provide information about how to perform the signal exchange between neurons in the network and how the potential evolves inside the single element. If not properly managed, those aspects can easily increase the computational load of the simulation. A further issue to consider is the duration of the neuronal activity to reproduce. Particular attention should be given to the time integration step that directly determines the number of times that the differential equations have to be solved.

Recently, the number of research neuroscience groups using multicore and/or manycore architectures has indeed increased due to the need of high computing power to simulate complex and realistic neuronal models. Among HPC architectures, the Graphics Processing Unit (GPU) technology is becoming one of the most popular since it is capable of processing in parallel the neuronal activity of a huge number of cells. One important aspect that can make the GPUs useful in this research field is that they can be hosted in desktop systems as well as in supercomputers.

In this work, authors present the Granular layEr Simulator (GES), a system capable of reconstructing in detail the granular layer network of the cerebellum (a major cortical structure of the vertebrate brain) in a 3D space and of reproducing its neuronal activity. This code has been written in C language and using the OpenMP API together with the CUDA framework to efficiently exploit desktop architectures characterized by multicore CPUs connected to single and multi-GPUs. The simulator consists of three modules: the *network design* displaces the neurons and the glomeruli in a volume and connects them considering axonal/dendritic field geometries, oriented in the 3D space, and following the convergence/divergence rates that, to the best of the authors' knowledge, are the most relevant in the literature. Once the neurons displacement and connections have been elaborated, the *simulation* module can reproduce the network neuronal activity. The neurons are modeled using the Hodgkin and Huxley representation (Hodgkin and Huxley, 1990). The third module is the *graphical interface* that allows the user to generate several network configurations, to simulate and to graphically visualize them in a 3D space. In fact, the aim is to build a parametric network that can reproduce different configurations only by changing the values of suitable variables. This parametric system is also scalable allowing the reproduction of networks with different sizes.

Section Materials and Methods presents the granular layer model and the optimized codes developed for the network reconstruction. Moreover, the serial and parallel codes of the simulation module will be detailed. Then, the graphical user interface will be described. Section Results and Discussion presents the results and an exhaustive analysis of the neurons placement with relative connections, of the processing times and of the system memory occupancy. A comparison with the state of the art and possible future works are shown. Finally, section Conclusions draws the main conclusions and the possible future directions of the work.

MATERIALS AND METHODS

In the previous works, authors developed the simulators for the single cells hosted in the cerebellar granular layer. The neurons have been modeled exploiting the Hodgkin and Huxley representation and their simulators have been developed targeting parallel devices. This activity had a crucial importance to validate the neuronal behaviors and to evaluate the best parallel technologies to use in the network implementation (Florimbi et al., 2016, 2019).

The development of the GES required three main phases that represent a further step-on in the conducted research. At first, an efficient algorithm to reconstruct the granular layer network in a 3D space was developed. It places and connects different types of neurons as realistically as possible, taking into account their cellular morphology and their axonal/dendritic field geometries, oriented in the 3D space. The cells connections are the input of the second step, which concerns the neuronal activity reproduction of the layer. The network activity simulation has been carried out on one of the most recent multi-GPUs systems, in order to reduce the computational time. In the last phase of the work, a graphical interface has been developed to visualize the displacement, the connections and the activation patterns of the different neurons providing to the scientists a useful and easy tool that allows to setup the simulation and to monitor its own behavior.

Overview of the Cerebellar Granular Layer Model

The Neurons Models

The Hodgkin-Huxley (HH) model (Hodgkin and Huxley, 1990) is one of the most accurate and complex representations to reproduce the neuronal activity. The model describes the cellular membrane as a capacitor C_m since it keeps the ions separated on its sides. The capacitor is connected in parallel with different branches, each one including a resistor and a voltage generator connected in series. The resistors stand for the ionic channels, contained in the membrane, which allow the ions crossing. The voltage generators represent the active transport mechanisms that characterized the cellular activity. The current I flowing

through the membrane is described as in Equation (1):

$$I = C_m \frac{dV_m}{dt} + I_{syn} + I_{ions} \tag{1}$$

where V_m is the membrane potential, I_{syn} the synaptic current, and I_{ions} is the sum of the ionic currents. Each ionic current (I_{ion}) is defined as the product between the channel conductance g_{ion} and the difference between the membrane potential V_m and the equilibrium potential of the specific ion E_{ion} (Equation 2):

$$I_{ion} = g_{ion}(V_m - E_{ion}) \tag{2}$$

The ionic channels are characterized by the presence of *gating particles*, whose position inside the channel allows its opening or closure. The HH model reproduces how they dynamically affect the channel conductance as in Equation (3):

$$g_{ion} = \overline{g}_{ion} \times x_{ion}^z \times y_{ion}^k \tag{3}$$

where $\overline{g}_{ion}$is the maximum conductance of the channel, x_{ion} and y_{ion} are the probabilities that the gating particles occupy a certain position in the membrane. z and k represent the number of activation and inactivation particles for each channel (Florimbi et al., 2017). The probability of a particle of being in a permissive state depends from two coefficients α_n and β_n related to the velocity of transition (D'Angelo et al., 2001). The relation is given by:

$$\frac{dn}{dt} = \alpha_n (1 - n) - \beta_n n \tag{4}$$

where the probability of being in a permissive state is n. Equation (4) can be simplified using these two relations:

$$n_\infty = \frac{\alpha_n}{\alpha_n + \beta_n} \tag{5}$$

$$\tau_n = \frac{1}{\alpha_n + \beta_n} \tag{6}$$

where n_∞ and τ_n are the stationary part and the activation time of the channel. Equation (4) can then be rewritten as:

$$\frac{dn}{dt} = \frac{n_\infty - n}{\tau_n} \tag{7}$$

which is solved by:

$$n(t) = n_\infty - (n_\infty - n_0) e^{-\frac{t}{\tau_n}} \tag{8}$$

where n_0 is the initial value of n.

The final model is obtained considering the gating particles for each ionic channel and including these relations in Equation (1):

$$I = C_m \frac{dV_m}{dt} + I_{syn} + \overline{g}_k n^4 (V_m - V_k) + \overline{g}_{Na} m^3 h (V_m - V_{Na}) + \overline{g}_L (V_m - V_L) \tag{9}$$

where n is the gating particle of the potassium channel and m and h are the ones of the sodium channel.

Concerning the soma of the granule (GRCs), the model described in D'Angelo et al. (2001), takes into account some particular mechanisms related to ions. The sodium channel is represented by three currents: a fast Na^+ (I_{Na-f}), a persistent Na^+ (I_{Na-p}), and a resurgent Na^+ (I_{Na-r}) currents. The potassium channel is represented by five currents that reproduce different dynamics: a current for rectified delayed channels (I_{K-V}), one depending on intracellular calcium concentration (I_{K-Ca}), one for inward rectified channels (I_{K-IR}), one for type-A channels (I_{K-A}) and a current for slow kinetic channels (I_{K-slow}). The reversal or Nernst potential of the sodium and of the potassium channels are constant during the neuronal activity evaluation. The calcium channel is characterized by a variable intracellular calcium concentration. The Ca^{2+} dynamic is described by the following differential equation (Florimbi et al., 2016) (Equation 10):

$$\frac{d[Ca^{2+}]}{dt} = \frac{-I_{Ca}}{2FAd} - (\beta_{Ca}([Ca^{2+}] - [Ca^{2+}]_0)) \tag{10}$$

where d is the depth of the vesicle linked to the cellular membrane, whose area is indicated with A. β_{Ca} determines the calcium ions leakage from the cell, F is the Faraday constant, $[Ca^{2+}]_0$ is the calcium concentration at rest. Once $[Ca^{2+}]$ is computed, E_{ca} can be determined and used in the calcium current evaluation. The kinetic of these ionic channels is described using the HH model and the gating particle mechanism described above. Each channel is characterized by a different number of activation and inactivation particles.

Also for the Golgi (GOC), the model adopted to reproduce its activity considers several ionic currents (Solinas et al., 2008; D'Angelo et al., 2013). The ones that reproduce the regular pacemaking of the cell are the sodium persistent current (I_{Na-p}), the h current (I_h), the SK-type small conductance calcium-dependent potassium current (I_{K-AHP}) and the slow M-like potassium current (I_{K-SLOW}). These currents together with the sodium resurgent one (I_{Na-r}) and the A-current (I_{K-A}) regulate the response frequency and delay the fast phase of a spike, which is present after the hyperpolarization. The I_{Ca-LVA} reinforces the rebound depolarization. The rebound excitation is caused by the currents I_{Ca-LVA} and I_h. All these currents are described by the gating particles model explained before. Instead, the I_{K-AHP} current is simulated with a Markov gating scheme characterized by six states, four closed and two open (Solinas et al., 2008; Florimbi et al., 2019).

The Synapses Models

The term I_{syn} in Equation (4) represents the synaptic currents, i.e., the currents injected to the cell by their connected neurons through excitatory and inhibitory synapses. To compute the synaptic current, it is important to provide a model that reproduces the presynaptic and the post-synaptic terminals. In the first case, a three-state kinetic scheme has to be solved to compute the amount of neurotransmitter (T) released by the presynaptic terminal (Nieus et al., 2006). This neurotransmitter

reaches the receptors hosted in the post-synaptic terminal, reproduced by a model that allows to compute the currents that flow in the receptors channels. The excitatory synapses is characterized by the N-methyl-D-aspartate (NMDA) and the α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid (AMPA) receptors in the post-synaptic terminal (Nieus et al., 2006), while the inhibitory synapses present the gamma-Aminobutyric acid (GABA) one (Nieus et al., 2014). The current flowing in each receptor channel is computed solving a kinetic scheme of first-order reactions, with five (NMDA), three (AMPA) and eight (GABA) states. A detailed description of the dynamic of these receptors can be found in Nieus et al. (2006) for NMDA and AMPA, and in Nieus et al. (2014) for GABA.

Concerning AMPA receptors, there are three possible channel states: open (O), closed (C), and desensitized (D). Therefore, the current contribution is given by:

$$I_{AMPA} = g_{\max,AMPA}\left(V_m - V_{rev,AMPA}\right) O(T) \tag{11}$$

where $g_{\max,AMPA}$ is the maximum conductance of the AMPA receptor (1,200 pS), V_m is the membrane potential, $V_{rev,AMPA}$ is the ionic reversal potential and $O(T)$ is the probability of being in the open state, which depends from the concentration of neurotransmitter T.

The NMDA receptor is more complex since it has five possible states: three closed states (C1, C2, and C3), an open state (O), and a desensitized state (D). The current contribution is given by:

$$I_{NMDA} = g_{\max,NMDA}\left(V_m - V_{rev,NMDA}\right) O(T)B \tag{12}$$

where $g_{\max,NMDA}$ is the maximum conductance of the NMDA receptor (18,800 pS), V_m is the membrane potential, $V_{rev,NMDA}$ is the ionic reversal potential, $O(T)$ is the probability of being in the open state and B is a term to take into account the concentration of the Mg^{2+} ion.

The GABA inhibitory receptors are made up of two parts, called α1-GABA and α6-GABA. These two parts can be modeled using the same Markov chain, which is made up of two open states (OA1 and OA2), three closed states (C, CA1, and CA2) and three desensitized states (DA1, DA2, and DA2f).

The current of each part of the GABA receptor is given by

$$I_{GABA} = g_{\max,GABA}(V_m - V_{rev})(OA1\,(T) + OA2(T)) \tag{13}$$

where $g_{\max,GABA}$ is the maximum conductance (918 pS for α1-GABA and 132 pS for α6-GABA), V_m is the membrane potential, $V_{rev,GABA}$ is the ionic reversal potential and the sum OA1(T)+OA2(T) represents the probability of being in an open state.

Finally, I_{syn} is given by receptor currents (Equation 14):

$$I_{syn} = I_{NMDA} + I_{AMPA} + I_{GABA} \tag{14}$$

A deeper description of the GRC, GOC, and synaptic models can be found in Florimbi et al. (2016, 2019).

The Network Connectivity

The cerebellar cortex is composed of three layers (the *granular*, the *Purkinje*, and the *molecular layers*), each one including different types of neurons. The granular layer hosts GRC and GOC cells that connect their dendrites and axons in structures called glomeruli (GLOs), reached also by the mossy fibers (MFs). These elements are connected in the so called *feedforward* and *feedback* loops (**Figures 1A,B**) (Mapelli et al., 2014). In the first case, the MFs excite the GRCs and GOCs dendrites and these latter inhibit the GRCs; in the second case, the MFs excite the GRCs and, then, the parallel fibers (PFs) excite the GOCs that inhibit the GRCs.

All the elements (GOC, GRC, GLO, MF, and PF) are connected following convergence/divergence rules present in literature. According to Solinas et al. (2010) and D'Angelo et al. (2016), the convergence rate between GLOs and GRCs is 4:1, which means that 3–5 GRC's dendrites are connected to each GLO. The GRCs dendrites cannot reach GLOs located more than 40 μm away (the mean dendritic length is 13.6 μm) and a single GRC cannot send more than one dendrite into the same GLO (D'Angelo et al., 2013). Moreover, each GRC must project its dendrites to four different GLOs (Solinas et al., 2010). Each GLO has about 50 connections available for the GRCs dendrites since the divergence rule between GLOs and GRCs dendrites is 1:53 (D'Angelo et al., 2016).

The GOCs axons are placed in the granular layer spreading longitudinally. They enter in the GLOs to inhibit the GRCs. The convergence rate between GLOs and the GOCs is 50:1 (Solinas et al., 2010; D'Angelo et al., 2016). A GOC axon can enter only in GLOs without GRCs in common: a GRC is not inhibited twice by the same GOC (Solinas et al., 2010). Moreover, each GOC axon can reach and inhibit a maximum of 40 different GLOs (i.e., reaching ∼2,000 GRCs following the ratio GOCs:GRCs equal to 1:430) (Korbo et al., 1993; D'Angelo et al., 2013).

The GOCs basal dendrites spread around the soma on the same plane. They reach the GLOs where they make excitatory synapses with the MFs. Each GOC receives excitatory inputs from about 40 MFs on basal dendrites (Kanichay and Silver, 2008; D'Angelo et al., 2013).

The GRCs axons cross vertically the cerebellar Purkinje layer (i.e., ascending axon), which contains the Purkinje soma, and reach the molecular layer where it branches into PFs running transversally. It has been observed that GRCs form their connections through PFs and also along the ascending axon (D'Angelo et al., 2013). Moreover, D'Angelo et al. (2016) report that the convergence rate between the ascending axon and the GOCs is 400:1 and between the PFs and the GOCs is 1,000:1. GOCs are connected through gap-junctions present in the basal and apical dendrites (D'Angelo et al., 2013, 2016).

The Granular layEr Simulator

The basic idea followed in the network development has been to construct a *non-fixed* parametric structure. This means that even though the network is defined by specific structural and connections rules, it is still possible to modify its size, reproducing different configurations. The volume that will be reproduced in this work is 600 (*length*) × 150 (*height*) × 1,200

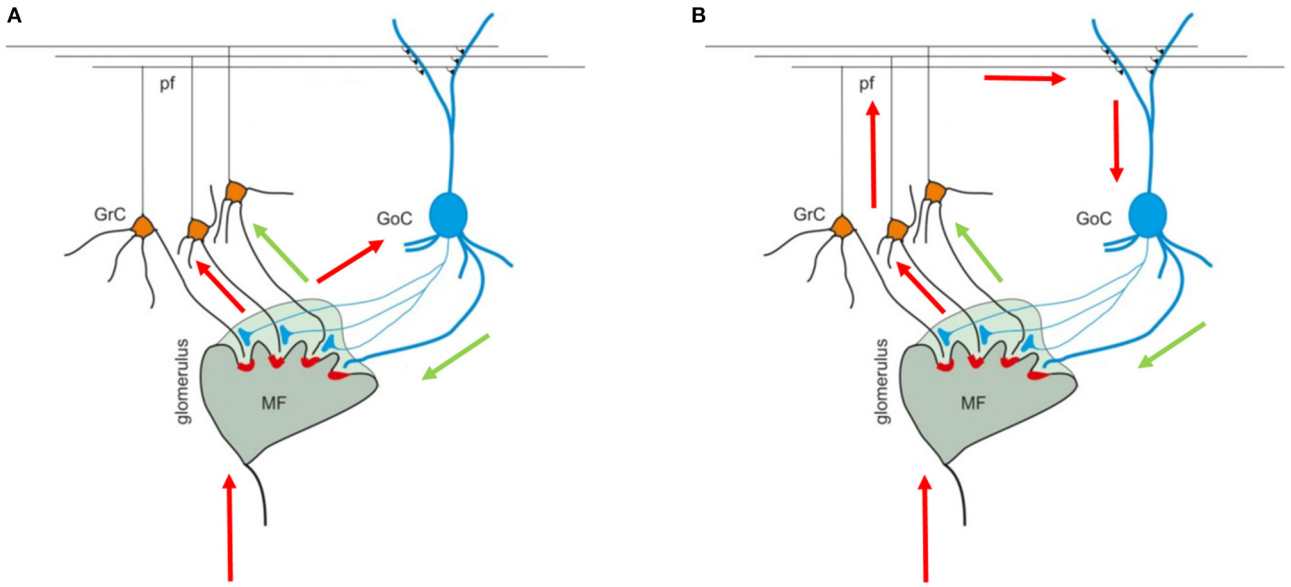


FIGURE 1 | Cerebellar granular layer. **(A)** The granular layer circuit receives the input from the mossy fibers (MFs) that reach the glomeruli (GLOs). Here, they reach and excite the Golgi cells (GOCs) and granules (GRCs) dendrites. Once the GRCs are stimulated, the signal travels along the GRCs ascending axon and parallel fibers (PFs) and, then, can reach the GOCs apical dendrites (feedback loop). **(B)** On the other hand, the MFs signals reach the GOCs cells that inhibit the GRCs (feedforward loop). The black arrows indicate the direction of the signals in the loops. The image is taken from Mapelli et al. (2014).

TABLE 1 | GOCs, GRCs, and GLOs density values and the number of elements that a volume of 600 × 150 × 1,200 μm^3 can host.

Cell or element ID	Density (mm^{-3})	Cells (or element) number	Soma diameter [μm]
GOCs	9,000 (Korbo et al., 1993)	972	15 (Dieudonné, 1998; Houston et al., 2017)
GRCs	4,000,000 (Korbo et al., 1993)	432,000	5 (Solinas et al., 2010)
GLOs	300,000 (Korbo et al., 1993)	32,400	5 (Rossi and Hamann, 1998)

Moreover, soma diameters are considered.

(*depth*) μm^3. This flexibility should be intended only in terms of parameters variability rather than new constraints introduction. The serial algorithm developed to reconstruct the granular layer is written in C language, which allows direct and efficient dynamic memory management.

Network Design

The network design module performs two main operations: the elements displacement and connection in a 3D volume.

In this case, the serial algorithm starts computing the number of GOCs, GRCs, and GLOs, referring to typical rat densities as shown in **Table 1**. It also shows the number of elements considered in the network configuration under study. Finally, since the neurons soma is modeled as a sphere, the correspondent diameter is reported.

Once the elements number is known, the algorithm computes the GOCs, GRCs and GLOs coordinates that are stored in three arrays (*c_goc*, *c_grc,* and *c_glo,* respectively), dynamically allocated through the *malloc* routine in the initial phase of the code. The algorithm inserts the elements in the space in a *partially random* way, considering specific physiological requirements (Korbo et al., 1993; Dieudonné, 1998; Rossi and Hamann, 1998; Solinas et al., 2010). The height of the volume (*z-axis*) is divided into several layers, whose number is related to the dimensions of the GOCs since they are the first type of elements introduced in the volume. Each layer includes several boxes, organized in rows (shown by the arrows in **Figure 2**), which dimensions are related to the GOCs ones. The basis of the box is a square, whose side is equal to the GOC diameter. On the other hand, the height of the box is higher than the diameter so that the algorithm can randomly compute the *z* coordinate of the cell inside the box. In this way, all the GOCs inside a layer are not placed at the same height. The algorithm has to insert a defined number of GOCs in each layer, selecting a box for each cell. The algorithm chooses the suitable box following morphological constraints (i.e., dendrites length and depth) and avoiding boxes already occupied by other neurons. When a cell is placed in a box, its *x* and *y* coordinates are defined. Once all the GOCs have been placed in the volume, the algorithm inserts the GLOs and, then, the GRCs, both represented by spheres with 5 μm diameter (**Table 1**). The strategy adopted to place GLOs and GRCs is the same as for GOCs. During the GLOs and GRCs displacement, a further constraint is added to avoid that these elements overlap the GOCs. Finally, the correspondent coordinates are written in the *c_goc*, *c_glo*, and *c_grc* arrays.

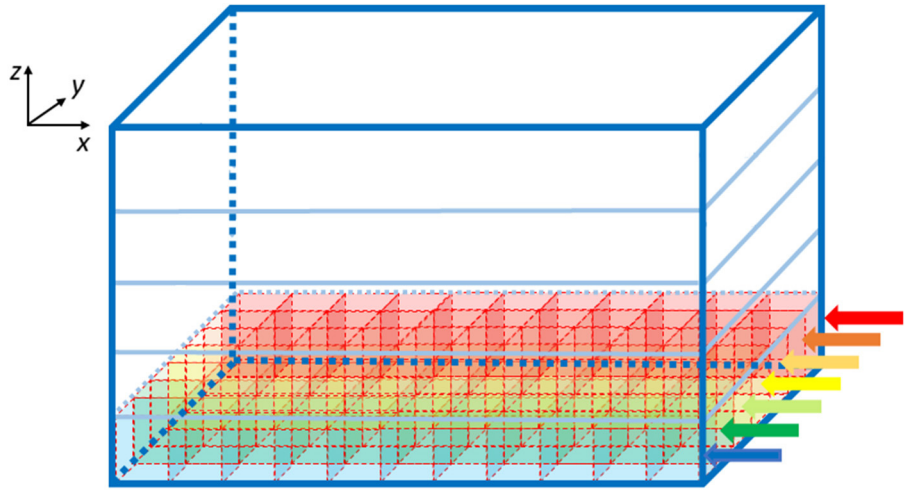


FIGURE 2 | Volume division for the displacement of the GOCs. The volume is divided into z-layers (five in this image). Each z-layer (x-y plane) is divided into rows along the y-axis (indicated with the arrows), where rectangular parallelepipeds are placed to host the cells. The algorithm starts placing the neurons from the blue row to the red one. The procedure is repeated for each z-layer.

Once the elements have been placed in the volume, the algorithm starts to connect them, generating the *connection matrices*. They are linear arrays containing the information on how elements are connected following convergence/divergence rules. The *connection matrices* reproducing the feedforward and feedback loops (D'Angelo et al., 2016) contain the links among the following elements (the name of the *connection matrices* is reported between brackets):

- GRCs and GLOs (*link_grc_glo*);
- GOCs axon and GLOs (*link_goca_glo*);
- GOCs basal dendrites and GLOs (*link_gocdb_glom*);
- GRCs (ascending axon and PFs) and GOCs (*aa_goc_link*, *pf_goc_link*);
- MFs and GLOs (*mf_glom_clustering*);
- GOCs and GOCs (*gap_junction*).

For each type of connection, the authors developed a suitable algorithm capable of connecting the elements following the morphological rules and the convergence/divergence ratios.

As an example, the algorithm that links the GRCs to the GOCs through ascending axon and PFs is detailed in the following.

As said in section Overview of the Cerebellar Granular Layer Model, the GRCs axons cross vertically the cerebellar Purkinje layer and reach the molecular one where it branches into PFs running transversally, i.e., along the *y*-axis (**Figure 3**). Even if this work aims to reproduce the granular layer, it is important to take into account these connection schemes, in order to reproduce the feedback and feedforward loops, simulating all the connections between neurons. As previously said, the GRCs form their connections with GOCs through PFs and also along the ascending axon with a convergence rate of 1,000:1 and 400:1, respectively (D'Angelo et al., 2016) (D'Angelo et al., 2013). Firstly, the algorithm computes the connections between the ascending axons and the GOCs. When the algorithm has to find those GRCs to connect to, it builds an elliptical cylinder (in red in **Figure 3**) around the GOC soma, whose major axis is the maximum length of the apical dendrites, while the minor axis is given by their depth. For each GOC, the algorithm selects 400 GRCs inside the red cylinder avoiding the area under the GOC soma, denoted by a blue cylinder in **Figure 3**, where it is less probable to have connections. The algorithm randomly selects a GRC among the 400, and checks if it is inside the red cylinder. If yes, the connection is performed (AA CONNECTION in **Figure 3**) and the GRC index is stored in the linear matrix *aa_goc_link*. Considering the PFs, the GOC receives 400 connections through the PFs of local GRCs and 1,200 distal connections (D'Angelo et al., 2013). The algorithm selects the GOCs to link to the GRCs, checking if in the red cylinder there are GRCs, whose PF crosses the apical dendrites area (as in the purple case in **Figure 3**): if yes, the connection is made (LOCAL PF CONNECTION). If the PF that crosses the apical dendrite area belongs to a GRC far from the GOC soma (orange parallel fiber), a distal connection (DISTAL PF CONNECTION) is implemented. These connections are stored in the linear matrix *pf_goc_link*.

Another interesting part of the system performs the connections between MFs and GLOs. In this case, authors developed a custom clustering algorithm to meet physiological constraints. Authors in Sultan and Heck (2003) described how the MFs branch in the cerebellum. They form clusters of presynaptic enlargements called *rosettes*, which represent the presynaptic part in the GLOs. Each MF can form multiple clusters of rosettes arranged with a characteristic spatial organization. In particular, authors in Sultan and Heck (2003) demonstrate that the clusters belonging to the same MF are separated from each other. For this reason, given the dimension of the network

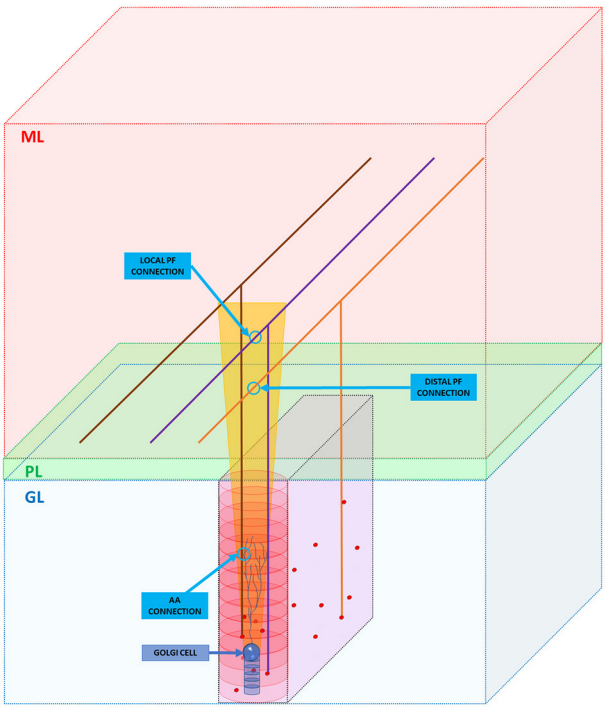


FIGURE 3 | GRC ascending axon and PF connections with GOC. The figure shows the different layers of the cerebellum cortex: granular layer (GL), Purkinje layer (PL), and molecular layer (ML). In the GL, GRC soma, and GOC soma are represented with red and blue spheres, respectively. The yellow trapezoid represents a schematization of the area occupied by the GOC apical dendrites (partially shown inside the area). The image shows three examples of connection. Firstly, the algorithm connects the GOC apical dendrites with the GRC ascending axon (AA CONNECTION). Then, it performs the connections through PFs (LOCAL PF CONNECTION and DISTAL PF CONNECTION). Notice that in the image, the sizes do not scale proportionally to improve the graphical view.

considered in this work, it is not reasonable to find two clusters belonging to the same MF.

They demonstrate that each cluster accumulates 7.7 (±4.1) rosettes and, thus, GLOs. Finally, always in Sultan and Heck (2003), the authors show that elements in a cluster are located within 350 μm from the cluster mean location. Taking into account this physiological information, a clustering algorithm capable of generating clusters with a defined dimension is needed. The elements inside the cluster must comply with the distance constraint. Authors developed an algorithm whose goal is to divide several points (inserted in a 3D volume, characterized by spatial coordinates) into N clusters with a fixed, properly set dimension. The number of clusters N is one of the algorithm inputs. Let us assume that the clusters dimension is NUM_POINT_CLUSTER (±DELTA): in this case, NUM_POINT_CLUSTER is set to 8 and DELTA is set to 4 according to Sultan and Heck (2003). In the initialization phase, the algorithm computes the centroids in a pseudo-random way. Once the coordinates are initialized, the algorithm computes the Euclidean distances between the GLOs and the centroids to identify the nearest centroid for each GLO. The algorithm computes the K-nearest centroids for each GLO (in this work, K = 100) and sorts them in ascending order, on the basis of the GLO-centroid distance. At the end, all these data are stored in the *data_clusters* structure, including these fields:

- GLO ID (ID_{glo});
- Nearest centroid ID ($ID_{primary}$);
- Distance GLO ID_{glo} and centroid $ID_{primary}$;
- Structure that contains ID and distances of the K-nearest centroids of the GLO ID_{glo}.

Each GLO is temporarily assigned to its nearest cluster. At this point, the algorithm computes the dimensions of each cluster (i.e., how many GLOs have that specific centroid as the nearest). If the cluster dimension is out of the range NUM_POINT_CLUSTER (± DELTA), its ID is stored in one of the following arrays:

- *buffer_low*, which contains the IDs of the centroids with a dimension lower than NUM_POINT_CLUSTER – DELTA;
- *buffer_high*, which contains the IDs of the centroids with a dimension higher than NUM_POINT_CLUSTER + DELTA.

The next step aims to reduce the centroids present in the *buffer_high,* so that its dimension can be within the range presented above. A *for* loop removes the *extra* elements in the clusters present in the *buffer_high* array and assigns them to other clusters with lower dimensions. The algorithm tries to select a cluster that has a dimension lower than NUM_POINT_CLUSTER – DELTA and, at the same time, is one of the nearest for the GLO that will be moved. Otherwise, it searches for another cluster in the volume giving priority to the ones with a dimension lower than NUM_POINT_CLUSTER + (DELTA/2).

Serial and Parallel Network Simulation

This section illustrates the serial and parallel codes developed to simulate the granular layer activity. In previous works (Florimbi et al., 2016, 2019), authors developed the GOCs and GRCs simulators where the neurons were not connected and their activities were evaluated in parallel. These works validated the GOCs and GRCs behaviors reproduced by the simulators. Moreover, this phase was of crucial importance to evaluate the GPU technology in this kind of applications. The significant speed-up obtained comparing the serial and parallel simulators demonstrates that the GPU is a suitable technology for neuronal simulations. For this reason, authors developed also a CUDA version of the granular layer simulator to exploit single and multi-GPU systems.

Once the *connection matrices* have been generated in the network design step, the GRC and GOC simulators can be integrated to reproduce the activity of the cerebellar granular layer.

In the *Initialization* phase, all the variables related to all the cells and their synapses are declared and initialized in a structure called *grc_cell* for the GRCs, and *goc_cell* for the GOCs. Moreover, each structure contains two further structures (one for the excitatory and one for the inhibitory connections), the

Algorithm 1 | Network simulation.

```
1 Connection matrices reading;
2 Initialization;
3 MF signal Initialization;
4 for t←0 to t_end
5       for n←0 to n_goc
6           GOC Synaptic activity computation;
7           GOC Cellular activity computation (solve Equation 2 for each ion);
8           GOC sum currents and conductances;
9           GOC membrane potential update (solve Equation 9);
10          Send signals to granule cells;
11      end
12      Gap junctions currents update;
13      for n←0 to n_grc
14          GRC Synaptic activity computation;
15          GRC Cellular activity computation (solve Equation 2 for each ion);
16          GRC sum currents and conductances;
17          GRC membrane potential update (solve Equation 9);
18          Send signals to Golgi cells;
19      end
20 end
21 Write results;
22 end
```

Algorithm 2 | Synaptic activity computation.

```
1  for t ← 1 to t_end do
2    for i ← 1 to n_goc do
3      for j ← 1 to buffers do
4        if spike then
5          compute neurotransmitter concentration;
6          remove the spike from the queue;
7        end
8      end
9      solve the AMPA, NMDA and GABA kinetic schemes;
10     compute the currents that flow in the receptors;
11     I_syn = I_AMPA + I_NMDA + I_GABA;
12       …
13   end
14 …
15 end
```

membrane potential V_m, the synaptic current I_{syn}, the ionic channel current I_{ion} and conductance g_{ion}, all the gating particles for each ionic channel, and the calcium Nernst potential. Each structure related to the connections contains an array storing the spikes that occur in the synapses. Then, in the *MF signal initialization* phase, the configuration of the simulation protocol, described in section Computational Results, has to be initialized, deciding how the MFs provide inputs to the network. At this point, the algorithm can start evaluating the network activity. For each t-th time step, the algorithm evaluates the synaptic and cellular activities of the GOCs and of the GRCs. The *for* loop that iterates over the simulation time is shown in **Algorithm 1** at line 4.

Inside this loop, lines 5 and 13 indicate the loops iterating on the GOCs and GRCs. For the t-th time step, the algorithm starts evaluating the GOCs synaptic activity (line 6), by solving the pre-synaptic and post-synaptic terminal models. In particular, the algorithm checks if some inputs have occurred in the GOCs basal and apical dendrites. To model this aspect, three buffers (*spike queues*) have been allocated in each *goc_cell* structure storing the inputs from the basal dendrites, the apical dendrites and the inhibitory synapses. The GOC dendrites are modeled as passive components characterized only by an axial resistance, which causes a delay in the signal transmission. This kind of representation allows analyzing, at each time step, if one or more inputs occurred in the different dendrites of the cell. **Algorithm 2** shows this process.

The *for* loops that iterate on the time steps and on the GOCs number are in lines 1–2 of **Algorithm 2**. Then, the algorithm checks if, in each buffer (**Algorithm 2**, line 3), a spike has occurred in the current time t (**Algorithm 2**, line 4). If the spike occurs, the algorithm solves the three-state kinetic scheme, cited above, to compute the neurotransmitter concentration (line 5). Once the input has been evaluated, the spike is removed from the spike queue. Once all the inputs have been evaluated, the algorithm solves the receptors schemes to compute the currents that flow in the channels present in AMPA, NMDA, and GABA receptors (lines 9–10). Finally, the synaptic current I_{syn} is updated and the *GOC Synaptic activity computation* phase ends. The flow of **Algorithm 1** continues evaluating the GOCs cellular activity (line 7): the value of the gating particles of each ionic channel is updated and, then, the channel conductances and currents are computed. At this point, the current and conductance contributions are summed and included in the membrane potential update. Moreover, in this last phase of the GOCs activity, the gap junctions currents are also considered. In the first iteration ($t = 0$), their value is initialized to zero since all the cells have the same membrane potential. Then, their values will be updated on the basis of the membrane potential difference between the cells linked through gap junctions. The last phase of the GOCs activity (*Send signals to granule cells*) manages the signals exchange between the GOCs and the GRCs. In fact, as said before, the signals travel along the GOCs axons that enter the GLOs, where the GRCs dendrites receive the signals from the GOC. At this point, the algorithm evaluates if the considered GOC generates a spike: if yes, the algorithm searches the GRCs linked to that cell through the connection matrix. Then, it stores the spike time in the suitable GRC spike queue. Here, two considerations are necessary: the first is that the GOC axon is modeled as a passive component. For this reason, the spike time is computed by adding a delay caused by the axonal resistance. The second is about how the GRCs dendrites have been modeled. As said before, the convergence rate between GLOs and GRCs is lower than the GOCs one (Solinas et al., 2010; D'Angelo et al., 2016), thus GRCs have 3–5 dendrites that enter the GLOs. For this reason, the four GRC dendrites have been represented with the same number of buffers for the excitatory and inhibitory connections.

Once all the GOCs signals have been stored in the suitable GRCs buffers, the *for* loop that iterates on the GOCs ends. At this point, the algorithm computed a new membrane potential value for all the GOCs and, then, the gap junctions currents are updated (*Gap junctions current update*). Then, for the same t-th iteration, the algorithm continues analyzing the GRCs synaptic and cellular activities. In the *for* loop that iterates on the GRCs, the algorithm

starts to evaluate the synaptic activity as shown in **Algorithm 2** for the GOCs: the only difference is that, in this case, each GRC has four buffers for the excitatory connections and four for the inhibitory ones. Clearly, the spikes generated by the GOCs in the t-th time step are not considered by the GRCs in this iteration. Once the membrane potentials have been updated, the action potentials generated by the GRCs are sent to the GOCs (*Send signals to Golgi cells* phase). In particular, as said before, the GOC receives excitation from the GRC through their ascending axons and/or through their PFs. Once all the GRC have been evaluated, the code can continue with the next time step ($t + \Delta t$).

The serial version of the granular layer network has been used as a basis for the development of two parallel codes, written in C/CUDA language. **Figure 4** shows the flow of the first parallel version, which runs in a single-GPU system.

The code starts on the host where the *connection matrices* are read (*Connection matrices reading*) and the variables are initialized. In addition to the initialization of the variables, in the

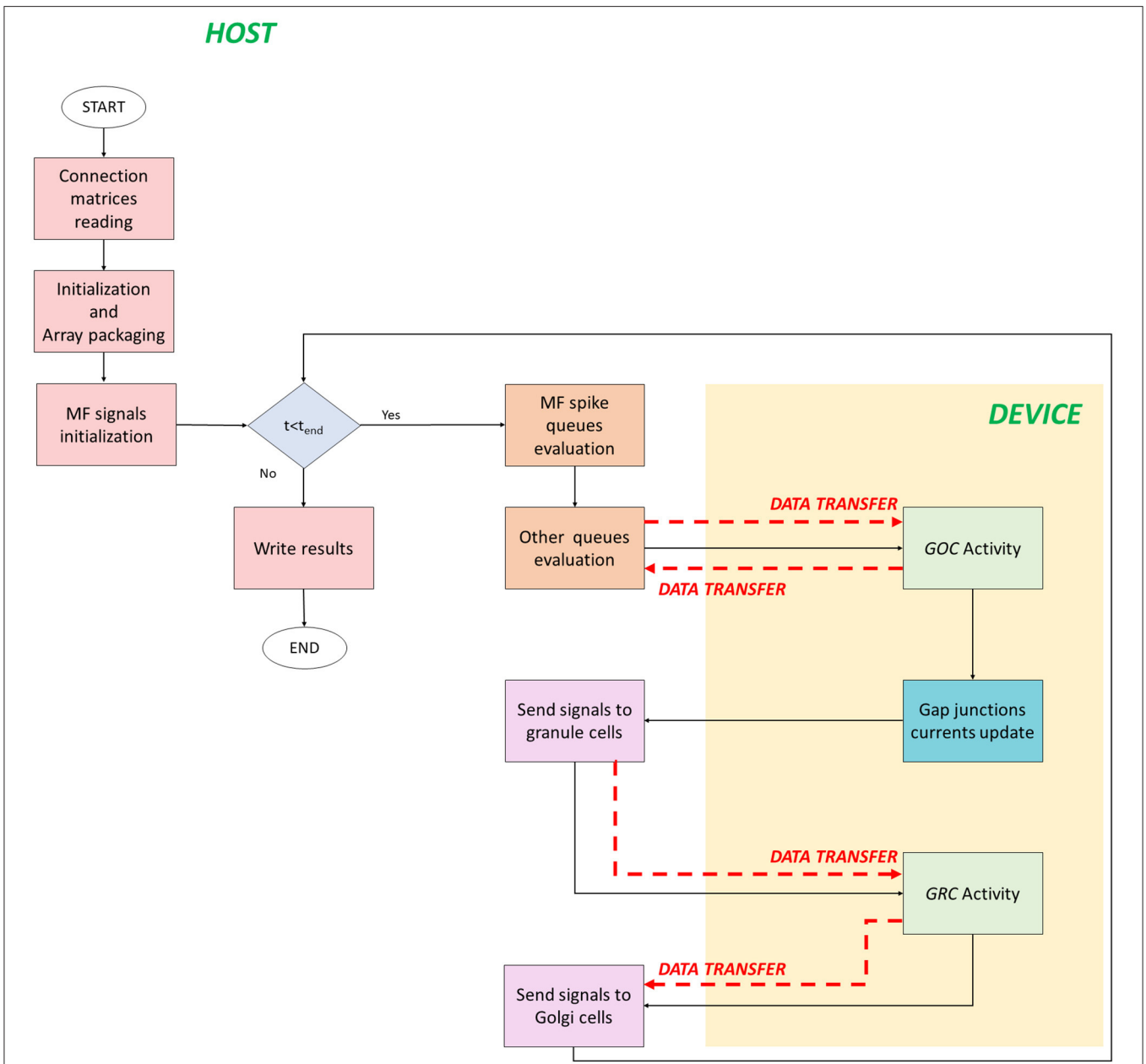


FIGURE 4 | Parallel flow for single-GPU system. The flow starts on the host where the *for* loop iterates on the time steps. The signals exchange is performed on the host, while the neurons activity computation is performed on the device (yellow box). The black arrows indicate the flow, while the red dashed arrows indicate the data transfers between host and device, and vice versa.

second phase, the algorithm prepares the data to be transferred from the host to the device memory. This phase is crucial to reach high performance and to reduce the computational times. In fact, if not properly managed, the data transfer could be the bottleneck of the process. In order to prevent this potential slow-down, all the data related to the cells have been stored at contiguous memory addresses, trying to minimize the bus activations during the transfers. The idea is to create a 1D array and to join the data according to their physiological meaning. In this phase (*Initialization and Array packaging*), the data related to the cellular activity and to the initialization of the pre-synaptic and post-synaptic models are prepared. On the other hand, the input signals will be set for the transfer after the configuration of the simulation protocol in the successive phase. In fact, in *MF signals initialization*, a protocol is chosen and the spike queues for each MF are generated. In this case, the authors chose to not transfer all the spikes queues of all the MFs from the host to the GPU global memory in order to not increase the memory usage. Once the protocol is generated, the *for* loop that iterates on the time steps begins. The *MF spike queues evaluation* phase has been introduced to properly manage the data transfers of the MFs queues. Since it is not efficient to transfer all the queues in the global memory, at each time step the algorithm evaluates if the MFs have generated an input signal. If yes, that input is stored in a temporary queue of the cell linked to that MF. Notice that each queue of all the MFs is evaluated in parallel exploiting a multicore strategy with the Application Programming Interface (API) OpenMP. It is a parallel programming model for shared-memory multiprocessors that provides a wide set of directives and strategies for the parallelization of loops and program sections through the *#pragma* directive. This statement is placed before the loop that should be parallelized. Moreover, in this directive, the variables or array are expressed as *private* to a single thread or *shared* among all the threads. In this case, at each current time, the MFs *for* loop is parallelized to simultaneously check the GRCs and GOCs linked to the MFs that have generated a spike. Finally, the algorithm generates two arrays of flags (one for GOCs, *goc_spikeMF* and one for the GRCs, *grc_spikeMF*) that contain the value 1 if the corresponding GOC or GRC received an input from the linked MFs. The flags are equal to zero if the corresponding cell is not stimulated. In this way, at each time step, the algorithm will transfer from the host to the device global memory, only two arrays of dimension $n_{goc} \times sizeof(int)$ and $n_{grc} \times sizeof(int)$, instead of all the MFs queues. In the O*ther queues evaluation* phase, other arrays are prepared for the transfer. In this code, the queues/buffers, already defined in the serial code, are present together with their related arrays of flags, exploited to transfer data to the global memory at each time step. For this reason, the GOCs have a buffer and an array of flags for the apical, basal and inhibitory connections (*goc_spikeAPIC, goc_spikeMF*, and *goc_spikeINH*, respectively). The GRCs have two flag arrays: one for the excitatory and one for the inhibitory connections. These two arrays are properly managed to transfer the signals stored in the four excitatory and inhibitory buffers present in the host. On the device global memory, space has been allocated to store the synaptic buffers and the gap junctions currents, i.e., an array whose dimension is $n_{goc} \times n_{gap} \times sizeof(float)$, where n_{gap} is the number of connections through gap junctions for each cell. All the *flag* arrays are transferred to the GPU global memory (red dashed arrows in **Figure 4**). At this point, the activity of all the GOCs, at the t-th time step, can be evaluated simultaneously on the device. This part (*GOC Activity* phase) represents a *kernel*, thus a function performed by parallel threads. For this reason, the device generates a number of threads equal to the number of GOCs, so that each thread can compute the activity of a specific GOC. Threads are organized in *blocks* that, in turn, constitute a *grid*. The block dimension (i.e., how many threads a block contains) is set as multiple of 32, according to the *warp* definition, to optimize the scheduling carried out by the NVIDIA Giga Thread scheduler (NVIDIA, 2019).

Before the kernel invocation, the code computes the grid dimension (i.e., the blocks number) as $[n_{goc}/n_{thread\_block}]$, where n_{goc} is the GOCs number (i.e., the total number of threads needed), and n_{thread_block} is the number of threads in a block, set to 32. If the remainder of the division is not equal to zero, the grid dimension is incremented by one. In this case, the last block contains more threads than necessary: this turns out in inactive threads assigned to the last block. Despite this, the inactive threads cannot be avoided because each block must have the same number of threads. Once all these parameters have been defined, the kernel can be activated. The first step of this phase is the data transfer from the global to the local memory of the device. All the threads within a block can access the same portion of the local memory and, for this reason, the goal is to copy the parts of the arrays that are needed by the threads in the block in this memory. In this way, the memory access latency diminishes. On the other hand, the local memory has a reduced size (dozens of KB) and, for this reason, not all the data can be transferred. In this way, only the arrays and variables that are the most used in the kernel are stored in the local memory. Moreover, these variables are accessed multiple times by all the threads in the block. When this first set of memory transfers has been concluded, all the threads start evaluating in parallel the GOCs activity. The evaluated phases are those shown for the serial code and represented in **Algorithm 1** lines 6–9. Once the GOCs activity is computed, all the updated data (i.e., membrane potential, kinetic scheme variables, gating particles, Nernst potentials, calcium concentration, and so on) are stored in the global memory because they will be used in the next iteration. Once this kernel has finished, another one starts computing the gap junctions currents. This kernel has the same number of threads and blocks previously defined since each thread evaluates the gap junctions connections of each GOC. The currents are stored in the device global memory and evaluated in the next GOCs activity evaluation. Once the kernels finished, two arrays are transferred from the device to host: the former stores the membrane potential of all the GOCs; the latter, called *flag_golgi_spike*, has size n_{goc} and, for each cell, stores a flag whose value is 1 if the corresponding GOC has generated a spike. The first arrays is then used to record the potentials in the mass memory of the system. As described above, this spike travels along the GOC axon, which enters in the GLOs, where the GRCs dendrites are hosted. At this point, exploiting the *connections matrices*, the code evaluates on the host which are the GRCs

linked to the GOCs that have generated a spike. For these GRCs, the spike time is computed and stored in their buffers for the inhibitory connections, since the GOCs provide an inhibition. In this way, all the GRCs inhibitory buffers are updated and, then, analyzed in order to see if they contain inputs that have to be evaluated by the GRCs in the current time *t*. This means that the flags of the array *grc_spikeINH* assume the value 1 if the corresponding GRC received an inhibitory signal to evaluate in *t*. Clearly, also in this case, the GRCs do not evaluate the GOCs spikes generated at the same time iteration. The flag array is then transferred from host to device, where a new kernel is invocated (*GRC activity* in **Figure 4**) to evaluate the activity of all the GRCs, for the *t*-th time step, simultaneously. In fact, the device generates a number of threads equal to the GRCs number. As for the GOCs activity, also in this case parts of data are transferred from the global to the local memory. Then, each thread computes the activity of one GRC, performing all the functions indicated in **Algorithm 1** lines 14–17. At the end, all the updated variables are transferred back from the local to the global memory, in order to be used in the next time iteration. For the GRCs, two arrays are also transferred from the device to the host: the first stores the neurons membrane potentials, and the second (*flag_grc_spike*) stores flags that indicate if the corresponding GRC has generated a spike. At this point, the code is processed on the host, where the array *flag_grc_spike* is analyzed. In this case, the code checks which are the GOCs linked to the GRCs (that have generated a spike) through the ascending axon and the PFs. For all these GOCs, the spike time is computed and stored in the apical connections buffer, which is then evaluated to update the *goc_spikeAPIC* array. The flags values will be set to 1 if the corresponding GOC is linked to a GRC that has sent a signal. The *Send signals to Golgi cells* phase is the last and the code can proceed with the next iteration.

The flow of the multi-GPU parallel version is shown in **Figure 5**. It starts on the host where data are initialized and prepared for the transfer. Despite the previous version, data have to be transferred from the host to two different devices. In fact, the neuronal activity evaluation is split between the two boards and, in particular, each one processes the activity of half of the neurons. Therefore, data related to the first half part of the cells are transferred to the *device 0* global memory, the others to the *device 1* one. For each simulation time step, the queues evaluation is the same done in the single-GPU version. The only difference is how data are prepared and split to be transferred to two devices. In order to invocate the *GOC Activity* kernel on two boards, firstly the kernel parameters have to be set. The number of threads in each kernel is given by $(n_{goc}/2)/n_{thread\_block}$, where $n_{thread_block}$is always set to 32. If the number of cells is not an even number, the threads number in one of the two devices is incremented by one. In order to perform two kernels simultaneously, CUDA provides the *streams* to concurrently execute and overlap kernels and data transfers (Rennich and NVIDIA, 2014). The streams carry out the transfers (indicated with red dashed lines in **Figure 5**) and activate the kernels on both the devices. All the previous considerations related to the transfers from the global to the local memory can be also done in this case for each board. Once the *GOC Activity* of all the GOCs is evaluated, the kernels end, and the flow is synchronized. At this point, the arrays that store the GOCs membrane potentials and flags (which indicate if each a GOC has generated a spike or not) are transferred from the device to the host memory. Moreover, a data transfer between the two boards is performed (light blue dashed line in **Figure 5**). In fact, the kernel related to the gap junctions currents computation is entirely performed on the device 0 since the code needs the membrane potential values of all the GOCs to update these currents. Indeed, each GOC is connected through gap junctions to other GOC which might not be located on the same device. For this reason, the membrane potentials evaluated by the device 1 are transferred to the device 0 through a *cudaMemcpyPeerAsync* function, which allows to directly transfer data between GPUs on the same PCI Express bus bypassing the CPU host memory.[1] Once the gap junctions currents have been updated and stored in the device 0 global memory, the flow returns on the host where the signals are exchanged and the flags arrays updated, as explained before. Then, the arrays are transferred on the two boards in order to activate the *GRC Activity* kernel on the two devices. As for the GOCs, each kernel evaluates on each board the activity of half part of the GRCs. Once the flow is synchronized, the arrays containing the GRCs membrane potentials and the flags, indicating the spike presence, are transferred from the devices to the host. At the end, the signals are exchanged between GRCs and GOCs and stored in the GOCs buffers for the apical connections. At this point, the flow can continue with the next iteration.

Graphical User Interface

The results of the design and of the network neuronal simulations can be graphically and quantitatively analyzed through a graphical interface, developed with the OpenGL API in order to achieve a GPU-accelerated rendering (Sellers et al., 2015). The main task of this graphical interface is to display the elements in a 3D volume, considering the spatial coordinates computed in the network design stage. In **Figure 6**, the main panel shows the granular layer network, with the dimension considered in this work (i.e., $600 \times 150 \times 1{,}200\ \mu m^3$). In particular, the GLOs are represented in green, the GRCs in red and the GOCs in blue. The interface is also useful to analyse the connections between GLOs and neurons, the MFs clustering and to watch the simulation output. As an example, two tasks have been shown in **Figure 7**. In particular, **Figure 7A** shows how the MFs branch in the cerebellum forming clusters of GLOs, represented with different colors. **Figure 7B** shows an example of connections between GRCs and GOCs through PFs. All the GRCs in red are the ones selected by the algorithm following the rules presented in section Network Design. Only the ascending axon and the PF of one GRC are represented to show how the PF crosses the space dedicated to the GOC apical dendrites, generating a connection. As far as

[1] *NVIDIA GPU Direct*. Available online at: https://developer.nvidia.com/gpudirect (accessed October 1, 2020).

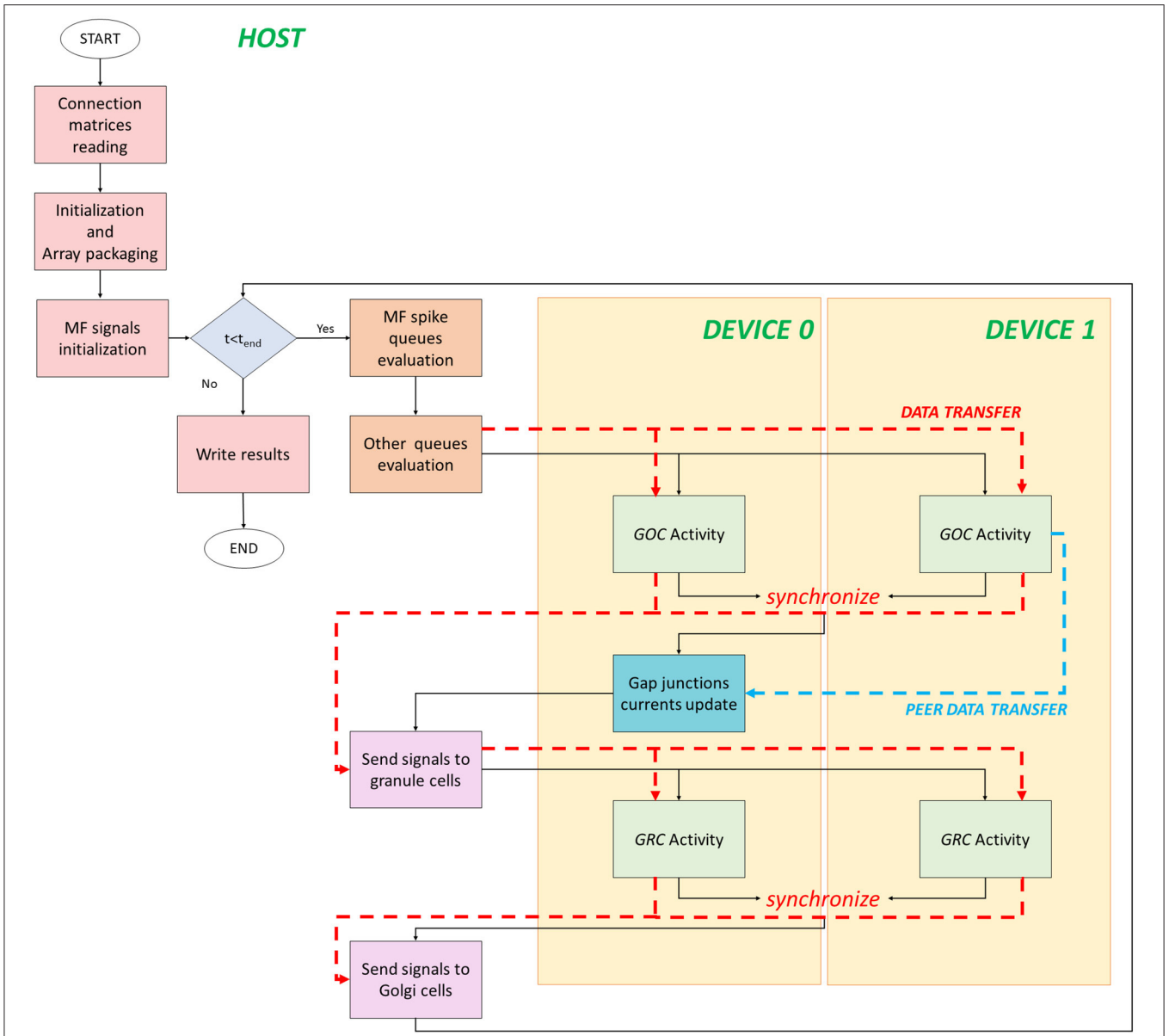


FIGURE 5 | Flow parallel version for the multi-GPUs system. The flow starts on the host, where the variables are initialized, and data are prepared for the transfers to the global memory of the devices. The GOC Activity and GRC Activity is managed by the two devices. The black lines indicate the flow, the red dashed lines indicated the host–device (and vice versa) transfers, and the blue dashed line indicates the transfer between devices.

the simulation tasks are concerned, the algorithm evaluates the membrane potential of each element in all the time steps of the simulation. In each time step, it changes the color of the elements that are generating a spike. In this way, the user can graphically analyse how neurons react to particular stimuli. In the center-surround simulation (**Figure 7C**), which will be described in section Computational Results, a particular technique is adopted to show the spiking neurons: at the beginning, all the neurons are not visible in the volume. As soon as a cell generates a spike, it will be shown with a color whose tone becomes darker as the spikes number increases.

RESULTS AND DISCUSSION

Neuron Placement and Connection Analysis

The first validation concerns the evaluation of how many elements can be correctly placed by the proposed network design algorithm. The analysis presented below is performed after running the design algorithm 20 times with different random seeds. Authors find that the algorithm always places the 100% of GOCs and GLOs, and 98% of the GRCs. This amount of not placed cells is negligible in terms of the correct evaluation

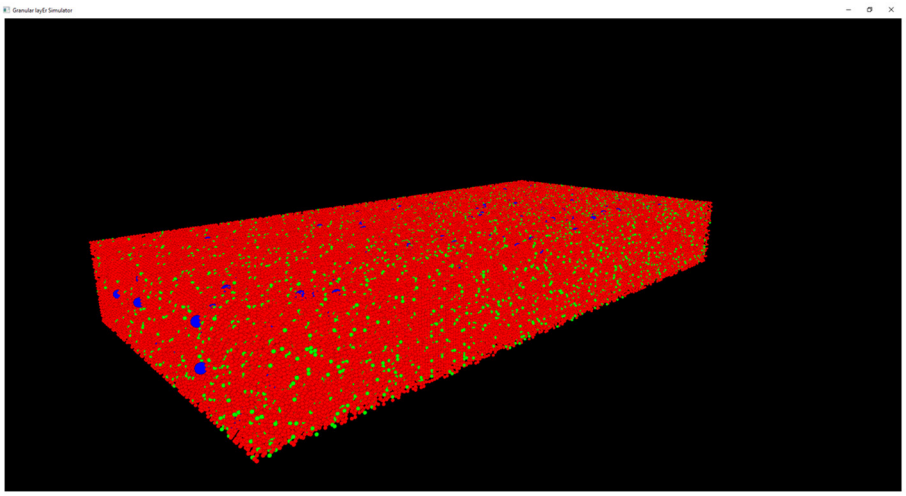


FIGURE 6 | Complete view of the network. Main panel where the complete network (with dimension 600 × 150 × 1,200 μm^3) is shown. Only the GRCs (red) and GOCs (blue) soma have been displayed. The GLOs have been represented as green spheres.

of the network activity. Moreover, authors carefully analyzed the *connection matrices* to evaluate the percentage of links established during the network design step. The analysis of the *link_grc_glo* matrix reveals that the algorithm completely fills the 50 available places of the 89.75% of the GLOs (**Figure 8A**). In only the 5.07% of cases, the GLOs are not linked to the GRCs dendrites and, in the other cases, the number of dendrites that reach the GLOs is between 1 and 49. Moreover, the evaluation of this connection matrix highlights that 82.30% of the GRCs sends its dendrites to four different GLOs, satisfying the convergence constraint (Solinas et al., 2010; D'Angelo et al., 2016) (**Figure 8B**). Moreover, this means that this percentage of GRCs is excited by four different MFs. In other cases, the GRCs are partially linked to 3 (3.90%), 2 (4.46%), and 1 (3.21%) GLOs/GLO. Only in the 6.20% of the cases, the GRCs are not connected to GLOs. The fact that not all the GLOs and GRCs are entirely linked is not a limit of the algorithm that is based on convergence/divergence average values (or ranges) taken from the literature. For this reason, performing the connections not reaching the maximum number of the expected elements is not an error. Instead, it provides more variability and realism to the network. These considerations are also valid concerning the connections presented below. **Figure 8C** shows the results of the analysis conducted on the *link_gocdb_glo* matrix, containing the GLOs where the GOCs spread their basal dendrites. The algorithm is capable of fully connecting the 90.95% of GOCs to 40 MFs. In the 9.06% of the cases, the GOCs are linked to <40 GLOs, but all the GOCs receive at least the signal from one MF. **Figures 8D–F** also show the count of connection for GLOs, GRCs and GOCs. Considering the inhibition that the GRCs receive from the GOCs through their axon, the matrix *link_goca_glo* evaluation highlights that the developed function generates the 94.97% of these connections. Moreover, considering the GRCs–GOCs connection through ascending axon and PFs, the algorithm fully connects the GOCs to the GRCs, following the convergence/divergence rates presented in section Overview of the Cerebellar Granular Layer Model. Finally, the gap junctions connections evaluation highlights that the 96.60% of GOCs are connected to two other cells (Vervaeke et al., 2010), while only the 1.85% shows one link and the 1.54% is not connected since those cells are located in the borders of the volume. The algorithm that reproduces the MFs branching creates all the clusters with a number of GLOs in the range of 7.7 (±4.1), satisfying the constraint proposed in Sultan and Heck (2003). The number of MFs present in this network configuration is 4051. **Figure 9A** presents a graph showing the percentages of clusters with a different number of GLOs. It is possible to notice that the algorithm creates the 22.72% of clusters with 4 elements, 20.15% with 10 and 19.28% with 12. Moreover, the distances between the elements within the clusters satisfy the constraint of 350 μm. Therefore, the procedure described in section Network Design does not alter the distances distribution inside a cluster. The distances distribution is shown in **Figure 9B**. **Figure 9C** shows all the GLOs belonging to a cluster in the same color.

Computational Results

Simulations have been carried out on a system equipped with an Intel i9-9900X CPU, working at 3.50 GHz, and with 128 GB of DDR4 RAM memory. The system is also equipped with two NVIDIA RTX 2080 GPU (Turing architecture), each one with 2944 CUDA cores, 8 GB of DDR6 memory and working at 1.8 GHz. The boards are connected to the host through a PCI Express 3.0.

The simulations have been also carried out on a single node of an EOS cluster hosted at our University. The node is equipped with two NVIDIA Tesla V100 GPU (Volta architecture), each one with 5120 CUDA cores, 32 GB of HBM2 RAM memory, and working at 1.38 GHz. Each node has an Intel Xeon Silver 4110 CPU, working at 2.1 GHz. Considering the network design stage,

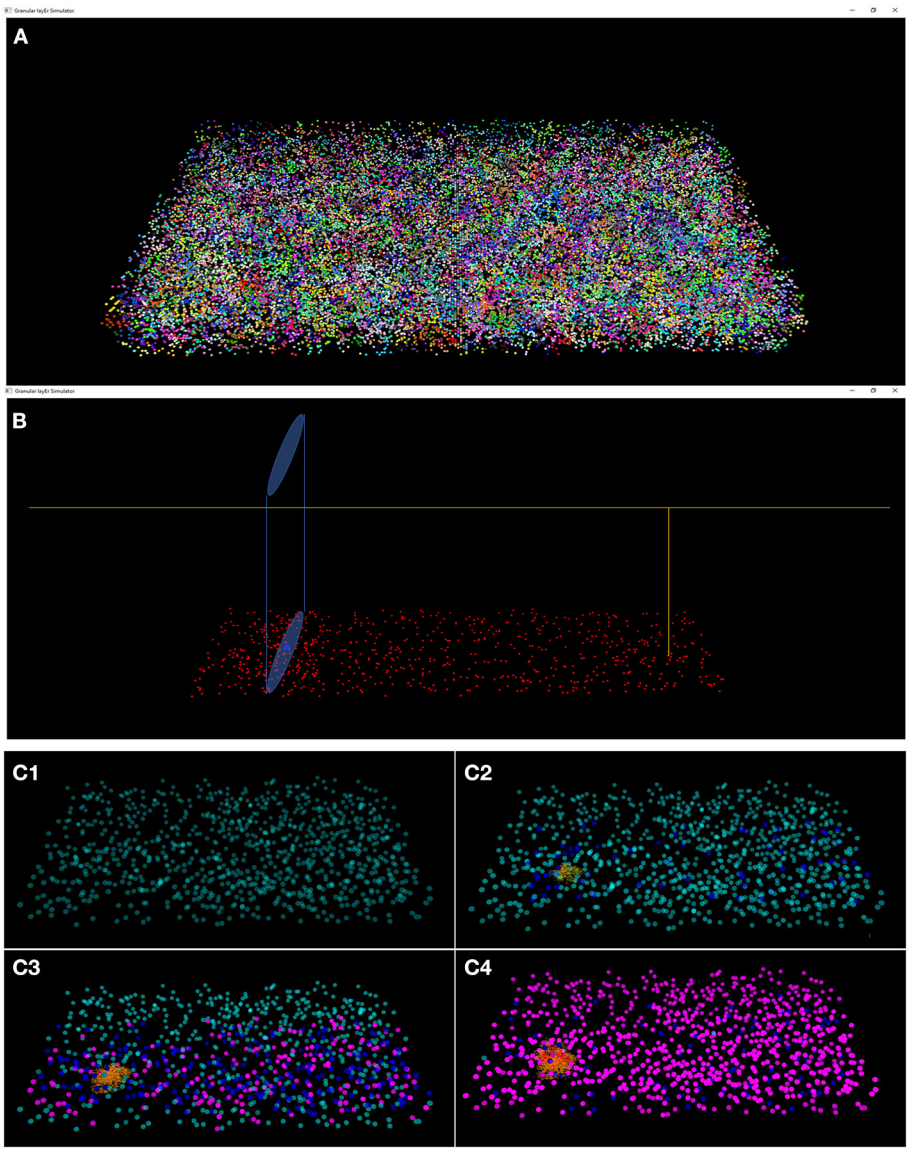


FIGURE 7 | Three tasks of the network. **(A)** MFs branch in the cerebellum forming clusters of GLOs. All the GLOs that belong to a cluster are shown in the same color; **(B)** example of connection between the GRCs (red) and the GOC (blue) through PFs: the GRC ascending axon branches in PF (yellow) that crosses the space dedicated to the GOC apical dendrites (light blue cylinder); **(C)** Four frames of the center-surround organization: **(C1)** only the GOCs have already generated a spontaneous spike; **(C2)** some GRCs and GOCs are stimulated by the active MFs; **(C3)** the core of the center-surround organization is more visible. The GOCs connected through PFs are more excited than the others; **(C4)** final frame of the center-surround.

the developed algorithm places and connects all the elements in only 235 s on an Intel i9 CPU. In particular, the elements placement takes 31.84 s, while their connections and the matrices generation take 203.16 s.

Considering the layer activity simulation, the differential equations in the neurons models have been solved adopting a first-order Euler method, with a time step equals to 0.025 ms. During the single-cells simulators development, the authors performed several tests to set the optimal time step in order to validate the results against the ones produced by the NEURON simulator (Florimbi et al., 2016, 2019).

To evaluate different neuronal behaviors of the network, several protocols have been developed. **Table 2** shows their characteristics.

The first protocol (*Prot1*) aims at evaluating the network response to a background signal of 1 Hz over all the MFs. These inputs start after a *delta* of 350 ms from the beginning of the simulation and last for the whole activity time (T_{end}). On the other hand, to evaluate the network behavior in response to bursts, 10% of the MFs are activated (*Prot2*). They generate bursts lasting 50 ms and whose initial time is randomly selected. Their frequency is 100 Hz. *Prot3* combines background and bursts in

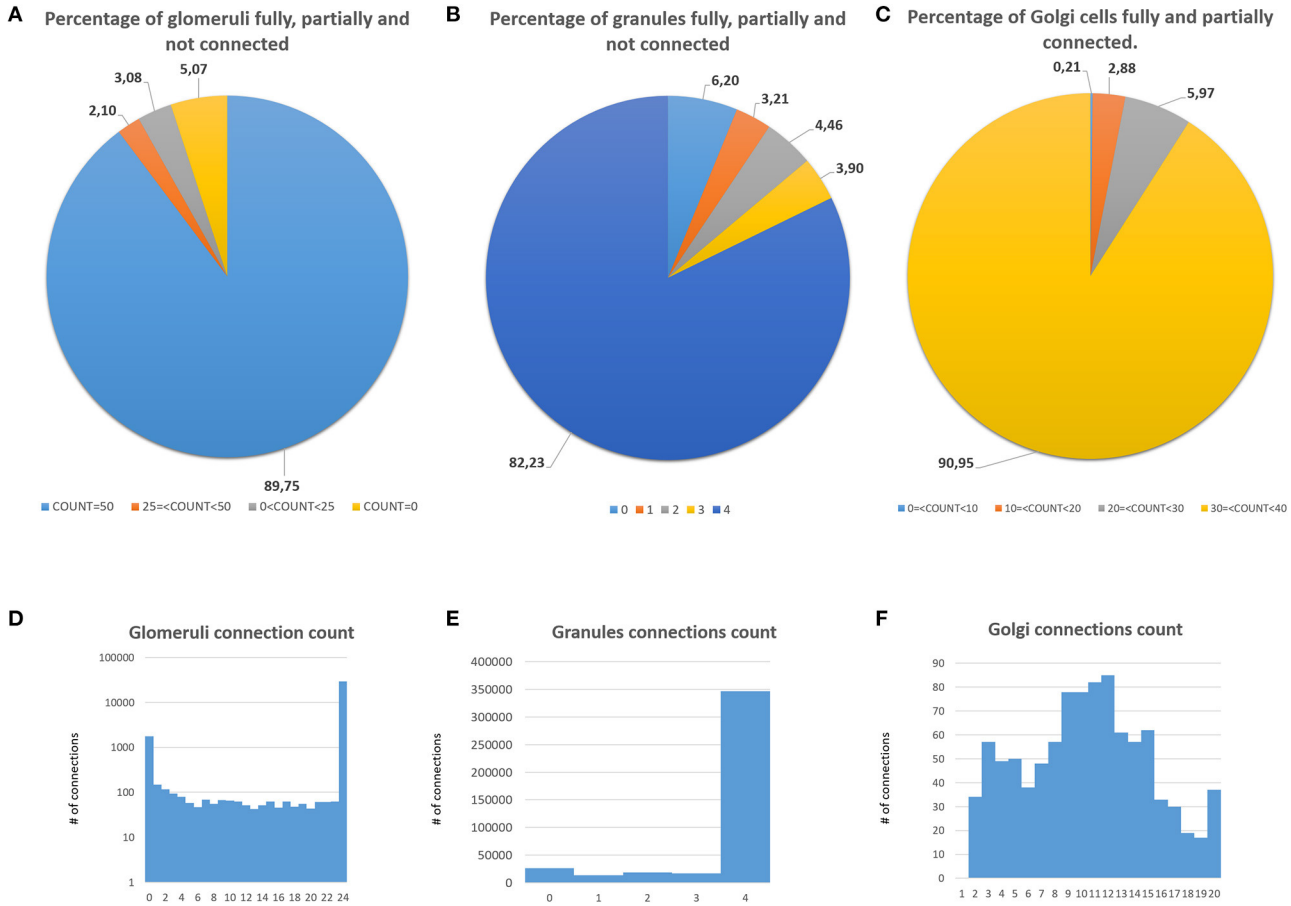


FIGURE 8 | Percentage of connection and connections count of GLOs, GRCs, and GOCs. **(A)** Percentage of GLOs fully (COUNT = 50), partially ($25 \leq$ COUNT < 50, $0 \leq$ COUNT < 25) and not linked (COUNT = 0) to the GRCs; **(B)** percentage of independent connections between GRCs and MFs (through GLOs). Each GRC can be linked to four different GLOs at most; **(C)** percentage of GOCs basal dendrites linked to the MFs (through GLOs). **(D)** Connections count of GLOs. **(E)** Connections count of GRCs. **(F)** Connections count of GOCs.

only one protocol. In particular, each MF presents a background stimulus and a burst. This scenario is not realistic from the physiological point of view since it is very improbable that all the fibers are characterized by both these stimuli in these kinds of simulations. However, this protocol has been introduced as a *stress test* to analyse the performance with a huge computational load. Finally, *Prot4* represents a realistic version of *Prot3*. In fact, all the MFs are characterized by a background stimulus but only 1% of them generates a burst during the simulation. Each protocol has been used as input of three different simulations, where 1, 3, and 10 s of neuronal activity have been evaluated. **Figure 11** shows three graphs presenting the processing time (in logarithmic scale) of each simulation on the different test systems. In particular, the *Serial* version has been processed on the Intel i9 CPU. The single and multi-GPU versions have been processed exploiting the NVIDIA RTX 2080 GPUs, and the NVIDIA V100 GPUs. Similarly, **Figures 10A–C** present the processing times for 1, 3, and 10 s of activity reproduction, respectively. All the elaborations refer to a network with size $600 \times 150 \times 1{,}200\ \mu m^3$, hosting a number of neurons equal to ∼423,066 and of 32,400 GLOs. This network dimension has been chosen to consider a relevant number of elements and to reproduce the characteristic network behaviors. When analyzing the graphs in **Figure 11**, we can firstly observe that, as expected, *Prot3* is the slowest among the four tests. In fact, this protocol provides to the network a huge number of inputs, which increases the times that the algorithm has to evaluate the presynaptic model. Taking into account the number of stimuli that are introduced in the network, it can be concluded that the computational time of the serial versions strongly depends on the number of inputs of the protocol. Both in the 1, 3, and 10 s simulations, the highest number of inputs is provided by the *Prot3* (∼24,300, ∼32,400, and ∼60,750 signals on the whole network, respectively), followed by the *Prot4* (∼4,250, ∼12,350, and ∼40,700, respectively), the *Prot1* (∼4,050, ∼12,150, and ∼40,500, respectively), and the *Prot2* (always ∼2025). As can be seen from these data, in *Prot1* and *Prot4* the number of inputs is similar, and this small difference does not guarantee that *Prot 4* is always the fastest solution among the two. In these serial simulations, *Prot2* takes always the lowest computational time, as expected. The considerations made

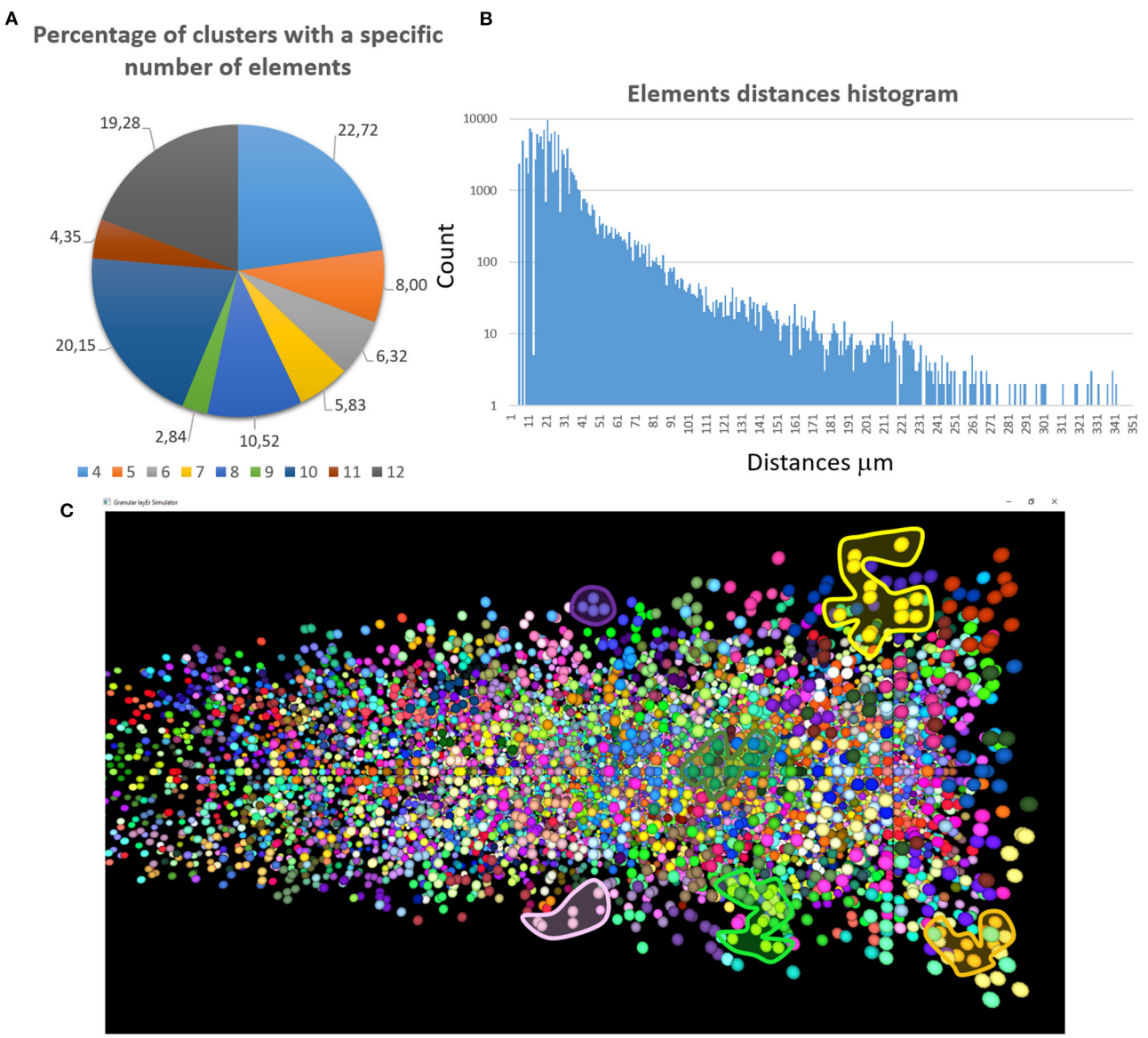


FIGURE 9 | MFs clustering. **(A)** The graph shows the percentage of clusters with a different number of GLOs (4 ÷ 12); **(B)** the graph shows the distances distribution; **(C)** the GLOs within the same cluster are displayed in the same color. By way of example some clusters have been highlighted.

on the link between the number of inputs and the processing time cannot be repeated for the parallel versions since different aspects have to be introduced. For example, the data transfers and the access to the device global memory can introduce delays that increase the processing time. Comparing the parallel and serial versions, all the parallel elaborations perform better than the serial ones. In fact, the serial versions of the 1, 3, and 10 s simulations last from 10 to 14 h, from 30 to 35 h, and from 103 to 321 h (i.e., from 4 to 13 days), respectively. These processing times are strongly decreased considering that, in the worst case, the parallel simulation takes about 6 h (considering the 10 s simulation, with one RTX, *Prot3*). Comparing the single-GPU versions (*RTX* and *EOS* in **Figure 10**), it can be noticed that there are no substantial differences between the processing times, considering the three simulations and the four protocols. This is due to the fact that, even if the EOS GPU (i.e., NVIDIA V100) is equipped with a higher number of CUDA cores than the NVIDIA RTX, this last one features a higher working frequency and a more recent architecture. These characteristics make the difference between the processing times negligible as expected. All the single-GPU parallel versions provide a speedup compared to the serial code processing time. For example, considering the 10 s simulation of *Prot4*, the serial code takes 484999.77 s (i.e., ∼5.61 days), to elaborate the network activity. The parallel code

TABLE 2 | Protocols details.

Protocol ID	Background (Hz)	TI_{back} (s)	TF_{back} (s)	Burst (Hz)	TI_{burst} (s)	TF_{burst} (s)	#MF burst
Prot1	1	0 + delta	T_{end}	–	–	–	–
Prot2	–	–	–	100	T_{rand}	T_{rand} + 0.05	10%
Prot3	1	0 + delta	T_{end}	100	T_{rand}	T_{rand} + 0.05	100%
Prot4	1	0 + delta	T_{end}	100	T_{rand}	T_{rand} + 0.05	1%

The table shows the protocol ID, the background frequency, the instant times when the background starts and ends. Moreover, it shows the burst frequency, the instant times when the burst starts and ends, and the number of MFs activated with the bursts.

on one RTX 2080 GPU takes 15650.66 s (∼4.34 h) while the one on the V100 GPU (EOS) takes 12693.46 s (∼3.52 h), obtaining a speedup of ∼31× and ∼38×, respectively. Also in the *stress test* case (*Prot3*), the single-GPU parallel versions perform better than the serial code, providing a speedup up to ∼72×. The multi-GPU versions always improve the performances compared to the corresponding single-GPU code. For example, always considering the 10 s simulation of *Prot4*, the processing time is 12120.83 s (∼3.37 h) considering two RTX 2080, and 8773.99 s (∼2.44 h) considering the two boards in the EOS system. In these cases, the speedup compared to the serial version increases to ∼40× and ∼55×, respectively. It is worth noticing that the usage of a dual-GPU system does not halve the processing time. This is because some elaborations are performed on a single GPU, moreover, the initialization and the results writing are performed in serial. Finally, not all the memory transfers from the host to the devices can be perfectly overlapped.

Authors also analyzed the code in order to highlight the computational weight of each part of the main *for* loop. The code profiling highlights that about the 95% of the time is taken by the CUDA kernels, the memory transfer account for the 4.6% and only the 0.4% is taken by the host functions. Therefore, there is no reason to implement the spike propagation on GPU. Moreover, the spike propagation could degrade the GPU performance since some parts are strictly sequential.

These results demonstrate that this kind of technology, together with an efficient code development, allows reducing the serial processing times. In this respect, authors decided to perform a very long simulation (50 s) of the neuronal activity using the GES system adopting the *Prot4*. The choice of this protocol has been made since it is the most realistic one, combining the background signals with the bursts. The simulation has been run on the EOS cluster exploiting two NVIDIA V100 GPUs. To reproduce 50 s of neuronal activity, the system takes 49839.68 s (∼13 h). This result demonstrates that this system is suitable to reproduce very long neuronal activity, giving the opportunity to study particular behaviors that are not reproducible with other kinds of simulators due to their slower processing times. The GPU technology, together with the optimization developed to efficiently perform the data transfers and the memory accesses and to process the neuronal activity, constitutes an appropriate solution for the network simulation. In particular, this system is capable of fast reproducing a considerable portion of the granular layer, characterized by a high number of neurons, described by complex mathematical models.

Figure 11 shows the raster plots to graphically visualize the network activity in response to *Prot2*. In **Figure 11A**, it is possible to evaluate the GOCs activity. In particular, these cells generate spontaneous firing and, when stimulated by MFs, they increase their firing frequency. Only some cells are shown (id 50–80) and in a reduced time-window (0–450 ms). On the other hand, the GRCs do not show spontaneous firing and they generate spikes only when stimulated (**Figure 11B**). In fact, as it is possible to notice from the raster plot, the cell generates a spike after receiving 3–4 stimuli by the MFs.

Finally, a further validation of the proposed network has been achieved reproducing the typical center-surround organization of the granular layer (Mapelli and D'Angelo, 2007; Solinas et al., 2010; Gandolfi et al., 2014). In fact, several electrophysiological experiments (Mapelli and D'Angelo, 2007; Mapelli et al., 2010a,b) showed that a MFs bundle can stimulate a specific area of the granular layer, generating a central area of excitation and a surrounding one of inhibition. To reproduce this organization, the protocol adopted as input is characterized by the activation of the MFs present in a selected area whose diameter is 50 μm. It is important to highlight that the GLOs, and thus the GRCs and GOCs, excited by these MFs can be also outside this area. In fact, as explained before, each MF stimulates all the GLOs within a cluster and, even if all the elements within a cluster are not so far, it could be possible that they are outside the selected area. Also in this case, it is possible to correctly reproduce the center-surround organization using as input the branched MFs. In particular, in this simulation, the MFs within the selected area are 9 and each one stimulates the GLOs with a burst of 50 ms and a frequency of 150 Hz. Moreover, the entire network is considered (i.e., all the elements can react to an eventual stimulus) and all the connections are switched-on while, in the reference papers, only the area of interest is switched-on. The response of the center-surround shown in **Figure 12** is the result of a single simulation run. The burst stimulation causes a central area with a stronger excitation (red area) than the surrounding one (blue area), where the GOCs inhibition limits the rate of GRCs output, overcoming the excitation around the core.

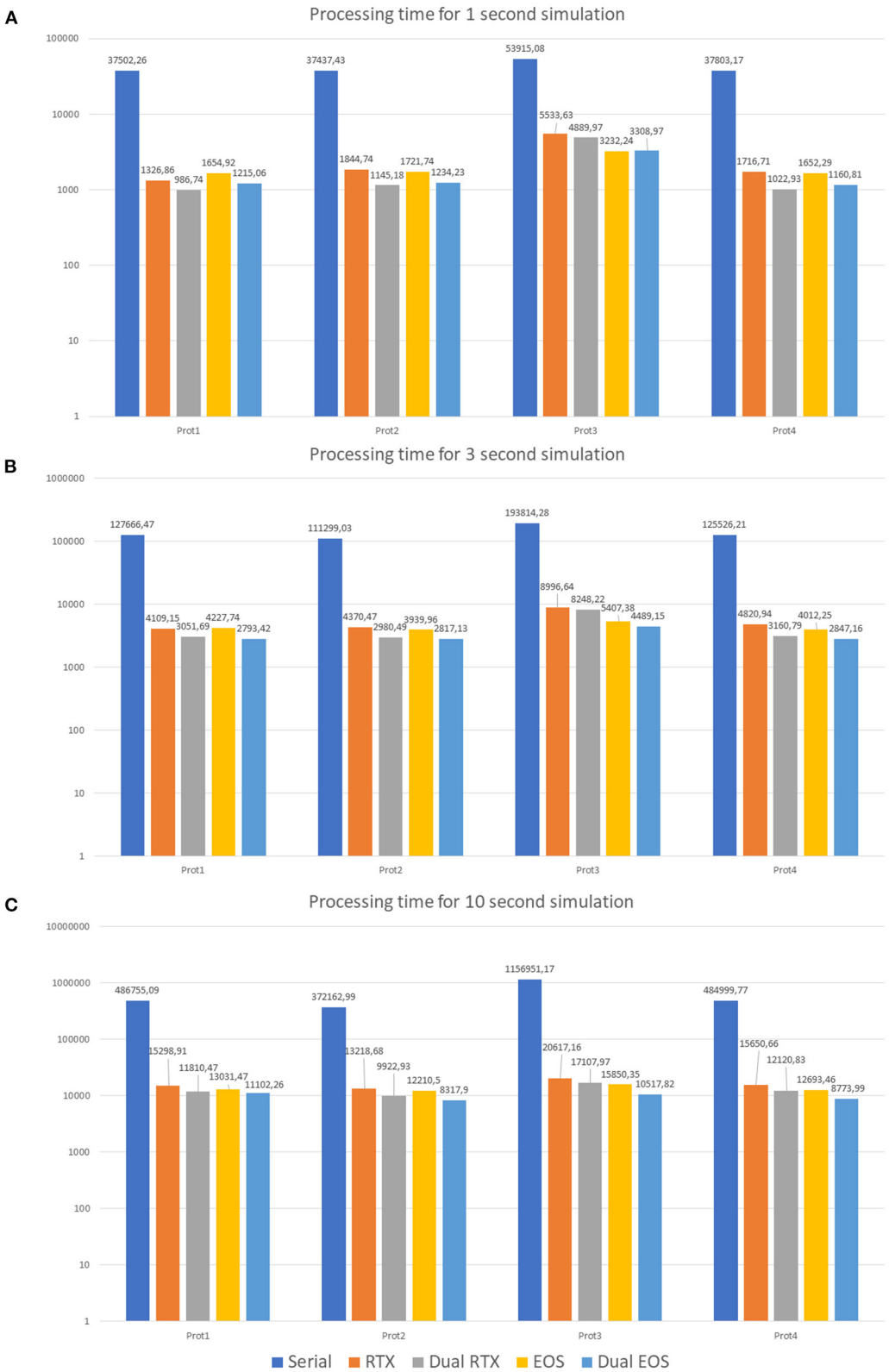


FIGURE 10 | Processing times. 1 s **(A)**, 3 s **(B)**, and 10 s **(C)** neuronal activity simulations on the different test systems. The serial simulation ran on Intel i9 CPU, RTX and Dual RTX refer to the NVIDIA RTX 2080 boards, and EOS and Dual EOS refer to the NVIDIA V100 boards. The graphs show the results of the simulations where the four protocols have been tested (Prot1, Prot2, Prot3, and Prot4). The graphs are in logarithmic scale. The legend refers the three graphs.

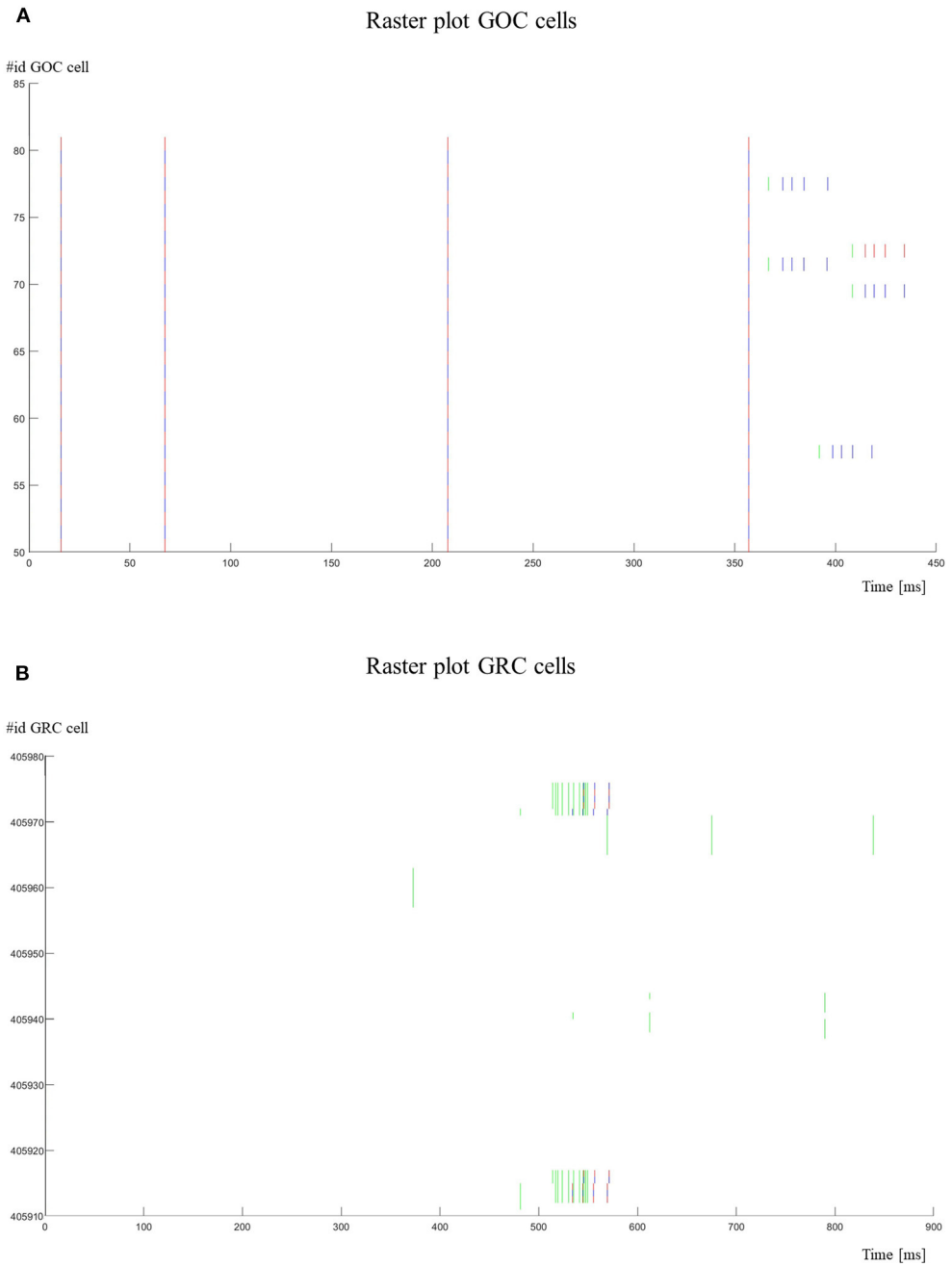


FIGURE 11 | Raster plots. **(A)** The activity of the GOCs (id 50–80) is shown. The cells show a spontaneous firing until they are stimulated (green lines) by MFs. In these cases, their firing frequency is increased; **(B)** the activity of the GRCs (id 405911–405976) is shown. Some GRCs are stimulated with bursts by MFs. It is possible to notice that GRCs generate a spike only after 3–4 stimuli. The red lines refer to the cells with an even id, while the blue lines refer to the cells with an odd id.

Memory Occupancy

One of the most important aspects of the simulator is that it is parametric. The user can vary several parameters, such as the volume of the network, to simulate different granular layer configurations. This characteristic makes the system very flexible for what concerns the network construction. Concerning the network design stage, running on the CPU, the code allocates 600 B for parameters used in the network construction and elements connections, whose number is not proportional to the number of neurons or elements. Moreover, the code allocates a memory space proportional to the number of GOCs, GRCs, GLOs and MFs, as shown in Equation (15):

$$MEMdesign = 600 + 15432\, n_{goc} + 32\, n_{grc} + 1292\, n_{glo} + 296\, n_{MF} \quad (15)$$

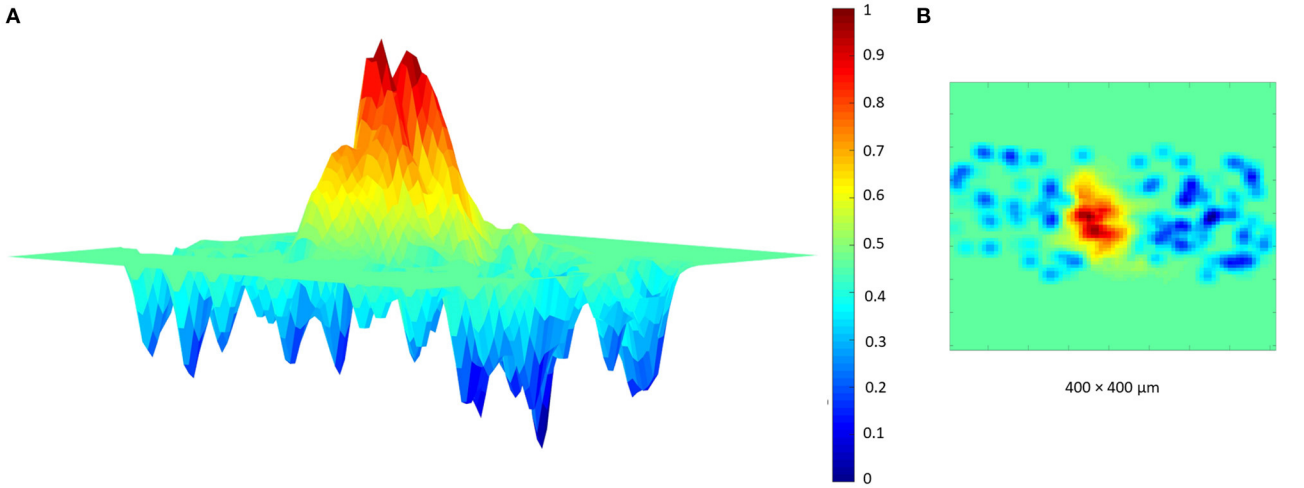


FIGURE 12 | Center surround organization. The MFs stimulate the GLOs with a burst of 50 ms and 150 Hz. The network response is characterized by an excited core caused by the GRC firing (red area). This center is surrounded by an area, where the GRCs response is inhibited by the GOCs. **(A)** Center-surround lateral view. **(B)** Center-surround top view.

where *MEMdesign* is the amount of memory expressed in byte. In the case of the present configuration, this value is equal to ∼70 MB. If the second stage, i.e., the network simulation, runs on the CPU, the code allocates a total amount of memory given by Equation (16):

$$MEMcpusim = 42542\, n_{goc} + 12220\, n_{grc} + 204\, n_{glo} + 8004\, n_{MF} \tag{16}$$

In the simulation of the present network configuration, the allocated amount of RAM is ∼5 GB.

If the network simulation is performed on the GPU, the major constraint is represented by the available RAM provided on the device. In order to evaluate the maximum volume that can be reproduced with a specific board, it is important to compute the amount of global memory device to be allocated using the *cudaMalloc* function. Considering the network configuration simulated, the code allocates 580 B for parameters used both for the GOCs and GRCs activities, whose allocation is not proportional to the number of reproduced cells. On the other hand, it is necessary to allocate space for the variables used in the neuronal activity computation. This memory size is proportional to the GOCs and GRCs number and, for this reason, $2788 \times n_{goc}$ B and $7172 \times n_{grc}$ B are allocated, respectively. Therefore, the total amount of memory needed to reproduce a generic network configuration is given by Equation (17).

$$RAMGPU = 580 + 2788\, n_{goc} + 7172\, n_{grc} \tag{17}$$

It is worth noticing that the RAM occupancy on the GPU is lower than on the CPU. The reason is that part of the connectivity is processed on the CPU; therefore, these data are not allocated on the GPU memory. Moreover, this memory amount can be generalized if two or more GPUs are used: in this case the values of n_{goc} and n_{grc} should be divided by the number of available devices.

If the number of cells is expressed as a function of cellular densities, it is possible to estimate if the volume of a certain network configuration can be stored using a specific GPU board. Equation (18) expresses the bound of the volume in function of the neurons densities and the available RAM memory.

$$V \leq \frac{RAMGPU - 580}{2788\, n_{goc} + 7172\, n_{grc}} \tag{18}$$

In Equation (11), V is the volume expressed in mm^3 and the memory occupancy is measured in byte. In the configuration adopted in this work, the total amount of allocated memory is ∼2.88 GB, which represents the ∼24% of the RAM of the NVIDIA RTX 2080 board. It is possible to conclude that the amount of memory allocated for the network design stage is negligible compared to the simulation stage performed both on the CPU and on the GPU. Finally, the memory requirements of the two stages are compatible with a standard desktop system. Therefore, it is not mandatory to use a cluster or supercomputer to run a realistic simulation with the proposed system.

Scalability Analysis

The scalability of the proposed system has been evaluated considering two network with x and y dimensions halved (Network2) and doubled (Network3) with respect to the network described in the previous sections (Network1). The performance has been evaluated both in terms of elements placed and connected and of processing times. In terms of elements placement and connections, the considerations are the same made for the original network. Concerning the processing times, Network2 takes approximatively four times less than Network1. This is an expected value since the volume simulated in Network1 is four time the one simulated in Network2. Similarly, network3

runs four times slower than network1 as it has a quarter of the volume.

Comparison With the State of the Art

The main differences between this work and the literature are related to the neuronal models chosen to reproduce the activity of the neurons, the simulation duration and the integration time step. Here, some of the most relevant and similar works at the state of the art are reported and compared with this work. In Naveros et al. (2015), authors developed an event- and time-driven spiking neural network simulator for a hybrid CPU-GPU platform. It consists of a very dense granular layer and a Purkinje layer with a small number of cells, where neurons are reproduced using LIF models and characterization tables (computed offline) containing the dynamic of each cell. To reproduce 10 s of neuronal activity, the simulation of 3 million neurons and 274 million synapses takes 987.44 s on an Intel i7 CPU equipped with 32 GB RAM and an NVIDIA GTX 470 GPU equipped with 1.28 GB RAM. This result cannot be directly compared to this work for two main reasons: the most important is the different model chosen and the other is related to the integration step, which varies from 0.1 to 1 ms that is higher than the one used in this work (0.025 ms).

Another interesting work that reproduces the cat cerebellum network containing more than a billion spiking neurons, is described in Yamazaki et al. (2019). Authors do not exploit the GPU technology but an HPC special purpose computer equipped with 1280 PEZY-SC processors. This system elaborates in real-time 1 s of neuronal activity, with an integration step of 1 ms. Also in this case, cells are described by LIF models and the connectivity rules are not updated. Moreover, the synapses are characterized only by the AMPA receptors. Finally, this architecture represents a completely different philosophy that from one side benefits the application specificity, from another one follows a not fairly comparable approach in terms of programmability, size/performance ratio and technological life of the employed components.

Authors of Gleeson et al. (2007) provide a tool to build, visualize and analyse network models in a 3D space. The network design reproduces very realistic and complex neuron morphologies exploiting the Hodgkin and Huxley model. Nevertheless, they run simulations of up to only 5,000 neurons on a single-processor machine that takes 1–2 h for 4 s of activity. In this case, even if the morphology is very detailed, the simulation part is not so efficient as the one proposed in this work. On the other hand, the cerebellar granular layer network developed in Solinas et al. (2010) is the one considered as reference for the present work. In fact, these networks present the same mathematical models (even if their models are written for the NEURON simulator) and connection rules. The main difference concerns the cellular morphology and the elements displacement. In this case, the cellular soma is represented by a point (not sphere) and this means that two soma can be overlapped. Moreover, during the cells displacement, the algorithm does not take into account the minimum distances between cells. They create a network inside a 3D space (i.e., a cube with 100 μm edge length) and that includes 315 MFs and 4,393 neurons (4,096 GRCs, 27 GOCs, 270 basket, and stellate cells). The reproduction of 3 s neuronal activity requires about 20 h on a Pentium-5 dual-core and 30 min using 80 CPUs on the CASPUR parallel cluster.

The work in Van Der Vlag et al. (2019) reports a multi-GPU implementation of a neuronal network based on the Hodgkin and Huxley model. The connectivity is based on the uniform or on the Gaussian distribution. Therefore, no realistic connection rules are considered. Moreover, the simulated time is only 100 ms with a time step of 0.05 ms.

Authors of Yavuz et al. (2016) proposed a systems to automatically generate CUDA kernels and runtime codes according to a user-defined network model. The work only supports single GPU systems.

A multi-GPU framework is proposed in Chou et al. (2018). However, this framework only includes the four and nine parameters Izhikevich models. Moreover, the authors evaluated the performance on a random spiking network. Thus, a direct comparison with our work would not be fair.

In Casali et al. (2019), authors present the whole cerebellar network reconstruction (i.e., granular, Purkinje, and molecular layers) based on the morphological details and connection rules used also in this work. Considering the design part of the system, the main differences with GES are the absence of the gap junctions and of the organization of the MFs in *rosettes*. Another important aspect to highlight is that in Casali et al. (2019) neurons are represented with single-point LIF models since the work is focused on a detailed network construction. Another difference between the systems is that the network in Casali et al. (2019) is simulated on pyNEST and pyNEURON while, in GES, optimized codes for the network design and simulation have been developed in C/CUDA languages. Authors in Casali et al. (2019) simulated a cerebellar cortex volume of $400 \times 400 \times 330$ μm^3 with a total amount of 96,734 cells even if the system is scalable. Authors do not provide information about the simulation time and the technical features of the HPC system used for the code elaboration. The integration step is set to 0.1 ms, so four times bigger than the one used in this work. Even if this network and the one described in the present work are based on the same physiological data exploited in the network reconstruction, it is not possible to make a comparison on the efficiency of the two systems since some data are missing.

Limits and Future Works

Even if the GES system reconstructs the granular layer and reproduces its behavior, some aspects can still be improved. One of the main features of this simulator is that it is possible to change the models representing the neurons, without any modifications in the network design module. For this reason, one of the aspects that can be improved is the introduction of multi-compartment models with active compartments. This aspect will lead also to include more detailed morphologies, which will be also graphically shown through the graphical interface. Another aspect that could be improved in the design module is the introduction of a more specific constraint in the way the gap junctions are generated. Moreover, it will be interesting to reproduce a larger area of the granular layer where the MFs will stimulate more than one cluster of GLOs. Finally, since authors

have already developed the Purkinje cells simulator on GPU (Torti et al., 2019), an efficient way to include these cells in the network will be studied. In this way, also the molecular and Purkinje cell layers will be reconstructed to obtain a complete cerebellar cortex network.

CONCLUSIONS

The use of HPC technologies in computational modeling in neuroscience is becoming more and more attractive and widespread. In particular, the GPUs play a critical role in the large-scale networks elaboration where the activity of a huge number of connected neurons is reproduced.

This paper presented the GES system capable of reconstructing, simulating and visualizing the cerebellar granular layer, exploiting a desktop system with the GPU device.

The algorithm reconstructs the cerebellar granular layer following detailed rules and data aligned with the state of the art, targeting a high level of realism. The granular layer reconstruction in a 3D space is performed by an efficient serial code that takes <4 min to place and connect the neurons in a $600 \times 150 \times 1{,}200\ \mu m^3$ volume (with 432,000 GRCs, 972 GOCs, 32,399 GLOs, and 4,051 MFs).

The simulator is also characterized by two parallel codes elaborating the network neuronal activity. The GPU device has proved to be vital to strongly reduce the computational time of the serial elaboration. Different protocols considering background, bursts and the combination of them have been tested. In particular, *Prot4* provided the most realistic scenario performing both background and bursts. In this case, the system reproduces 10 s of neuronal activity in 4.34 and 3.37 h exploiting a single and multi-GPU desktop system (equipped with one or two NVIDIA RTX 2080 GPU, respectively). Moreover, if the code runs on one node of the EOS system the processing time further decreases to 3.52 and 2.44 h exploiting one or two NVIDIA V100 GPU, respectively. The processing time of the related serial code takes ∼135 h (∼5.61 days) on an Intel i9 CPU and this means that the parallel versions reach a speedup up to ∼38× in the single-GPU version, and up to ∼55× in the multi-GPU code. This kind of technology and the development of an efficient code allowed to perform very long simulations, useful to study particular network behaviors reproducible only analyzing long time frames. In this work, authors presented a first long-lasting simulation (*Prot4*), reproducing 50 s of the network activity in ∼13 h on one node of the EOS system. A crucial aspect to highlight is that the code is flexible and allows the user to reconstruct and simulate networks with different dimensions. Finally, a graphical interface has been developed to graphically analyse the results.

DATA AVAILABILITY STATEMENT

The code for this study can be found in the mclab website http://mclab.unipv.it/index.php/ges. The source code is available from the correspondent authors upon reasonable request.

AUTHOR CONTRIBUTIONS

GF, ET, and SM: conceptualization and methodology. GF and ET: investigation, software, and writing—original draft. SM, ED'A, and FL: writing—review and editing. FL: supervision. All authors contributed to the article and approved the submitted version.

FUNDING

ED'A received funding from the European Union's Horizon 2020 Framework Program for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2) and Specific Grant Agreement No. 945539 (Human Brain Project SGA3).

REFERENCES

Beyeler, M., Richert, M., Dutt, N. D., and Krichmar, J. L. (2014). Efficient spiking neural network model of pattern motion selectivity in visual cortex. *Neuroinformatics* 12, 435–454. doi: 10.1007/s12021-014-9220-y

Bouchard, K. E., Aimone, J. B., Chun, M., Dean, T., Denker, M., Diesmann, M., et al. (2016). High-performance computing in neuroscience for data-driven discovery, integration, and dissemination. *Neuron* 92, 628–631. doi: 10.1016/j.neuron.2016.10.035

Casali, S., Marenzi, E., Medini, C., Casellato, C., and D'Angelo, E. (2019). Reconstruction and simulation of a scaffold model of the cerebellar network. *Front. Neuroinform.* 13:37. doi: 10.3389/fninf.2019.00037

Chou, T. S., Kashyap, H. J., Xing, J., Listopad, S., Rounds, E. L., Beyeler, M., et al. (2018). "CARLsim 4: an open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," in *Proceedings of the International Joint Conference on Neural Networks* (Rio de Janeiro: Institute of Electrical and Electronics Engineers Inc.). doi: 10.1109/IJCNN.2018.8489326

Cremonesi, F., and Schürmann, F. (2020). Understanding computational costs of cellular-level brain tissue simulations through analytical performance models. *Neuroinform.* 18, 407–428. doi: 10.1007/s12021-019-09451-w

D'Angelo, E., Antonietti, A., Casali, S., Casellato, C., Garrido, J. A., Luque, N. R., et al. (2016). Modeling the cerebellar microcircuit: new strategies for a long-standing issue. *Front. Cell Neurosci.* 10:176. doi: 10.3389/fncel.2016.00176

D'Angelo, E., Nieus, T., Maffei, A., Armano, S., Rossi, P., Taglietti, V., et al. (2001). Theta-frequency bursting and resonance in cerebellar granule cells: experimental evidence and modeling of a slow K^+-dependent mechanism. *J. Neurosci.* 21, 759–770. doi: 10.1523/jneurosci.21-03-00759.2001

D'Angelo, E., Solinas, S., Mapelli, J., Gandolfi, D., Mapelli, L., and Prestori, F. (2013). The cerebellar Golgi cell and spatiotemporal organization of granular layer activity. *Front. Neural Circuits* 7:93. doi: 10.3389/fncir.2013.00093

Dieudonné, S. (1998). Submillisecond kinetics and low efficacy of parallel fibre-Golgi cell synaptic currents in the rat cerebellum. *J. Physiol.* 510, 845–866. doi: 10.1111/j.1469-7793.1998.845bj.x

Fidjeland, A. K., Gamez, D., Shanahan, M. P., and Lazdins, E. (2013). Three tools for the real-time simulation of embodied spiking neural networks using GPUs. *Neuroinformatics* 11, 267–290. doi: 10.1007/s12021-012-9174-x

Florimbi, G., Torti, E., Danese, G., and Leporati, F. (2017). "High performant simulations of cerebellar Golgi cells activity," in *Proceedings—2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2017* (St. Petersburg: IEEE), 527–534. doi: 10.1109/PDP.2017.91

Florimbi, G., Torti, E., Masoli, S., D'Angelo, E., Danese, G., and Leporati, F. (2016). The human brain project: parallel technologies for biologically accurate simulation of granule cells. *Microprocess. Microsyst.* 47, 303–313. doi: 10.1016/J.MICPRO.2016.05.015

Florimbi, G., Torti, E., Masoli, S., D'Angelo, E., Danese, G., and Leporati, F. (2019). Exploiting multi-core and many-core architectures for efficient simulation of biologically realistic models of Golgi cells. *J. Parallel Distrib. Comput.* 126, 48–66. doi: 10.1016/j.jpdc.2018.12.004

Gandolfi, D., Pozzi, P., Tognolina, M., Chirico, G., Mapelli, J., and D'Angelo, E. (2014). The spatiotemporal organization of cerebellar network activity resolved by two-photon imaging of multiple single neurons. *Front. Cell Neurosci.* 8:92. doi: 10.3389/fncel.2014.00092

Gleeson, P., Steuber, V., and Silver, R. A. (2007). neuroConstruct: A tool for modeling networks of neurons in 3D space. *Neuron* 54, 219–235. doi: 10.1016/j.neuron.2007.03.025

Hodgkin, A. L., and Huxley, A. F. (1990). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bull. Math. Biol.* 52, 25–71. doi: 10.1007/BF02459568

Houston, C. M., Diamanti, E., Diamantaki, M., Kutsarova, E., Cook, A., Sultan, F., et al. (2017). Exploring the significance of morphological diversity for cerebellar granule cell excitability. *Sci. Rep.* 7:46147. doi: 10.1038/srep46147

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw.* 15, 1063–1070. doi: 10.1109/TNN.2004.832719

Kanichay, R. T., and Silver, R. A. (2008). Synaptic and cellular properties of the feedforward inhibitory circuit within the input layer of the cerebellar cortex. *J. Neurosci.* 28, 8955–8967. doi: 10.1523/JNEUROSCI.5469-07.2008

Korbo, L., Andersen, B. B., Ladefoged, O., and Møller, A. (1993). Total numbers of various cell types in rat cerebellar cortex estimated using an unbiased stereological method. *Brain Res.* 609, 262–268. doi: 10.1016/0006-8993(93)90881-m

Mapelli, J., and D'Angelo, E. (2007). The spatial organization of long-term synaptic plasticity at the input stage of cerebellum. *J. Neurosci.* 27, 1285–1296. doi: 10.1523/JNEUROSCI.4873-06.2007

Mapelli, J., Gandolfi, D., and D'Angelo, E. (2010a). Combinatorial responses controlled by synaptic inhibition in the cerebellum granular layer. *J. Neurophysiol.* 103, 250–261. doi: 10.1152/jn.00642.2009

Mapelli, J., Gandolfi, D., and D'Angelo, E. (2010b). High-pass filtering and dynamic gain regulation enhance vertical bursts transmission along the mossy fiber pathway of cerebellum. *Front. Cell Neurosci.* 4:14. doi: 10.3389/fncel.2010.00014

Mapelli, L., Solinas, S., and D'Angelo, E. (2014). Integration and regulation of glomerular inhibition in the cerebellar granular layer circuit. *Front. Cell Neurosci.* 8:55. doi: 10.3389/fncel.2014.00055

Naveros, F., Luque, N. R., Garrido, J. A., Carrillo, R. R., Anguita, M., and Ros, E. (2015). A spiking neural simulator integrating event-driven and time-driven computation schemes using parallel CPU-GPU co-processing: a case study. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1567–1574. doi: 10.1109/TNNLS.2014.2345844

Nieus, T., Sola, E., Mapelli, J., Saftenku, E., Rossi, P., and D'Angelo, E. (2006). LTP regulates burst initiation and frequency at mossy fiber-granule cell synapses of rat cerebellum: experimental observations and theoretical predictions. *J. Neurophysiol.* 95, 686–699. doi: 10.1152/jn.00696.2005

Nieus, T. R., Mapelli, L., and D'Angelo, E. (2014). Regulation of output spike patterns by phasic inhibition in cerebellar granule cells. *Front. Cell Neurosci.* 8:246. doi: 10.3389/fncel.2014.00246

NVIDIA (2019). *CUDA C Best Practices Guide*. Available online at: https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf

Rennich, S., and NVIDIA (2014). *CUDA C/C++ Streams and Concurrency*. Available online at: https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

Rossi, D. J., and Hamann, M. (1998). Spillover-mediated transmission at inhibitory synapses promoted by high affinity α6 subunit GABA(A) receptors and glomerular geometry. *Neuron* 20, 783–795. doi: 10.1016/S0896-6273(00)81016-8

Sellers, G., Wright, R. S., and Haemel, N. (2015). *OpenGL SuperBible. 7th Edn.* A.-W. Professional.

Solinas, S., Forti, L., Cesana, E., Mapelli, J., De Schutter, E., and D'Angelo, E. (2008). Computational reconstruction of pacemaking and intrinsic electroresponsiveness in cerebellar golgi cells. *Front. Cell Neurosci.* 1:2. doi: 10.3389/neuro.03.002.2007

Solinas, S., Nieus, T., and D'Angelo, E. (2010). A realistic large-scale model of the cerebellum granular layer predicts circuit spatio-temporal filtering properties. *Front. Cell Neurosci.* 4:12. doi: 10.3389/fncel.2010.00012

Sultan, F., and Heck, D. (2003). Detection of sequences in the cerebellar cortex: numerical estimate of the possible number of tidal-wave inducing sequences represented. *J. Physiol.* 97, 591–600. doi: 10.1016/j.jphysparis.2004.01.016

Torti, E., Masoli, S., Florimbi, G., D'Angelo, E., Ticli, M., and Leporati, F. (2019). "GPU parallelization of realistic Purkinje cells with complex morphology," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (Pavia: IEEE), 266–273. doi: 10.1109/EMPDP.2019.8671581

Van Der Vlag, M. A., Smaragdos, G., Al-Ars, Z., and Strydis, C. (2019). Exploring complex brain-simulation workloads on multi-GPU deployments. *ACM Trans. Archit. Code Optim.* 16, 1–25. doi: 10.1145/3371235

Vervaeke, K., Lorincz, A., Gleeson, P., Farinella, M., Nusser, Z., Angus Silver, R., (2010). Rapid Desynchronization of an Electrically Coupled Interneuron Network with Sparse Excitatory Synaptic Input. *Neuron* 67, 435–451. doi: 10.1016/j.neuron.2010.06.028

Yamazaki, T., Igarashi, J., Makino, J., and Ebisuzaki, T. (2019). Real-time simulation of a cat-scale artificial cerebellum on PEZY-SC processors. *Int. J. High Perform. Comput. Appl.* 33, 155–168. doi: 10.1177/1094342017710705

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. *Sci. Rep.* 6:18854. doi: 10.1038/srep18854

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroinformatics

ORIGINAL RESEARCH
published: 22 March 2022
doi: 10.3389/fninf.2022.790966

BOLD Monitoring in the Neural Simulator ANNarchy

*Oliver Maith†, Helge Ülo Dinkelbach†, Javier Baladron, Julien Vitay and Fred H. Hamker**

Department of Computer Science, Chemnitz University of Technology, Chemnitz, Germany

Multi-scale network models that simultaneously simulate different measurable signals at different spatial and temporal scales, such as membrane potentials of single neurons, population firing rates, local field potentials, and blood-oxygen-level-dependent (BOLD) signals, are becoming increasingly popular in computational neuroscience. The transformation of the underlying simulated neuronal activity of these models to simulated non-invasive measurements, such as BOLD signals, is particularly relevant. The present work describes the implementation of a BOLD monitor within the neural simulator ANNarchy to allow an on-line computation of simulated BOLD signals from neural network models. An active research topic regarding the simulation of BOLD signals is the coupling of neural processes to cerebral blood flow (CBF) and cerebral metabolic rate of oxygen (CMRO2). The flexibility of ANNarchy allows users to define this coupling with a high degree of freedom and thus, not only allows to relate mesoscopic network models of populations of spiking neurons to experimental BOLD data, but also to investigate different hypotheses regarding the coupling between neural processes, CBF and CMRO2 with these models. In this study, we demonstrate how simulated BOLD signals can be obtained from a network model consisting of multiple spiking neuron populations. We first demonstrate the use of the Balloon model, the predominant model for simulating BOLD signals, as well as the possibility of using novel user-defined models, such as a variant of the Balloon model with separately driven CBF and CMRO2 signals. We emphasize how different hypotheses about the coupling between neural processes, CBF and CMRO2 can be implemented and how these different couplings affect the simulated BOLD signals. With the BOLD monitor presented here, ANNarchy provides a tool for modelers who want to relate their network models to experimental MRI data and for scientists who want to extend their studies of the coupling between neural processes and the BOLD signal by using modeling approaches. This facilitates the investigation and model-based analysis of experimental BOLD data and thus improves multi-scale understanding of neural processes in humans.

Keywords: blood-oxygen-level-dependent signal, neural simulator, spiking networks, rate-coded networks, Balloon model, neurovascular coupling, cerebral blood flow, cerebral metabolic rate of oxygen

OPEN ACCESS

Edited by:
Kelly Shen,
Simon Fraser University, Canada

Reviewed by:
Dominic Standage,
University of Birmingham,
United Kingdom
Erin Lindsay Mazerolle,
St. Francis Xavier University, Canada

***Correspondence:**
Fred H. Hamker
fred.hamker@
informatik.tu-chemnitz.de

†These authors have contributed equally to this work

Received: *07 October 2021*
Accepted: *08 February 2022*
Published: *22 March 2022*

Citation:
Maith O, Dinkelbach HÜ, Baladron J, Vitay J and Hamker FH (2022) BOLD Monitoring in the Neural Simulator ANNarchy. Front. Neuroinform. 16:790966. doi: 10.3389/fninf.2022.790966

1. INTRODUCTION

Network models are simulated neural networks composed of multiple computational units that model the dynamics of biological neurons at various levels of complexity: macroscopic mean-field or neural mass models simulate the average dynamics of large groups of neurons, rate-coded point neuron models simulate the instantaneous mean firing rate of individual neurons, spiking point

neuron models simulate precise spike timings, while multi-compartmental neuron models also consider the 3D structure of the neurons. Such network models can exhibit complex dynamics due to the recurrent connectivity between the simulated neurons and can be validated against a large amount of experimental data and make extensive predictions at different scales, such as patterns in spike timing, local field potentials or electroencephalography, and blood-oxygen-level-dependent (BOLD) signals from magnetic resonance imaging (MRI). Large-scale network models are becoming increasingly common in computational neuroscience (see Einevoll et al., 2019 for a review about brain simulations with network models). Concerning MRI data, network models can be used primarily to examine the underlying neural mechanisms of the experimental non-invasive data or, for example, to better understand the relationship between the structural connectivity and the functional dynamics of neural circuits (Popovych et al., 2019).

The ANNarchy neural simulator (Vitay et al., 2015) provides a user-friendly equation-based interface which can be used to create large-scale rate-coded and spiking network models at different levels of biological realism. Recently, the ANNarchy neural simulator has been combined with the whole-brain neural simulator The Virtual Brain (TVB) (Ritter et al., 2013; Sanz Leon et al., 2013; Meier et al., 2021) to allow the creation of multi-scale network models. This allows to study how processes in detailed spiking network models of specific brain regions such as the basal ganglia created in ANNarchy affect the dynamics of the whole cortex simulated in TVB (Meier et al., 2021). To further improve the usability of ANNarchy, we introduce a BOLD signal monitoring module (called BOLD monitor in ANNarchy) that allows obtaining simulated BOLD signals from spiking and rate-coded network models in an on-line manner.

Several modeling tools already provide utilities to obtain simulated BOLD signals from network models TVB, Dynamic Causal Modeling (Friston et al., 2003) in SPM (Penny et al., 2011), neuRosim (Welvaert et al., 2011), which so far have been applied mainly to network models at the macroscopic level of detail (Vanni et al., 2015). These methods mainly use variants of the Balloon model to compute simulated BOLD signals (Buxton et al., 1998, 2004; Stephan et al., 2007). Hereafter, we will refer to the Balloon model and other such models that convert an input time signal into a simulated BOLD signal, generally as BOLD models. A critical open issue when simulating BOLD signals from network models is the neurovascular coupling, i.e., which neural mechanisms are associated with the metabolism and dynamics of the blood vessels that ultimately cause the BOLD signal. This is essential information needed to meaningfully couple a network model with a BOLD model. The issue of the neurovascular coupling remains unsolved and is an active area of research (Vanni et al., 2015; Buxton, 2021; Howarth et al., 2021). Recently, it has been proposed that cerebral blood flow (CBF) and cerebral metabolic rate of oxygen (CMRO2) may be driven separately by distinct neural processes (Buxton, 2012, 2021). As these variations are not captured by the classic Balloon model implementations in current tools, researchers need more flexible tools that allow them to define their own BOLD models.

The neural simulator ANNarchy is primarily concerned with models ranging from the mesoscopic to the microscopic level that simulate biological neurons as single units and can thus account for more detailed processes, which can include different ionic membrane currents and account for the dynamics of specific classes of real neurons (Humphries et al., 2009; Corbit et al., 2016; Goenner et al., 2021). Thus, ANNarchy allows to consider various neural processes for the implementation and investigation of neurovascular coupling. The BOLD monitor not only allows linking predefined BOLD models (e.g., the Balloon model variants, Stephan et al., 2007) to a rate-coded or spiking network model but also gives the user freedom in defining the neurovascular coupling and the BOLD model itself, allowing to investigate different hypotheses regarding the link between neural processes and BOLD signals.

In this article, we present the rationale, implementation and use of the BOLD monitor in ANNarchy. We first demonstrate the use of the classic Balloon model as a BOLD model for the BOLD monitor. We then demonstrate how to create a user-defined BOLD model. Finally, using a simple network model as an example, we demonstrate how the BOLD monitor can be used to compare various hypotheses about neurovascular coupling in simulation.

2. THE BALLOON MODEL

2.1. The Classic Balloon Model

The Balloon model was originally designed by Buxton et al. (1998). It describes the changes in the BOLD signal of a tissue region, often called region of interest (ROI), as a function of normalized CBF (f_{in}). According to this model, the BOLD signal corresponds to the sum of the extravascular and intravascular signal resulting from the normalized total deoxyhemoglobin content (q) and the normalized venous volume fraction (v). The normalized venous volume fraction is described as a balloon that expands with increasing inflow and slowly recovers after a stimulus. The normalized deoxyhemoglobin content is determined by the dynamics of the volume fraction and the blood oxygen extraction fraction (E), whose behavior is based on the oxygen limitation model (Buxton and Frank, 1997).

Friston et al. (2000) extended the Balloon model so that it can be used to simulate BOLD signals using network models. The extension included a neurovascular coupling component that links the normalized CBF of the Balloon model to simulated neuronal activity. Based on this extension, the normalized CBF is modeled as a damped oscillator that is stimulated by neuronal activity. This extension allows the Balloon model to be used to simulate a change in the BOLD signal due to a change in some type of simulated neuronal activity (hereafter, more generally referred to as input signal). In this form, the Balloon model has been used in several studies to compute simulated BOLD signals from network models (Friston et al., 2003; Smith et al., 2011; Deco and Jirsa, 2012; Van Hartevelt et al., 2014; Bennett et al., 2015; Maith et al., 2021). The individual components of the extended Balloon model and their dynamics following a rectangular input signal change are shown in **Figure 1A**.

Different values for the parameters and even variations of some equations of the model can be found in the literature. We use a version from Stephan et al. (2007) with a non-linear BOLD equation with revised coefficients for our default BOLD monitor.

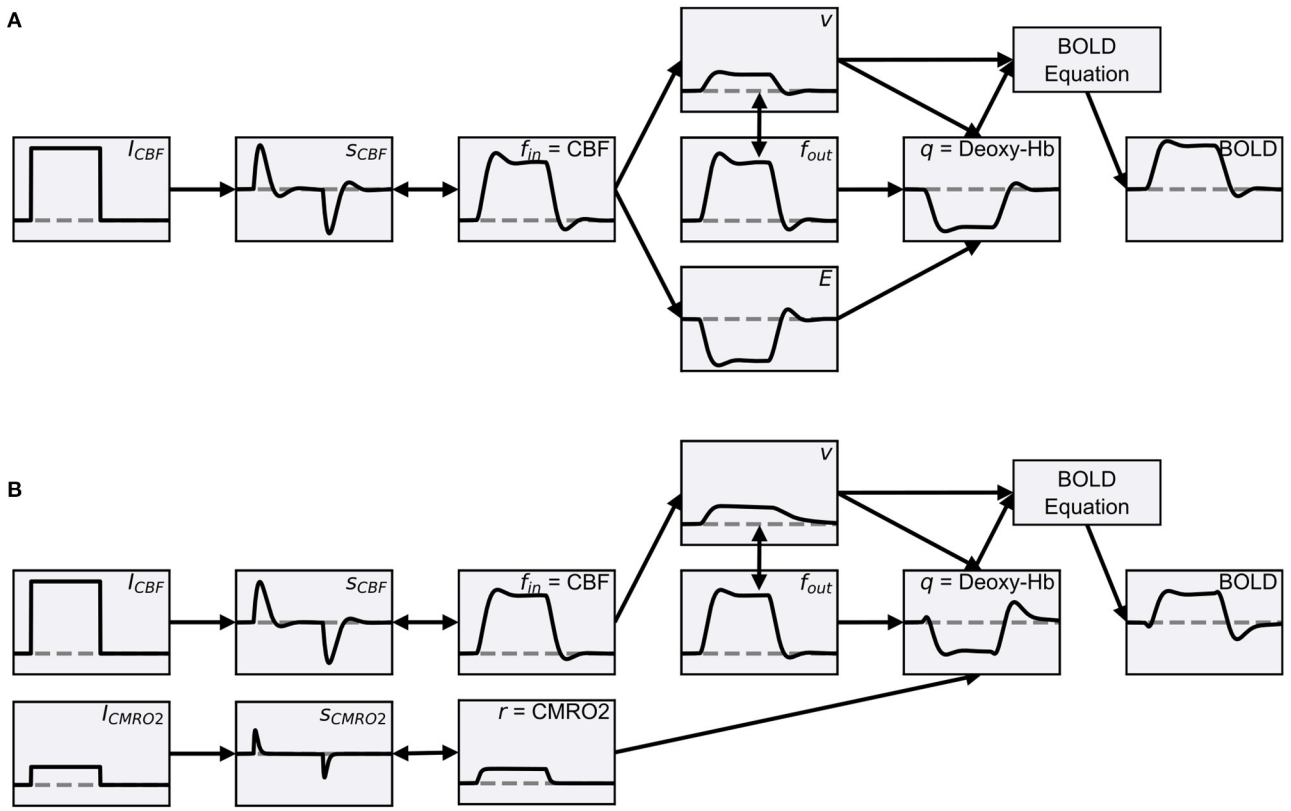


FIGURE 1 | (A) Schematic overview of the classic Balloon model (Buxton et al., 1998) with the neurovascular coupling extension of Friston et al. (2000). The data was simulated using ANNarchy's default BOLD monitor which recorded a single artificial neuron, whose activity (the source variable for the BOLD monitor) was manually set. The activity of the neuron and thus the input signal (I_{CBF}) were manually increased from zero to 0.2 for 20 s. The normalized CBF (f_{in}) changes as a function of the CBF-driving signal (s_{CBF}), which is subject to negative feedback from itself and f_{in}. f_{in} is coupled to the blood oxygen extraction fraction (E) and increases the normalized volume fraction of the venous compartment (v), which behaves like a balloon and decreases with outflowing blood (f_{out}). The normalized total deoxyhemoglobin content (q) increases by oxygen extraction of inflowing blood and decreases with outflowing deoxyhemoglobin-containing blood. Finally, relative changes of the BOLD signal are calculated. The gray horizontal lines correspond to one for the quantities normalized to their baseline in the Balloon model (f_{in}, f_{out}, v, q). For I_{CBF}, s_{CBF} and *BOLD* the gray horizontal lines correspond to zero and for E to E_0. **(B)** Balloon model with parallel driven CBF and CMRO2. Instead of coupling the increase in q with f_{in} *via* E, the normalized CMRO2 (r) is used directly (see Buxton et al., 2004), which is driven by a second input signal (I_{CMRO2}) like the normalized CBF (increased to 0.05 for 20 s). Further, f_{out} is described by the equations of Buxton et al. (2004) which causes v to decrease slower. Besides these changes, the processes are the same as in **(A)** and the plots of the same quantities have the same limits in **(A,B)**. The equations of both models can be found in **Supplementary Sections 2, 4.2**.

The other versions of Stephan et al. (2007) are also implemented in ANNarchy and available as alternatives. All equations are summarized in **Supplementary Section 2**. The implementation of the default model in ANNarchy is described in Section 3.4.

2.2. The Two-Input Balloon Model

In the classic Balloon model, CBF and CMRO2 are tightly coupled. The greater increase in CBF compared to CMRO2 in response to a stimulus is explained by the oxygen limitation model (Buxton and Frank, 1997). This model is based on the assumptions that oxygen coming from the capillaries is completely metabolized in the tissue and that all brain capillaries are perfused at rest. As a consequence, an increase in CMRO2 would only be possible by increasing the transport of oxygen from the capillaries to the tissue, and an increase in CBF would be accompanied by an increase in capillary blood velocity. Because an increase in CBF increases the available oxygen in the capillaries, but also decreases the fraction of oxygen extracted from the capillaries, an increase in CMRO2 (i.e., oxygen transport from the capillaries to the tissue) requires a disproportionate increase in CBF (for further details, see Buxton and Frank, 1997).

However, in recent years, it has been proposed that CBF and CMRO2 are driven in parallel by different sources rather than being tightly coupled (Buxton, 2012, 2021; Buxton et al., 2014). Recently, Buxton (2021) has put forward a new theory, based on the thermodynamics of metabolism, that could explain why CBF needs to increase more than CMRO2 in response to a stimulus and has proposed that CBF and CMRO2 are both driven in parallel in a feed-forward manner. The open question here is by which neural signals CBF and CMRO2 are driven. One suggestion is that CMRO2 is tightly coupled to the energy consumption of neurons, whereas CBF is controlled

by vasodilatory signals. These vasodilatory signals are not necessarily coupled to energy consumption and are caused, for example, by activated astrocytes (Buxton, 2012; Howarth et al., 2021). Network models, in which a wide variety of populations can be simulated and manipulated in a controlled manner, may be useful in investigating this question. Therefore, not only the classic Balloon model with tightly coupled CBF and CMRO2 can be used in our BOLD monitor, but also user-defined BOLD models, potentially using more than one input signal from the network model.

We demonstrate how to define BOLD models with multiple input signals for the BOLD monitor in ANNarchy by implementing a modified version of the Balloon model where CBF and CMRO2 are driven in parallel by separate input signals (hereafter referred to as two-input Balloon model). For simplicity, in the two-input Balloon model, we describe both, the normalized CBF and CMRO2, as damped oscillators similar to the normalized CBF in the classic Balloon model version of Friston et al. (2000). Equal input signals elicit responses with equal amplitudes for the normalized CBF and CMRO2. Thus, the coupling between CBF and CMRO2 is determined by the coupling of the two input signals. **Figure 1B** demonstrates how the individual components of the two-input Balloon model change during stimulation. Compared to the normalized CBF, the normalized CMRO2 responds faster to a changing input signal and without an overshoot or undershoot. The faster response allows for an initial dip in the BOLD signal. For the transformation from normalized CBF and CMRO2 to q and v, we use the Balloon model equations from Buxton et al. (2004). This is a slightly modified version of the classic Balloon model, which additionally considers viscoelastic effects causing the venous volume fraction to lag behind its steady-state relation with the outflow during transient changes. Thus, a post-stimulus undershoot in the BOLD signal is caused by the undershoot of the CBF (based on the damped oscillator modeling approach) as well as by the slow recovery of the venous volume fraction (based on the viscoelastic effects). Finally, the change in the BOLD signal is computed by the non-linear BOLD equation with revised coefficients from Stephan et al. (2007). A more detailed description including the equations of the two-input Balloon model summarized here can be found in **Supplementary Section 4.2**.

3. BOLD MONITOR

3.1. ANNarchy Neural Simulator

The ANNarchy neural simulator is intended for the simulation of network models at the single-unit level using rate-coded and spiking neuron models. The equation-based interface of ANNarchy allows a flexible and easy implementation of network models by defining equations describing the dynamics of specific neuron types in so called neuron models and equations defining synaptic transmission dynamics (e.g., plasticity) in so called synapse models (Vitay et al., 2015). For efficiency, the model description is transformed into optimized C++ code, optionally using parallel programming frameworks such as openMP for multi-core CPUs or CUDA for GPUs (Dinkelbach et al., 2019). An earlier version of the BOLD monitor in ANNarchy relied on the normalization of pre-synaptic activity and was used in Maith et al. (2021). This implementation was limited to one specific BOLD model and allowed only a few parameter variations, unlike the version presented here. All the simulations in this work use the version 4.7.0.1 of the neural simulator ANNarchy. All references to neurons, populations, synapses, BOLD signals and other neural quantities and data in the following sections refer to simulated values from a network model.

3.2. General Concept

BOLD models, for example the Balloon model (Buxton et al., 1998) or the Davis model (Davis et al., 1998), are based on signals that characterize the dynamics of an entire ROI, such as the change in the normalized CBF or CMRO2 (**Figure 2**, f_{in}, r). To combine such a BOLD model with a network model, it is necessary to bridge the gap between these ROI-wide signals and the individual components of the ROI in the network model (e.g., multiple populations, individual neurons). In this section, we will focus on the processes necessary to obtain the input signals for a BOLD model from a ROI that represents part of a network model consisting of multiple populations. **Figure 2** shows the general functionality of the BOLD monitor in ANNarchy.

First, the populations of the network model that are part of the ROI for the BOLD computation have to be specified and instantiated. In the example shown in **Figure 2**, the ROI consists of two populations labeled pop1 and pop2. From the definition of their neuron models, variables must be selected or defined (hereafter referred to as source variables) which will be used to derive the input signals of the BOLD model. In **Figure 2**, two different source variables are defined: one variable var_{CBF} that causes the input signal of the CBF (I_{CBF}) and one variable var_{CMRO2} that causes the input signal of the CMRO2 (I_{CMRO2}). These source variables can correspond to any variables or combinations of variables present in the neuron models (membrane potential, firing rate, etc.).

After defining the ROI and the mapping between source variables in the neuron models and the input variables of the BOLD model, the BOLD monitor implements four preprocessing steps. First, the source variables are averaged over all the neurons for each population of the ROI, resulting in only one signal per population and source variable. This averaging is followed by an optional population-wide normalization that computes the relative deviation of the signal from a baseline value. The baseline corresponds to the mean of the raw averaged source variable signal calculated over a specified initial period. This normalization is useful when deviations from the resting-state are required as input signal in the BOLD model. After the optional normalization, the signals are scaled per population. By default, the signals of each population are scaled based on the ratio between the size of the population and the total number of neurons in the ROI. Thus, the larger a population, the greater its influence on the input variables of the BOLD model. Finally, the population signals are summed across all populations of the ROI, resulting in one input signal for each input variable of the BOLD model.

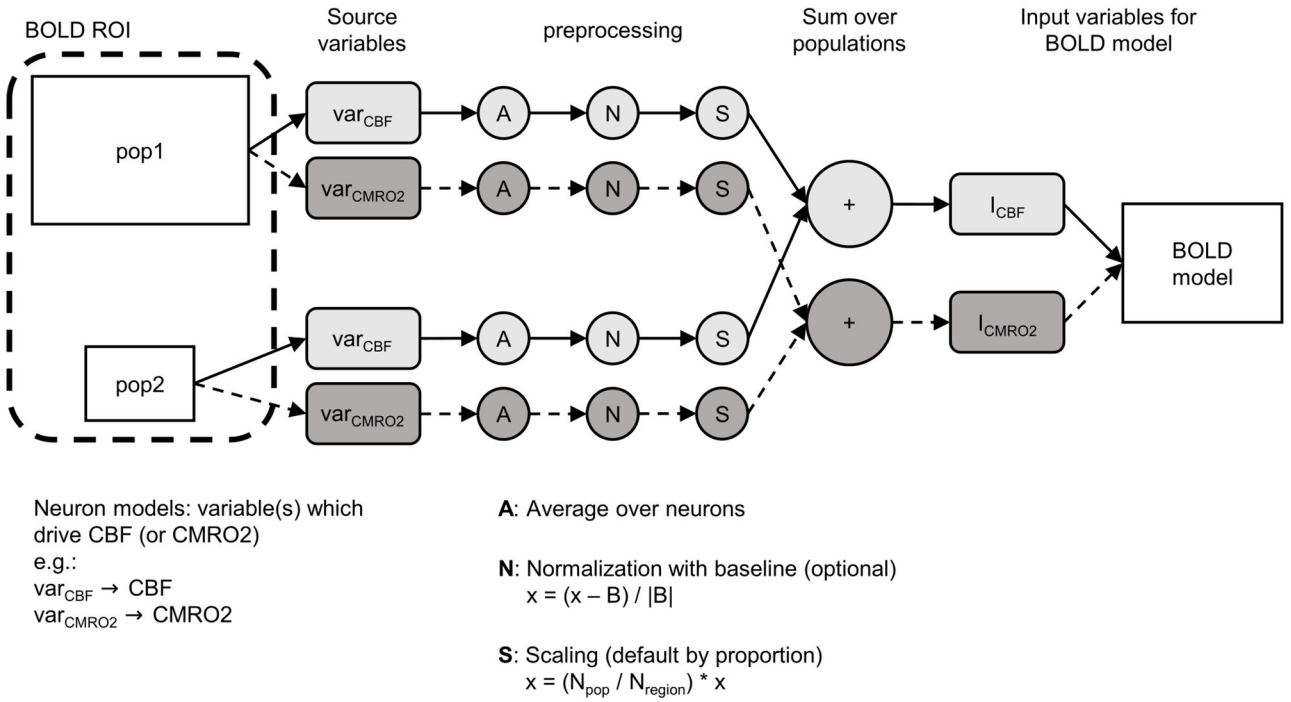


FIGURE 2 | Schematic overview of how the BOLD monitor calculates the input signals (here I_{CBF}, I_{CMRO2}) for a BOLD model (e.g., Balloon model). The ROI from which the BOLD signal should be calculated can consist of multiple populations (here pop1, pop2). In the neuron models of each population, the variables (var_{CBF}, var_{CMRO2}) have to be defined, which are used to calculate the input signals (referred to as source variables). Three preprocessing steps are applied to the source variables per population: (1) averaging over all neurons of the population, (2) optional normalization, (3) scaling by proportion of the population in the ROI. Finally, the signals resulting from the preprocessed source variables are summed across all populations and fed into the BOLD model as input signals. x, the averaged signal; B, baseline; N_{pop}, size of the population; N_{region}, total number of neurons in the ROI.

3.3. A Simple Example

This section describes a minimal example demonstrating the use of the BOLD monitor in the ANNarchy framework. ANNarchy modules and the BOLD extension must first be imported:

```
from ANNarchy import setup, Population, Izhikevich,
    compile, simulate
from ANNarchy.extensions.bold import BoldMonitor,
    balloon_RN
```

The evaluation of equations is performed with the forward Euler numerical method using a fixed time grid of step *dt* (in ms):

```
setup(dt = 1.0)
```

Two populations, both composed of 100 Izhikevich spiking neurons are then created (line 4, 5). The Izhikevich neuron model is part of the standard models pre-implemented in ANNarchy, with equations and parameters derived from Izhikevich (2003). Initially, the baseline activity in both populations is defined by setting their *noise* variables to 5.0 (line 7). The term *noise* refers to an internal variable of the pre-implemented Izhikevich neuron model in ANNarchy which simply determines a baseline current in the membrane potential equation.

```
pop0 = Population(100, neuron=Izhikevich)
pop1 = Population(100, neuron=Izhikevich)

pop0.noise = 5.0; pop1.noise = 5.0
```

To keep the example simple and still have a modulation in the source variable of the BOLD monitor, the baseline activity (the *noise* variable) is varied during the simulation to mimic the effect of external inputs. The mean-firing rate *r* of the individual neurons is used as the source variable for the computation of the BOLD signal. As the computation of this value requires an additional overhead, it must be enabled explicitly. The time window for the averaged activity is set to 100 ms:

```
pop0.compute_firing_rate(window=100.0)
pop1.compute_firing_rate(window=100.0)
```

The BOLD monitor is then created and initialized (line 10–16). The populations in the ROI have to be assigned in the *populations* argument in form of a list of or a single population (line 11). The desired BOLD model can be optionally defined in the argument *bold_model* by assigning the corresponding BOLD model object (line 12). The BOLD model can be either one of the built-in BOLD models provided by the module or user-defined as we will demonstrate in Section 3.4. The default BOLD model is the built-in implementation *balloon_RN* containing the Balloon model with revised coefficients and a non-linear BOLD equation (described in Section 2, implementation shown in Section 3.4).

The mapping between the source variables of the populations (here mean-firing rate *r*) and the input signals of the BOLD model (referred to as input variables, here *I_CBF*) has to be defined in the *mapping* argument by providing a dictionary for each input variable-source variable pair (line 13).

A time window relevant to the normalization of the source variables can be optionally defined (in ms, line 14), whose purpose we explain in Section 4.2. By default, no normalization is performed.

Finally, the variables of the BOLD model which should be recorded during the simulation can be optionally assigned in the *recorded_variables* argument (line 15) as a string or list of strings. All variables of the BOLD model can be recorded. By default, the output variable of the BOLD model (here the variable *BOLD*) defined in the BOLD model implementation (see Section 3.4) is recorded.

```
10 m_bold = BoldMonitor(
11     populations=[pop0, pop1],
12     bold_model=balloon_RN,
13     mapping={"I_CBF": "r"},
14     normalize_input=2000,
15     recorded_variables=["I_CBF", "BOLD"]
16 )
```

The C++ code representing the model (network model and BOLD monitor) can now be generated and compiled:

```
17 compile()
```

The last part of this section describes a sample simulation to demonstrate the BOLD recording on our simple example. A short simulation period (1,000 ms, line 19) ensures that the network reaches a stable state, which is necessary for a meaningful baseline calculation (required for the normalization outlined in Section 4.2). The recording of BOLD signals is started (line 22) and the simulation is run for 5 s (line 25). After this, the baseline activity (*noise* variable) of half of the recorded neurons (one population, pop0) is increased for 5 s (lines 26, 27) and afterwards set back to the previous value (line 28, 29).

```
18 # Ramp up time
19 simulate(1000).
20
21 # Start recording
22 m_bold.start()
23
24 # Manipulate the noise for half of the neurons
25 simulate(5000)      # 5s with low noise
26 pop0.noise = 7.5
27 simulate(5000)      # 5s with higher noise
28 pop0.noise = 5
29 simulate(10000)     # 10s with low noise
30
31 # Retrieve the BOLD recordings
32 bold_recordings = m_bold.get()
```

This leads to an increased mean firing rate in the recorded area and consequently to a BOLD signal response as depicted in **Figure 3**. The figure shows that the increase of the *noise* variable in *pop0* leads to an increase in the mean-firing rate, which is the source variable for the BOLD monitor (**Figure 3A**, blue line). This increase of activity results in an increase of the input signal (input variable *I_CBF*) of the BOLD model depicted in **Figure 3B**, consequently leading to an increase of the BOLD signal depicted in **Figure 3C**. After resetting the *noise* variable, the firing rates of both populations reach again the same level, which reduces the input signal of the BOLD model as well as the resulting BOLD signal.

3.4. BOLD Model Definition

In the previous example, the default Balloon model (*balloon_RN*) was used as the BOLD model, but ANNarchy allows users to create their own BOLD model by defining a *BoldModel* object representing the desired equations. We describe the definition of a *BoldModel* object using the BOLD model *balloon_RN* (described in Section 2, applied in Section 3.3) as an example. This BOLD model is implemented as follows:

```
balloon_RN = BoldModel(                                                    1
    parameters = """                                                       2
        phi       = 1.0         ;   kappa     = 1/1.54                   3
        gamma     = 1/2.46      ;   E_0       = 0.34                     4
        tau       = 0.98        ;   alpha     = 0.33                     5
        V_0       = 0.02        ;   v_0       = 40.3                     6
        TE        = 40/1000.    ;   epsilon   = 1.43                     7
        r_0       = 25.         ;   second    = 1000.0                   8
    """,                                                                   9
    equations = """                                                        10
        # CBF input                                                        11
        I_CBF           = sum(I_CBF)                                       12
        ds/dt           = (phi * I_CBF - kappa * s -                       13
    gamma * (f_in - 1))/second
        df_in/dt        = s / second                                       14
                                  : init=1, min=0.01
                                                                           15
        # Balloon model                                                    16
        E               = 1 - (1 - E_0)**(1 / f_in)                        17
                                  : init=0.3424
        dq/dt           = (f_in * E / E_0 - (q / v) *                      18
    f_out)/(tau*second)            : init=1, min=0.01
        dv/dt           = (f_in - f_out)/(tau*second)                      19
                                  : init=1, min=0.01
        f_out           = v**(1 / alpha)                                   20
                                  : init=1, min=0.01
                                                                           21
        # Revised coefficients                                             22
        k_1             = 4.3 * v_0 * E_0 * TE                             23
        k_2             = epsilon * r_0 * E_0 * TE                         24
        k_3             = 1.0 - epsilon                                    25
                                                                           26
        # Non-linear BOLD equation                                         27
        BOLD            = V_0 * (k_1 * (1 - q) + k_2 *                     28
    (1 - (q / v)) + k_3 * (1 - v))
    """,                                                                   29
    inputs = "I_CBF",                                                      30
    output = "BOLD"                                                        31
)                                                                          32
```

A *BoldModel* object requires a *parameters* argument (line 2), which is a string defining all constants of the BOLD model in a key-value pair notation, i.e., a parameter name on the left and the initialization value on the right side of the assignment operator.

The *equations* argument (line 10) describes all time-dependent variables, defined either by regular equations or ordinary differential equations evaluated on a fixed time grid. Note that parameters resulting from the combination of other parameters

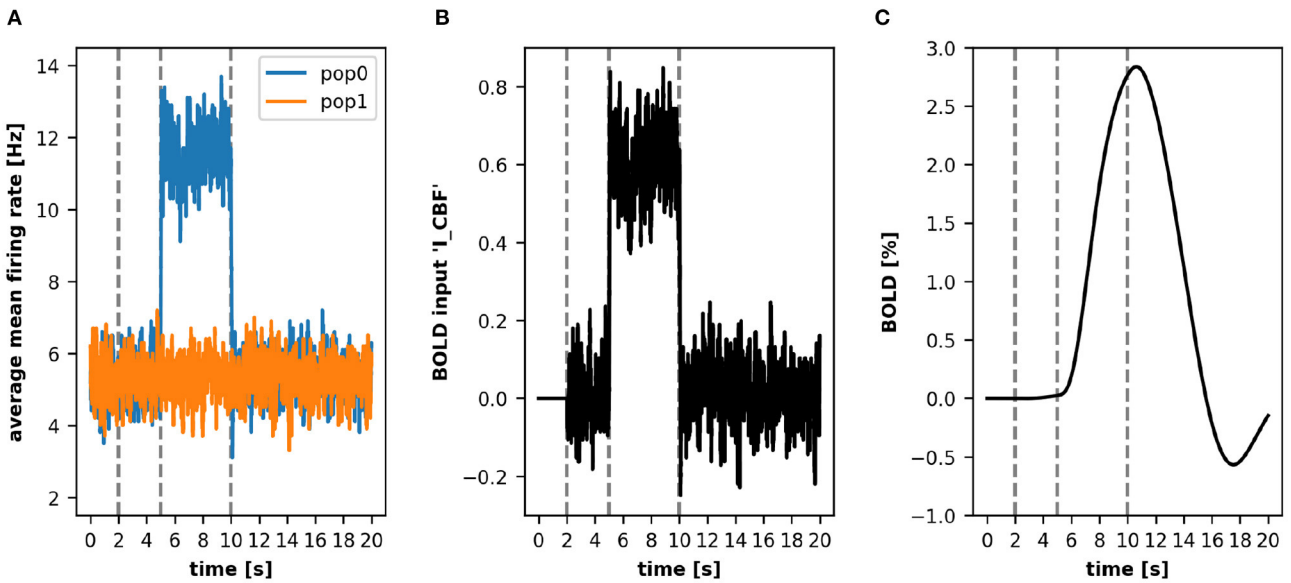


FIGURE 3 | A simple simulation using the ANNarchy BOLD monitor. In this example, we obtain a BOLD signal from two populations *pop0* and *pop1*. Both populations contribute their mean-firing rate *r* **(A)** to the BOLD model which we defined in Section 3.4. After 5 s of simulation, the *noise* variable in *pop0* is increased which leads to a higher mean-firing rate. This increases the input signal *I_CBF* **(B)** and consequently the computed BOLD signal **(C)**. For clarity, the vertical lines depict three relevant time points (left to right): end of baseline period, time point of increased *noise* variable, time point of reset *noise* variable.

can also be defined here (in this example *k_1*, *k_2*, *k_3*). In the case of a regular equation, the variable name is on the left side and the update performed in each step on the right side. If the update is defined by a differential equation, the left side needs to contain a $\frac{d[var]}{dt}$ symbol. To limit the range of values taken by a variable, the *min* and *max* keywords can be used. The initial value for variables is 0.0 by default, but it can be changed by providing an *init* keyword.

The *inputs* argument (line 30) specifies which input signals are expected by the BOLD model. It consists of a single string or a list of strings. These variables can be accessed in the BOLD model definition by using *sum(NAME)* in the *equations* argument, where *NAME* corresponds to the name of the variable (here *I_CBF*, line 12).

Finally, in the *output* argument (line 31), one output variable of the BOLD model is defined, which is automatically recorded by the BOLD monitor. In the following implementation example and all other BOLD models implemented in ANNarchy mentioned in this work, this default output variable corresponds to the BOLD signal (variable *BOLD*), which is also the default value for the *output* argument (here only defined for demonstration purposes).

The *balloon_RN* model is one of the four pre-implemented BOLD models (*balloon_RN*, *balloon_RL*, *balloon_CN* and *balloon_CL*, Stephan et al., 2007) and therefore does not need to be defined by the user (but its parameters can be changed dynamically). With the *BoldModel* object, the user can implement new models with the same equation-based interface. For example, a user might want to additionally implement the Davis model (Davis et al., 1998) described by Equation 1 to calculate the change of the BOLD signal $\Delta BOLD$ from normalized CBF f and CMRO2 r.

$$\Delta BOLD = M\left[1 - f^{\alpha}\left(\frac{r}{f}\right)^{\beta}\right] \tag{1}$$

Here, M, α, and β are additional parameters of the Davis model. In the *BoldModel* above, the normalized CBF is already defined (f_{in}). Thus, only the calculation of the normalized CMRO2 (r) must be added. This could be done with the term $f_{in} \cdot \frac{E}{E_0}$ (see also Buxton et al., 2004). The following code demonstrates how the previous *BoldModel* could be extended to additionally compute the normalized CMRO2 (r, line 34) and the Davis model BOLD signal (line 35) in the *equations* argument:

```
    ...                                                                    29
    BOLD = V_0 * (k_1 * (1 - q) + k_2 * (1 - (q /                          30
v)) + k_3 * (1 - v))
                                                                           31
    # Davis model                                                          32
    r = f_in * E / E_0                                                     33
                    : init=1, min=0.01
    BOLD_Davis = M * (1 - f_in**alpha_D * (r /                             34
f_in)**beta)
""",                                                                       35
...                                                                        36
```

This way, a custom *BoldModel* is obtained, where the BOLD signal is additionally calculated according to the Davis model and the modified signal ($BOLD_{Davis}$) can additionally be recorded. In addition, the parameters of the Davis model would have to be added to the *parameters* argument, which we have

not shown explicitly (but see **Supplementary Section 4.3** for a full implementation).

4. EXAMPLE USE CASES

4.1. Model Description

In this section, we implement a simple network model of a cortical microcircuit (hereafter referred to as microcircuit model) to further demonstrate use cases of the BOLD monitor. The microcircuit model consists of a population of excitatory neurons and a population of inhibitory interneurons. As neuron models, we use a regular spiking cortical neuron model for the excitatory population (corE) and a fast-spiking cortical interneuron model for the inhibitory population (corI), both introduced in Izhikevich (2007). The two populations receive excitatory inputs from another population whose neurons randomly emit spikes such that their inter-spike intervals correspond to a Poisson process (hereafter referred to as Poisson neurons). The structure of the microcircuit model is shown in **Figure 4A**. The projections of the microcircuit model include feed-forward excitation (Poisson neurons → corE), feed-forward inhibition (Poisson neurons → corI → corE), and feedback inhibition (corE → corI → corE). The ratio between excitatory neurons and inhibitory interneurons is 4:1, as found, for example, for the visual cortex (Beaulieu et al., 1992; Potjans and Diesmann, 2014). The equations and parameters of the microcircuit model can be found in **Supplementary Section 3**.

Each neuron receives synaptic input from 10 random neurons in the pre-synaptic population for each projection. Following our previous modeling approaches (Baladron et al., 2019; Goenner et al., 2021; Maith et al., 2021), we model synaptic inputs as conductance-based synapses in our neuron models. Therefore, the synaptic currents (which drive the membrane potential of the neurons) are proportional to the product of a voltage difference (between the membrane potential and the synaptic reversal potential) and a conductance variable representing the spike input of the corresponding synapse (see **Supplementary Section 3** for equations). We model only two different types of conductance-based synapses, excitatory synapses (AMPA) and inhibitory synapses (GABA). The conductance variables of the synapses are instantaneously increased by a fixed value (by the weight of the synaptic connection) for each incoming spike and otherwise decay exponentially to zero with a time constant of 10 ms.

A conductance greater than zero causes a synaptic current that drives the membrane potential toward the reversal potential associated with the synapse (0 mV for AMPA synapses and −90 mV for GABA synapses). All synaptic weights are drawn from a log-normal distribution and scaled by a factor for each projection during model initialization. The weights and scaling factors were optimized to replicate distributions from excitatory post-synaptic potentials (Song et al., 2005) and firing rates (Buzsáki and Mizuseki, 2014) with the microcircuit model (see **Figure 4B**). Further details about obtaining the distributions and optimizing the parameters can be found in **Supplementary Section 3.3**.

Although the use of neuron models mimicking spiking patterns of real cortical neurons and tuning the parameters to replicate experimental data can provide more realistic network models (see e.g., Humphries et al., 2006; Günay et al., 2008; Pospischil et al., 2008; Goenner et al., 2021), the microcircuit model presented here only aims at demonstrating the application of the BOLD monitor and not at replicating any particular experimental data. To keep the model simple, we chose a network model with two spiking populations and multiple excitatory and inhibitory projections. No particular functional processing takes place in this microcircuit model, as it consists of only two small homogeneous populations, the connectivity is random and synaptic plasticity, important neurotransmitters such as NMDA, the effect of neuromodulators and potential dynamic changes in activity were not taken into account during construction. However, the applicability of the BOLD monitor to larger-scale network models is demonstrated in Section 4.4.

4.2. Normalization for Resting-State Activity

We first demonstrate the effect of baseline normalization in the BOLD monitor using the microcircuit model. To do so, we simulate a brief stimulus presentation corresponding to studies of the event-based BOLD response (Glover, 1999; Serences, 2004) by briefly increasing the mean firing rate of the Poisson neurons and meanwhile recording the BOLD response.

All simulations start with an initialization period of 2 s to allow the microcircuit model to enter its steady-state. After that, the recordings are started. A 10-s resting-period is simulated, after which the mean firing rates of the Poisson neurons are increased by a factor of five for 100 ms (hereafter referred to as stimulus pulse). Finally, another post-stimulus resting-period is simulated until a total simulation time of 25 s. This procedure is performed for 40 different random microcircuit model initializations (each with different seeds producing different synaptic contacts, weights, and mean firing rates of Poisson neurons). Additionally, we run 40 simulations without a stimulus pulse, in which only a 25 s resting-period is simulated for comparison.

The BOLD response is recorded simultaneously using two differently initialized BOLD monitors. Both BOLD monitors use the default BOLD model (*balloon_RN*) shown in Section 3.4 and determine the BOLD signal of the ROI which comprises both the corE and corI populations. The source variable for the BOLD monitor is the synaptic activity of the neurons normalized by the number of afferent connections, which has already been used and described in Maith et al. (2021). The key difference between the two BOLD monitors is the baseline normalization. One BOLD monitor uses no baseline normalization and the other BOLD monitor uses a baseline computed over the first 5 s after the 2-s initialization period.

Figure 5 shows the recorded variables of the BOLD model: the input variable (I_{CBF}) of the BOLD model and the resulting BOLD signal. Although the response of the microcircuit model to the stimulus pulse can be clearly seen in the I_{CBF} of both BOLD monitors, an important difference is that the I_{CBF} of the BOLD monitor without baseline normalization has an offset greater than zero, while the I_{CBF} with baseline normalization fluctuates around zero. It is also noticeable that I_{CBF} with baseline

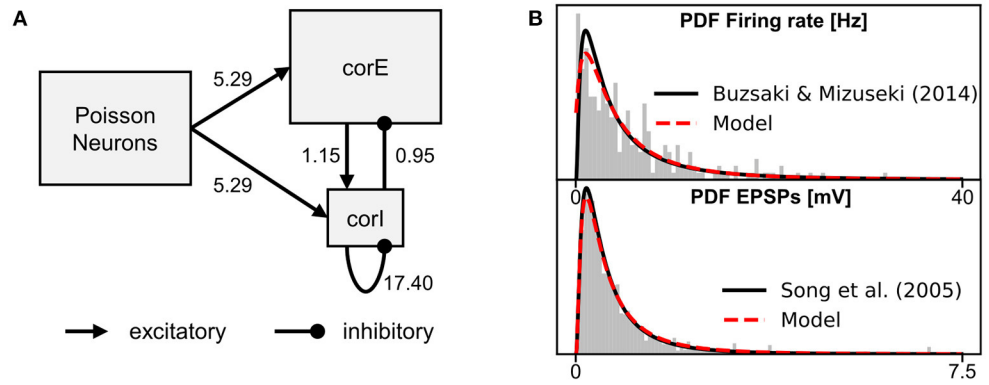


FIGURE 4 | Overview of the microcircuit model. **(A)** The structure of the microcircuit model. Rectangles represent the populations and arrows the projections. The numbers at the projections indicate the factors which scale the weights of each projection. **(B)** Probability density function (PDF) and histogram of the firing rate distribution of the neurons of the corE and corI populations (top) and PDF and histogram of the excitatory post-synaptic potentials (EPSPs) evoked by single spikes in the excitatory and inhibitory neurons (bottom). The weights for the projections of the microcircuit model were drawn from a weight distribution which was tuned to generate the EPSPs distribution of Song et al. (2005) (indicated in black). The weights of each projection were further scaled so that the corE and corI populations produce the firing rate distribution of Buzsáki and Mizuseki (2014) (indicated in black). The scaling factors were optimized, for more details see **Supplementary Section 3.3**.

normalization is zero in the first 5 s. This is because the input variable for the BOLD model is not calculated during the time in which the baseline for normalization is determined. In the normalized CBF signal and the BOLD signal, one can clearly see the effect of baseline normalization on the Balloon model dynamics. The response to the stimulus pulse is much more pronounced for the BOLD monitor with baseline normalization. Without baseline normalization, the normalized CBF signal and BOLD signal at rest have an offset greater than zero, whereas with baseline normalization, the signals fluctuate around one and zero, respectively.

The CBF and BOLD signals are defined in the Balloon model relative to their value at rest (normalized CBF and relative change of BOLD). Therefore, the normalized CBF signal or the BOLD signal should only deviate from one or zero, respectively, when the underlying system deviates from its resting-state. For models with resting-state activity, we recommend using the baseline normalization of the BOLD monitor when using the Balloon model.

4.3. The Effect of Different Source Variables

One important motivation for developing the BOLD monitor is to provide a simple way to flexibly adjust both the source variables and the BOLD model itself. In Section 3.4, we have already shown how to implement a user-defined BOLD model. Here, we also want to show the possibility to use different variables of the neurons as source variables. The underlying neural processes influencing CBF and CMRO2, and thus ultimately the BOLD signal, are still rather unclear (Howarth et al., 2021). Many different hypotheses and modeling approaches can be found in the literature (Smith et al., 2011; Van Hartevelt et al., 2014; Bennett et al., 2015; Heikkinen et al., 2015; Schmidt et al., 2018). The flexible BOLD monitor in ANNarchy allows us to easily create and compare BOLD models implementing different hypotheses on spiking or rate-coded network models. In this section, we demonstrate this by implementing six different hypotheses using our microcircuit model. For each hypothesis, we add a different BOLD monitor to the microcircuit model, each with different source variables. The six different BOLD monitors are summarized in **Table 1**. The source code for adding them to the microcircuit model can be found in **Supplementary Section 4**. Note that the simulated BOLD signals are not compared with experimental data, so we do not make any statements about the validity of the hypotheses. Such an analysis would require an extensive underlying network model, tailored to the brain region under investigation.

We again use the stimulus pulse simulation from Section 4.2 to compare the different BOLD signal responses (see **Figure 6**). The first three hypotheses are based on previous studies that used the classic Balloon model. Thus, we also use the classic Balloon model (BOLD model *balloon_RN*) for the BOLD calculation, which includes a single CBF-driving input signal (see **Figure 1A**) whose source variable we vary for each hypothesis. The first hypothesis we implement is that the CBF or the BOLD signal is driven by the total synaptic activity of the neurons (as in Van Hartevelt et al., 2014; Schmidt et al., 2018; Maith et al., 2021). To implement this, we use the normalized synaptic activity as the source variable of the BOLD monitor (BOLD monitor A), as previously in Section 4.2. The second hypothesis we implement is that the CBF or the BOLD signal is driven only by the excitatory (glutamatergic) synaptic activity (similar to Heikkinen et al., 2015). For this, we use the conductance variable of the excitatory synapses of the neurons as source variable for the BOLD monitor (BOLD monitor B). The third hypothesis is that the CBF or the BOLD signal is driven by the neuronal output of the neurons, for example, the mean firing rate (as in Smith et al., 2011; Bennett et al., 2015). Thus, for this BOLD monitor (BOLD monitor C), we use the mean firing rate of the neurons

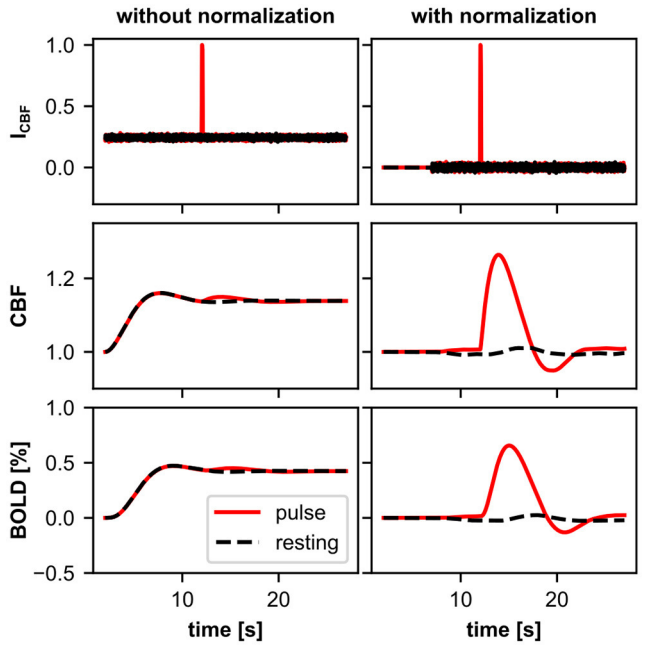


FIGURE 5 | Recordings from two BOLD monitors with (right) and without (left) baseline normalization. Shown are the averaged recordings of 40 resting-state simulations (black) and 40 simulations with a 100 ms stimulus pulse (red). Both BOLD monitors use the same source variable of the same underlying microcircuit model. For the BOLD monitor with baseline normalization, the input signal to the Balloon model (I_{CBF}) corresponds to the relative change in the signal from the source variable, thus fluctuates around zero. Whereas, without baseline normalization, the input signal has an offset greater than zero. Thus, with baseline normalization only, the normalized CBF signal and the BOLD signal of the Balloon model during the resting-state are approximately one and zero, respectively, corresponding to the definition of the Balloon model. The response to the stimulus pulse is more pronounced with baseline normalization. For visualization, I_{CBF} values are shown divided by their maximum value.

TABLE 1 | The input and source variables of the 6 different BOLD monitors of Section 4.3.

Monitor ID	BOLD model	Input variables	Source variables	
			corE	corI
A	*balloon_RN*	I_{CBF}	*syn*	
B	*balloon_RN*	I_{CBF}	g_{AMPA}	
C	*balloon_RN*	I_{CBF}	*r*	
D	*balloon_two_inputs*	I_{CBF}	$I_{AMPA} + 1.5\, I_{GABA}$	
		I_{CMRO2}	I_{AMPA}	
E	*balloon_two_inputs*	I_{CBF}	$I_{AMPA} + 1.5\, I_{GABA}$	
		I_{CMRO2}	I_{AMPA}	*r*
F	*balloon_two_inputs*	I_{CBF}	$I_{AMPA} + 1.5\, I_{GABA}$	
		I_{CMRO2}	$I_{AMPA}^{\frac{1}{3}}$	

If the source variables of a specific input variable are different for excitatory and inhibitory neurons (corE and corI populations), they are given separately for corE and corI. I_{CBF}, CBF-driving input; I_{CMRO2}, CMRO2-driving input; syn, normalized total synaptic activity; g_{AMPA}, conductance variable of AMPA synapse; r, neuron firing rate; I_{AMPA}, current caused by AMPA synapses; I_{GABA}, current caused by GABA synapses.

as source variable, as in Section 3.3. **Figures 6A–C** shows that the normalized CBF and BOLD responses vary for these three BOLD monitors with different source variables. The response based on the mean firing rates (**Figure 6C**) is the strongest, because the firing rates change more relatively to the resting-state compared to the two other source variables. However, the shape of the responses is almost identical.

As mentioned in Section 2.2, it has also been proposed that the CBF and CMRO2 are driven in parallel in a feed-forward manner. Therefore, for the following three BOLD monitors, we use the two-input Balloon model defined in Section 2.2 (*balloon_two_inputs*), which requires two input signals (I_{CBF}, I_{CMRO2}, see **Figure 1**). The source variables used to obtain I_{CBF} and I_{CMRO2} can be freely chosen from the neuron models of the corE and corI populations.

The first hypothesis considering CBF and CMRO2 being driven in parallel proposes that the CMRO2 is driven only by excitatory synaptic processes and that the CBF is driven by both excitatory and inhibitory synaptic processes (Buxton, 2012, 2021). To implement this hypothesis in BOLD monitor D, we define the current caused by AMPA synapses (I_{AMPA}) as the CMRO2-driving source variable, and the sum of I_{AMPA} and the current caused by GABA synapses (I_{GABA}) as the CBF-driving source variable. These source variables have to be additionally defined in the neuron models of the neurons of the corE and corI populations (see **Supplementary Section 4.1**).

The next hypothesis is similar, but additionally states that in inhibitory interneurons, energy consumption, and thus CMRO2, is driven by neuronal output rather than synaptic input (in contrast to excitatory neurons) (Howarth et al., 2021). To implement this in BOLD monitor E, the mean firing rate of the neurons rather than I_{AMPA} is defined as the CMRO2-driving source variable for the inhibitory interneurons of the corI population. For the excitatory neurons of the corE population, the same source variables are used as in the previous BOLD monitor.

Figures 6D,E show that the normalized CBF, CMRO2, and BOLD (relative change) responses of these two BOLD monitors are significantly different from the previous ones (with a single input). The BOLD signal shows a much stronger initial dip as CMRO2 increases much faster than CBF. There is little difference between the responses of the two BOLD monitors. The CMRO2 of BOLD monitor E is slightly lower because the firing rate of the inhibitory interneurons increases less than their synaptic current caused by AMPA synapses. However, because the inhibitory interneurons only contribute one-fifth to the input signal of the BOLD monitor (due to the ratio between corE and corI sizes), there is only a small difference from BOLD monitor D to E.

In the last BOLD monitor (**Figure 6F**), we use almost the same source variables as in BOLD monitor D. We only introduce an additional non-linear operation for the source variable driving CMRO2 by defining the current caused by the AMPA synapses, to the power of one third, as the source variable (instead of the current itself). As a result, energy consumption or CMRO2 no longer increases linearly with the current. Thus,

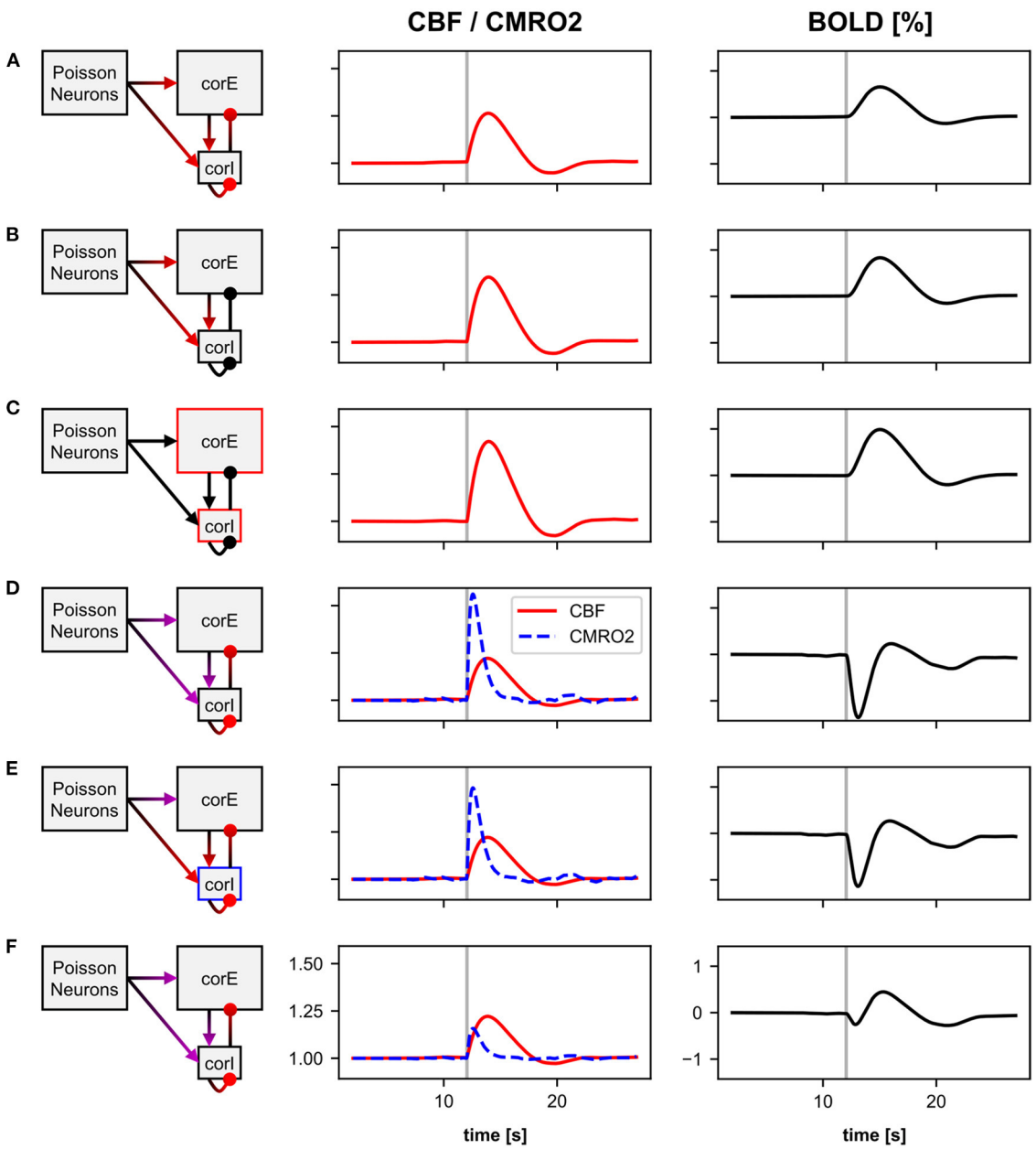


FIGURE 6 | Normalized CBF, CMRO2, and BOLD (relative change) responses of six different BOLD monitors including different source variables. The averaged recordings of 40 simulations with a 100 ms stimulus pulse are shown. The vertical gray bar in columns two and three indicate the time window of the active stimulus pulse. The left column illustrates, which compartments of the microcircuit model were used as source variables for the BOLD monitors. red, CBF-driving; blue, CMRO2-driving; purple, CBF and CMRO2-driving. For a more detailed description about the source variables see Section 4.3 and **Table 1**. The BOLD monitors **(A–C)** using the classic Balloon model with a single CBF-driving input signal. The BOLD monitors **(D–F)** using the two-input Balloon model defined in this work with two input signals driving CBF and CMRO2 separately.

we are still basically following the same general hypothesis (CMRO2 driven by AMPA synaptic processes, CBF driven by AMPA and GABA synaptic processes), but assuming different mathematical relationships for CMRO2. This change causes CMRO2 to increase much less due to the stimulus pulse, as shown in **Figure 6F**. Thus, the initial dip in the BOLD response is smaller than for the BOLD monitors D & E.

In summary, the BOLD monitor allows users to determine the BOLD signal based on individually chosen source variables. Without much effort, we can define different source variables and even compare different BOLD models (e.g., a model with two input variables). With the classic Balloon model, the BOLD response for our microcircuit model hardly differs for different source variables. Since all variables in the microcircuit model

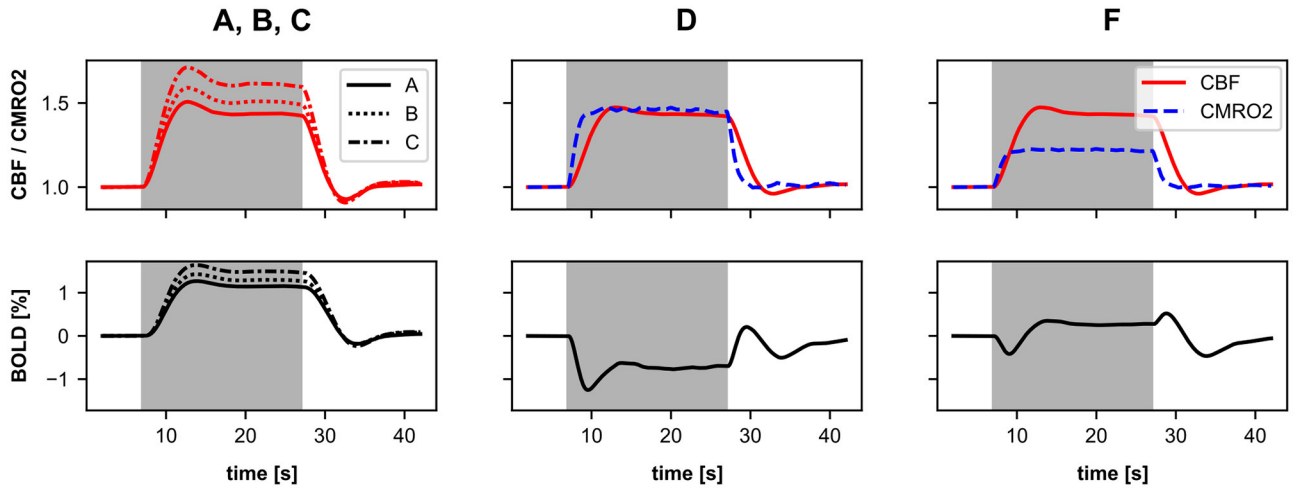


FIGURE 7 | Normalized CBF, CMRO2, and BOLD (relative change) responses of the different BOLD monitors (**A–C**, left), (**D**, mid), and (**F**, right) including different source variables. The averaged recordings of 40 simulations with a 20 s stimulus pulse are shown. The background highlighted in gray indicates the time window of the active stimulus. Here, the firing rate of the Poisson neurons is increased by a factor of 1.2 for 20 s. For a more detailed description about the source variables see Section 4.3 and **Table 1**. The BOLD monitors **(A–C)** using the classic Balloon model with a single CBF-driving input. The BOLD monitors **(D–F)** using the two-input Balloon model defined in this work with separately driven CBF and CMRO2. The responses of BOLD monitor E are shown in the **Supplementary Figure 1** as they are almost identical to those of BOLD monitor **(D)**.

increase similarly in response to the stimulus pulse, the BOLD response also looks similar and only differs in amplitude. When driving CBF and CMRO2 in parallel with different source variables, the choice of the source variable is much more important, because the relationship between them critically affects the shape of the BOLD response not only the amplitude. Nevertheless, the effect of changing the source variable on the resulting BOLD signal may be different for other underlying network models with different dynamics of the different variables (e.g., synaptic currents, mean firing rate, etc.), even when the classic Balloon model is used.

In a second experiment, we perform a simulation with sustained stimulation (longer stimulus pulse) with our six different BOLD monitors. The firing rate of the Poisson neurons is increased by a factor of 1.2 for 20 s. Like in the stimulus pulse simulations, the responses of the first three BOLD monitors (A-C) are very similar and only differ in amplitude (**Figure 7**, left). The CBF or BOLD responses show a slight initial overshoot, then reach a plateau, and finally, show a slight post-stimulus undershoot. The three BOLD monitor variants with two input signals (D-F) again show significant differences from the three BOLD monitors using the classic Balloon model. The BOLD monitors D and E showed almost identical responses consisting of an initial undershoot a negative plateau and a post-stimulus over- and undershoot (for results of BOLD monitor E see **Supplementary Figure 1**). It is particularly noticeable that the plateau is negative for the BOLD monitors D & E but not for BOLD monitor F because only for BOLD monitor F, the CBF increases more than the CMRO2. This again illustrates how critical the choice of source variables is when CBF and CMRO2 are driven in parallel by them.

4.4. Computational Time Analysis

In this section, we study the additional computational time introduced by the BOLD monitor (hereafter referred to as computational overhead). We use a scaled version of the microcircuit model described in Section 4.1, by incrementally increasing the number of neurons for the populations and leaving the number of synaptic inputs for a neuron fixed to 10 connections (from 10 different neurons of the pre-synaptic population) per projection. **Table 2** shows an overview of the total number of neurons and connections for each network model instance.

Figure 8 depicts the single thread computational time in seconds as a function of the number of recorded neurons with (blue line) and without (orange line) BOLD recording. For each configuration, we performed 10 runs, each simulating 25 s biological time and we measured the elapsed real time with the Python *time* module. The relative standard deviation was in the range of 0.55% to 2.53% which is too small to be depicted meaningfully in the graph and was therefore omitted.

For all simulated configurations, the computational time with and without BOLD recording is globally similar (between 1% and 8% of overhead depending on the model's size). The relative computational overhead (visualized as gray bars) is larger for small network models but shrinks when the model size increases. Therefore, if network models get more complex, in the sense of number of neurons, complexity of neuron models and the number of connections, one can expect that the share of the computational overhead will shrink accordingly. Overall, the computational time is dominated by the complexity of the network model and the BOLD recording plays a minor role, especially in complex network models.

TABLE 2 | Overview on the network model sizes used for the computational time analysis.

Number of recorded neurons	Number of neurons	Number of connections
250	450	5,500
500	900	11,000
1,000	1,800	22,000
2,000	3,600	44,000
4,000	7,200	88,000
8,000	14,400	176,000
16,000	28,800	352,000
32,000	57,600	704,000

The first column are the number of recorded neurons (i.e., corE and corI populations), the second column the number of all neurons (additionally the Poisson population) within the network model and the third column the overall number of connections.

5. DISCUSSION

In this work, we presented a BOLD monitor for obtaining simulated BOLD signals from spiking or rate-coded network models in the ANNarchy neural simulator. All variants of the Balloon model summarized by Stephan et al. (2007), thus the currently prevailing BOLD models, are available as built-in models. The integrated BOLD monitor makes it easy for users to connect their network models to a mathematical BOLD model such as the Balloon model (Buxton et al., 1998) or their own user-defined BOLD models. Users only need to specify from which populations they want to record the BOLD signal, which BOLD model they want to use and which variables of the neurons should be mapped to the input signal(s) of the BOLD model.

The optional baseline normalization of the source variables is a useful feature, as it allows the use of variables with arbitrary magnitudes for the BOLD calculation (including, for example, negative membrane potentials or large synaptic currents), since it sets the relative change of the source variables as the input signal for the BOLD calculation. This is a simple and effective alternative to the previous normalization approaches of the input signals (Schmidt et al., 2018; Maith et al., 2021). Another advantage of baseline normalization is that the resulting input signal for the BOLD model is approximately zero at rest and thus suitable for the Balloon model. A limitation is that it can only use variables that have a relatively constant non-zero mean in the resting-state of the network model. It is highly recommended that users verify that the normalization is appropriate for their chosen variables and used BOLD model. For example, an unsuitable source variable would be the mean firing rate of neurons that are quiescent during the baseline calculation phase and are activated due to a model manipulation after the baseline calculation phase (e.g., during an experiment with input presentation). Another example would be if the selected source variable must first enter a steady-state at the beginning of the simulation (e.g., increase from 0 to a constant non-zero value) and one conducts the baseline calculation during this ramp-up period. This would lead to a too low baseline and thus to a permanently positive normalized signal during recording.

The implementation of the BOLD monitor is flexible enough so that the source variables for the BOLD calculation can be any of the variables present in the neuron models (e.g., a combination of different synaptic currents). Recently, an energy-dependent leaky integrate-and-fire neuron model has been developed that accounts for the neuron's energy consumption by calculating adenosine triphosphate (ATP) dynamics (Jaras et al., 2021). The variables involved there, which are associated with the brain's metabolism, could be of great interest for calculating the BOLD signal and could be easily linked to BOLD models in ANNarchy using the BOLD monitor. Such flexibility makes ANNarchy with the BOLD monitor an useful environment for investigating hypotheses about the coupling between neural processes and BOLD signals, which is an active area of research (Buxton, 2021; Howarth et al., 2021). Since the coupling between neural processes and BOLD signals is still quite unclear, there is no recommended standard method for obtaining simulated BOLD signals with network models (Einevoll et al., 2019). We have demonstrated here how to use the BOLD monitor to study the role of different source variables in a simple network model of a cortical microcircuit. As such, ANNarchy and the new BOLD monitor can support research in neurovascular coupling, which may lead to the development of better BOLD models in the future and possibly to a better understanding of the BOLD response.

The ability to easily obtain BOLD signals from network models opens up more potential applications for ANNarchy, particularly in the area of model-based analysis of neuroimaging data (see Popovych et al., 2019 for a review). The basic idea here is to adjust network models to replicate experimental MRI data while simulating underlying neural processes that cannot be inferred from the MRI data alone. Especially for the study of neuronal diseases in humans, model-based analysis offers new opportunities. Network models customized to patients can be compared with network models customized to healthy controls, or the customized network models can be used as a virtual test bed for specific treatments (Cabral et al., 2013; Van Hartevelt et al., 2014; Jirsa et al., 2017; Meier et al., 2021). Since this approach has been mainly performed with macroscopic network models, ANNarchy can extend this approach by being used mainly in the study of processes at the mesoscopic level of detail. A possible application would be the study of deep brain stimulation (DBS) in, e.g., Parkinson's disease patients, the mechanisms of which may be more extensively and realistically implemented in ANNarchy (similar to other mesoscopic network models, e.g., Rubin and Terman, 2004; Hahn and McIntyre, 2010) than in macroscopic network models (e.g., Meier et al., 2021). Similar to the recently proposed approach to predict DBS-induced clinical improvements using MRI data from Parkinson's disease patients (Horn et al., 2017, 2019), predictors for clinical improvements could also be obtained from model-based analysis of the MRI data. Speculatively, these predictive approaches could potentially even be used in combination with intraoperative MRI (Cui et al., 2016) in the future to optimize electrode positions during DBS electrode implementation. In addition, model-based analysis of MRI data could potentially provide new biomarkers

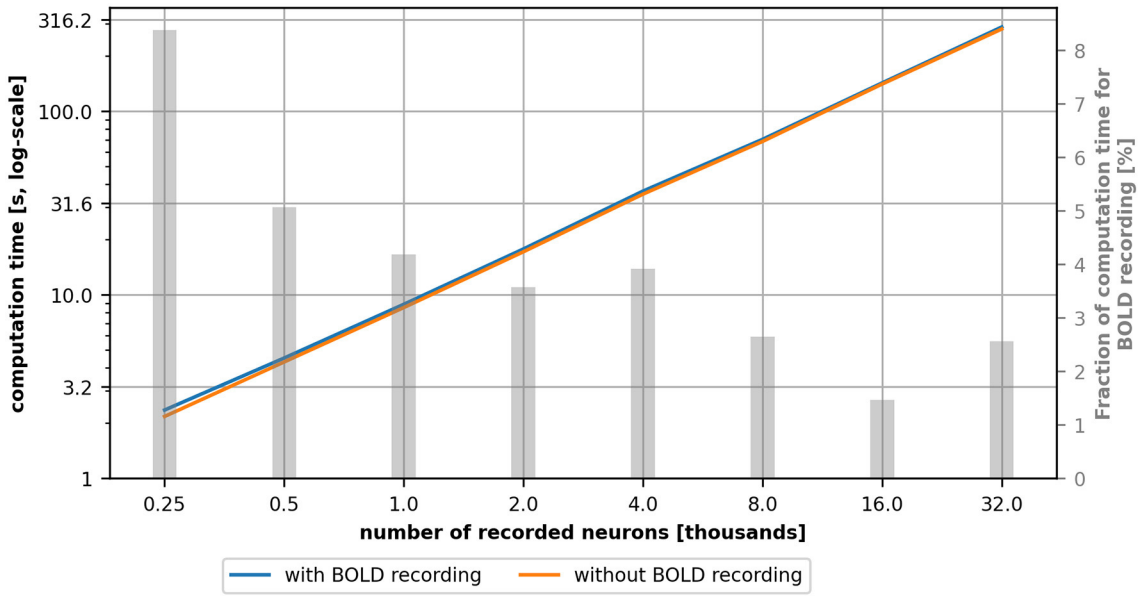


FIGURE 8 | Computational time for a simulation with (blue) and without (orange) BOLD recording as a function of the number of recorded neurons. The gray bars indicate the percentage of the computational overhead of the BOLD recording in the total computation time. This value was computed on the average values and therefore no standard deviation is depicted. The computational overhead is higher for small network models and gets smaller for larger models.

for mental disorders for which MRI data alone are not well-suited (Linden, 2012).

The BOLD monitor is already quite flexible and user friendly, but a potential improvement may be an optional delay for the input signals of the BOLD model. This was demonstrated, for example, for the Balloon model in Buxton et al. (2004). A delayed CBF relative to the CMRO2 could be the cause for the initial dip in the BOLD signal (Buxton et al., 2004; Buxton, 2012). In our two-input Balloon model, we currently implement this with a faster responding for CMRO2 than for CBF. However, whether the initial dip in the BOLD response is actually caused by a faster CMRO2 response is still a matter of debate in the literature (Buxton, 2012). Another useful extension would be individual scaling factors for each source variable signal in the preprocessing of the BOLD monitor. This would allow, for example, one population to be heavily weighted for CBF and another population for CMRO2. Currently, the scaling factor is based on the size of the population and can optionally be adjusted. One of the most important possible further developments concerns the simulation of realistic noise components of the BOLD signal. Experimentally collected BOLD signals are subject to physiological noise, especially motion, cardiac, and respiratory artifacts, as well as instrumental noise (Birn et al., 2008; Chang et al., 2009; Caballero-Gaudes and Reynolds, 2017). To meaningfully compare simulated and experimental signals, these noise sources should also be considered.

We have demonstrated the properties of the BOLD monitor using a simple network model of a cortical microcircuit. However, we did not focus on a use case that includes a comparison of a realistic network model with experimentally obtained BOLD data. Our microcircuit model is not such a use case, but mainly functions as a means to demonstrate the possibilities of the BOLD monitor. Thus, the simulated BOLD responses should not be overinterpreted. Our implementation could be helpful for researchers to compare different BOLD models. Our simulations showed that different source variables of the same underlying network model can affect the simulated BOLD signal differently and, most importantly, that this can be easily tested with the BOLD monitor in ANNarchy. To actually link experimental BOLD signals to their underlying neural processes, more realistic and detailed network models should be used (Vanni et al., 2015).

In this work, we implemented a modified version of the Balloon model in which CBF and CMRO2 are driven in parallel by two different input signals. This two-input Balloon model was composed of model components from previous publications (Buxton et al., 1998, 2004; Friston et al., 2000). By implementing this BOLD model, we demonstrated how ANNarchy allows users to define their own systems of equations as a BOLD model. A BOLD model considering parallel excitation of CBF and CMRO2 will be necessary for future model-based investigation of current hypotheses regarding the origin of the BOLD signal (Buxton, 2021).

Other modeling tools also provide the ability to simulate BOLD signals or analyze MRI data in a model-based manner. One of the best known is Dynamic Causal Modeling (DCM) by Friston et al. (2003), which is included in the Matlab Software Package SPM (Penny et al., 2011). DCM can be used to obtain the effective connectivity of network models from MRI data. The model implementation in DCM differs significantly from that in ANNarchy, where more complex network models can be

implemented at finer scales, for example with spiking neurons, detailed neuron and synapse definitions. In DCM, the focus is not on explicitly implementing neural processing, but on investigating how brain regions interact: the dynamics of the brain regions are usually simulated by an activity vector which depends on a connectivity matrix and driving and modulating inputs defined by an experimental paradigm. The length of the activity vector usually corresponds to the number of regions included, i.e., each region is described by one activity value. Simulated BOLD signals for the different brain regions are obtained from the activities of the regions using the Balloon model versions of Stephan et al. (2007). Based on this, free parameters (e.g., the connectivity matrix) are optimized using Bayesian inference to replicate the MRI data and keep the model complexity low (also called Bayesian model inversion). In DCM, other BOLD models than the Balloon model are not available. DCM is not designed to flexibly test hypotheses regarding neurovascular coupling. Therefore, DCM in SPM and ANNarchy with the new BOLD monitor are designed for different applications.

Another modeling tool that incorporates simulation of BOLD signals is The Virtual Brain (TVB) (Ritter et al., 2013; Sanz Leon et al., 2013). TVB is a neural simulator to create large-scale network models usually of the whole cortex and not a mathematical setup for model inversion using BOLD data as DCM, which is only one possible application of TVB. In TVB, network models are usually implemented as a combination of neural mass models, sets of equations that describe the average dynamics of large neuron populations (macroscopic models), but neglect processes at the single-neuron level. Therefore, a TVB – multi-scale co-simulation toolbox that links TVB and neural simulators which model the lower scale processes such as ANNarchy and NEST (Gewaltig and Diesmann, 2007), has been recently introduced (Meier et al., 2021; Schirner et al., 2022). BOLD simulation in TVB is mainly used to validate large-scale network models on experimental MRI data. In TVB, the different versions of the Balloon model of Stephan et al. (2007) are available. However, a flexible definition of source variables or the BOLD model is not currently available because the focus is not on examining the relationship between the BOLD signal and detailed neural processes.

Several successful neural simulators, such as NEST (Gewaltig and Diesmann, 2007) and Brian2 (Stimberg et al., 2019), do not yet have an integrated BOLD simulation routine. For these simulators, users currently have to use external tools for BOLD simulation like the R package neuRosim (Welvaert et al., 2011). Several hemodynamic response functions (HRF) are available in neuRosim, including the Balloon model from Buxton et al. (2004), which can be used to calculate a BOLD response from a given stimulus signal. The stimulus signal typically follows an experimental design, with 1 indicating the presence and 0 the absence of a stimulus. Simulating the BOLD signal based on specific neural processes is actually not the intended use of neuRosim. Nevertheless, neuRosim can be applied to specific simulated signals from network models (Schmidt et al., 2018). A separate definition of the BOLD model (or the HRF in neuRosim) is not currently available. The strengths of neuRosim are the possibility to define spatial positions and the extent of BOLD activation and the modeling of different noise sources of the BOLD signal.

An important advantage of on-line BOLD computation in ANNarchy over off-line computation such as using neuRosim is that simulated data of the recorded neurons (e.g., membrane potentials or synaptic currents) do not need to be stored separately to be used for BOLD computation after simulation. The latter can result in significant increased memory requirements, especially for larger network models. On the other hand, the on-line BOLD computation increases the computation time of the simulations. However, this is a less crucial factor than, for example, the size of the network model, as we show in Section 4.4. Moreover, the share of the on-line BOLD computation in the computation time decreases as the complexity of the model increases. Therefore, the use of the BOLD monitor is also appropriate for larger network models than those used in this work.

In summary, we introduced the BOLD monitor in ANNarchy which allows the on-line computation of simulated BOLD signals directly from spiking or rate-coded network models. Highlights of the BOLD monitor are the flexible definition of source variables in the neuron models of the recorded network model and the possibility to use new user-defined BOLD models. We demonstrated here how this can be done and how this can be used, for example, to compare different hypotheses regarding neurovascular coupling. This tool allows both the validation and optimization of network models with experimental MRI data and the model-based analysis of the BOLD response for a better understanding of its neural basis.

DATA AVAILABILITY STATEMENT

All used source code is publicly available. This data can be found here: the ANNarchy neural simulator (4.7.0 release) is available on github: https://github.com/ANNarchy/ANNarchy. The simulation code of this work is available on github: https://doi.org/10.5281/zenodo.5547665.

AUTHOR CONTRIBUTIONS

OM and HD: designed the research, performed the research, programming, data analysis, and writing (first draft). JB, JV, and FH: guided the research. FH: acquired the funding. OM, HD, JB, JV, and FH: writing (reviewing) and editing. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) SPP-2041 Computational Connectomics as part of the project Clinical Connectomics: A network approach to deep brain stimulation (DFG HA2630/11-2 and HA2630/11-1) and in part by Auto-tuning

for neural simulations on different parallel hardware (DFG HA2630/9-1). The publication of this article was funded by Chemnitz University of Technology and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 491193532.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.790966/full#supplementary-material

REFERENCES

Baladron, J., Nambu, A., and Hamker, F. H. (2019). The subthalamic nucleus-external globus pallidus loop biases exploratory decisions towards known alternatives: a neuro-computational study. *Eur. J. Neurosci.* 49, 754–767. doi: 10.1111/ejn.13666

Beaulieu, C., Kisvarday, Z., Somogyi, P., Cynader, M., and Cowey, A. (1992). Quantitative distribution of gaba-immunopositive and-immunonegative neurons and synapses in the monkey striate cortex (area 17). *Cereb. Cortex* 2, 295–309. doi: 10.1093/cercor/2.4.295

Bennett, M. R., Farnell, L., Gibson, W. G., and Lagopoulos, J. (2015). Cortical network models of firing rates in the resting and active states predict bold responses. *PLoS ONE* 10, e0144796. doi: 10.1371/journal.pone.0144796

Birn, R. M., Smith, M. A., Jones, T. B., and Bandettini, P. A. (2008). The respiration response function: the temporal dynamics of fMRI signal fluctuations related to changes in respiration. *Neuroimage* 40, 644–654. doi: 10.1016/j.neuroimage.2007.11.059

Buxton, R. B. (2012). Dynamic models of bold contrast. *Neuroimage* 62, 953–961. doi: 10.1016/j.neuroimage.2012.01.012

Buxton, R. B. (2021). The thermodynamics of thinking: connections between neural activity, energy metabolism and blood flow. *Philos. Trans. R. Soc. B* 376, 20190624. doi: 10.1098/rstb.2019.0624

Buxton, R. B., and Frank, L. R. (1997). A model for the coupling between cerebral blood flow and oxygen metabolism during neural stimulation. *J. Cereb. Blood Flow Metab.* 17, 64–72. doi: 10.1097/00004647-199701000-00009

Buxton, R. B., Griffeth, V. E., Simon, A. B., and Moradi, F. (2014). Variability of the coupling of blood flow and oxygen metabolism responses in the brain: a problem for interpreting bold studies but potentially a new window on the underlying neural activity. *Front. Neurosci.* 8, 139. doi: 10.3389/fnins.2014.00139

Buxton, R. B., Uludağ, K., Dubowitz, D. J., and Liu, T. T. (2004). Modeling the hemodynamic response to brain activation. *Neuroimage* 23, S220–S233. doi: 10.1016/j.neuroimage.2004.07.013

Buxton, R. B., Wong, E. C., and Frank, L. R. (1998). Dynamics of blood flow and oxygenation changes during brain activation: the balloon model. *Magnet. Reson. Med.* 39, 855–864. doi: 10.1002/mrm.1910390602

Buzsáki, G., and Mizuseki, K. (2014). The log-dynamic brain: how skewed distributions affect network operations. *Nat. Rev. Neurosci.* 15, 264–278. doi: 10.1038/nrn3687

Caballero-Gaudes, C., and Reynolds, R. C. (2017). Methods for cleaning the bold fMRI signal. *Neuroimage* 154, 128–149. doi: 10.1016/j.neuroimage.2016.12.018

Cabral, J., Fernandes, H. M., Van Hartevelt, T. J., James, A. C., Kringelbach, M. L., and Deco, G. (2013). Structural connectivity in schizophrenia and its impact on the dynamics of spontaneous functional networks. *Chaos* 23, 046111. doi: 10.1063/1.4851117

Chang, C., Cunningham, J. P., and Glover, G. H. (2009). Influence of heart rate on the bold signal: the cardiac response function. *Neuroimage* 44, 857–869. doi: 10.1016/j.neuroimage.2008.09.029

Corbit, V. L., Whalen, T. C., Zitelli, K. T., Crilly, S. Y., Rubin, J. E., and Gittis, A. H. (2016). Pallidostriatal projections promote β oscillations in a dopamine-depleted biophysical network model. *J. Neurosci.* 36, 5556–5571. doi: 10.1523/JNEUROSCI.0339-16.2016

Cui, Z., Pan, L., Song, H., Xu, X., Xu, B., Yu, X., et al. (2016). Intraoperative MRI for optimizing electrode placement for deep brain stimulation of the subthalamic nucleus in Parkinson disease. *J. Neurosurg.* 124, 62–69. doi: 10.3171/2015.1.JNS141534

Davis, T. L., Kwong, K. K., Weisskoff, R. M., and Rosen, B. R. (1998). Calibrated functional MRI: mapping the dynamics of oxidative metabolism. *Proc. Natl. Acad. Sci. U.S.A.* 95, 1834–1839. doi: 10.1073/pnas.95.4.1834

Deco, G., and Jirsa, V. K. (2012). Ongoing cortical activity at rest: criticality, multistability, and ghost attractors. *J. Neurosci.* 32, 3366–3375. doi: 10.1523/JNEUROSCI.2523-11.2012

Dinkelbach, H. Ü., Vitay, J., and Hamker, F. H. (2019). "Scalable simulation of rate-coded and spiking neural networks on shared memory systems," in *2019 Conference on Cognitive Computational Neuroscience* (Berlin: Cognitive Computational Neuroscience), 526–529. doi: 10.32470/CCN.2019.1109-0

Einevoll, G. T., Destexhe, A., Diesmann, M., Grün, S., Jirsa, V., de Kamps, M., et al. (2019). The scientific case for brain simulations. *Neuron* 102, 735–744. doi: 10.1016/j.neuron.2019.03.027

Friston, K. J., Harrison, L., and Penny, W. (2003). Dynamic causal modelling. *Neuroimage* 19, 1273–1302. doi: 10.1016/S1053-8119(03)00202-7

Friston, K. J., Mechelli, A., Turner, R., and Price, C. J. (2000). Nonlinear responses in fMRI: the Balloon model, volterra kernels, and other hemodynamics. *Neuroimage* 12, 466–477. doi: 10.1006/nimg.2000.0630

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2, 1430. doi: 10.4249/scholarpedia.1430

Glover, G. H. (1999). Deconvolution of impulse response in event-related bold fMRI1. *Neuroimage* 9, 416–429. doi: 10.1006/nimg.1998.0419

Goenner, L., Maith, O., Koulouri, I., Baladron, J., and Hamker, F. H. (2021). A spiking model of basal ganglia dynamics in stopping behavior supported by arkypallidal neurons. *Eur. J. Neurosci.* 53, 2296–2321. doi: 10.1111/ejn.15082

Günay, C., Edgerton, J. R., and Jaeger, D. (2008). Channel density distributions explain spiking variability in the globus pallidus: a combined physiology and computer simulation database approach. *J. Neurosci.* 28, 7476–7491. doi: 10.1523/JNEUROSCI.4198-07.2008

Hahn, P. J., and McIntyre, C. C. (2010). Modeling shifts in the rate and pattern of subthalamopallidal network activity during deep brain stimulation. *J. Comput. Neurosci.* 28, 425–441. doi: 10.1007/s10827-010-0225-8

Heikkinen, H., Sharifian, F., Vigario, R., and Vanni, S. (2015). Feedback to distal dendrites links fMRI signals to neural receptive fields in a spiking network model of the visual cortex. *J. Neurophysiol.* 114, 57–69. doi: 10.1152/jn.00169.2015

Horn, A., Reich, M., Vorwerk, J., Li, N., Wenzel, G., Fang, Q., et al. (2017). Connectivity predicts deep brain stimulation outcome in Parkinson disease. *Ann. Neurol.* 82, 67–78. doi: 10.1002/ana.24974

Horn, A., Wenzel, G., Irmen, F., Huebl, J., Li, N., Neumann, W.-J., et al. (2019). Deep brain stimulation induced normalization of the human functional connectome in Parkinson's disease. *Brain* 142, 3129–3143. doi: 10.1093/brain/awz239

Howarth, C., Mishra, A., and Hall, C. N. (2021). More than just summed neuronal activity: how multiple cell types shape the bold response. *Philos. Trans. R. Soc. B* 376, 20190630. doi: 10.1098/rstb.2019.0630

Humphries, M. D., Stewart, R. D., and Gurney, K. N. (2006). A physiologically plausible model of action selection and oscillatory activity in the basal ganglia. *J. Neurosci.* 26, 12921–12942. doi: 10.1523/JNEUROSCI.3486-06.2006

Humphries, M. D., Wood, R., and Gurney, K. (2009). Dopamine-modulated dynamic cell assemblies generated by the gabaergic striatal microcircuit. *Neural Netw.* 22, 1174–1188. doi: 10.1016/j.neunet.2009.07.018

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440

Izhikevich, E. M. (2007). *Dynamical Systems in Neuroscience*. Cambridge: MIT Press.

Jaras, I., Harada, T., Orchard, M. E., Maldonado, P. E., and Vergara, R. C. (2021). Extending the integrate-and-fire model to account for metabolic dependencies. *Eur. J. Neurosci.* 54, 5249–5260. doi: 10.1111/ejn.15326

Jirsa, V. K., Proix, T., Perdikis, D., Woodman, M. M., Wang, H., Gonzalez-Martinez, J., et al. (2017). The virtual epileptic patient: individualized

whole-brain models of epilepsy spread. *Neuroimage* 145, 377–388. doi: 10.1016/j.neuroimage.2016.04.049

Linden, D. E. (2012). The challenges and promise of neuroimaging in psychiatry. *Neuron* 73, 8–22. doi: 10.1016/j.neuron.2011.12.014

Maith, O., Villagrasa Escudero, F., Dinkelbach, H. Ü., Baladron, J., Horn, A., Irmen, F., et al. (2021). A computational model-based analysis of basal ganglia pathway changes in Parkinson's disease inferred from resting-state fMRI. *Eur. J. Neurosci.* 53, 2278–2295. doi: 10.1111/ejn.14868

Meier, J., Perdikis, D., Blickensdörfer, A., Stefanovski, L., Liu, Q., Maith, O., et al. (2021). Virtual deep brain stimulation: multiscale co-simulation of a spiking basal ganglia model and a whole-brain mean-field model with the virtual brain. *bioRxiv*, 1–38. doi: 10.1101/2021.05.05.442704

Penny, W. D., Friston, K. J., Ashburner, J. T., Kiebel, S. J., and Nichols, T. E. (2011). *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. London: Elsevier.

Popovych, O. V., Manos, T., Hoffstaedter, F., and Eickhoff, S. B. (2019). What can computational models contribute to neuroimaging data analytics? *Front. Syst. Neurosci.* 12, 68. doi: 10.3389/fnsys.2018.00068

Pospischil, M., Toledo-Rodriguez, M., Monier, C., Piwkowska, Z., Bal, T., Frégnac, Y., et al. (2008). Minimal Hodgkin–Huxley type models for different classes of cortical and thalamic neurons. *Biol. Cybernet.* 99, 427–441. doi: 10.1007/s00422-008-0263-8

Potjans, T. C., and Diesmann, M. (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cereb. Cortex* 24, 785–806. doi: 10.1093/cercor/bhs358

Ritter, P., Schirner, M., McIntosh, A. R., and Jirsa, V. K. (2013). The virtual brain integrates computational modeling and multimodal neuroimaging. *Brain Connect.* 3, 121–145. doi: 10.1089/brain.2012.0120

Rubin, J. E., and Terman, D. (2004). High frequency stimulation of the subthalamic nucleus eliminates pathological thalamic rhythmicity in a computational model. *J. Comput. Neurosci.* 16, 211–235. doi: 10.1023/B:JCNS.0000025686.47117.67

Sanz Leon, P., Knock, S. A., Woodman, M. M., Domide, L., Mersmann, J., McIntosh, A. R., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinform.* 7, 10. doi: 10.3389/fninf.2013.00010

Schirner, M., Domide, L., Perdikis, D., Triebkorn, P., Stefanovski, L., Pai, R., et al. (2022). Brain modelling as a service: the virtual brain on ebrains. *NeuroImage*, 251:118973. doi: 10.1016/j.neuroimage.2022.118973

Schmidt, M., Bakker, R., Shen, K., Bezgin, G., Diesmann, M., and van Albada, S. J. (2018). A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. *PLoS Comput. Biol.* 14, e1006359. doi: 10.1371/journal.pcbi.1006359

Serences, J. T. (2004). A comparison of methods for characterizing the event-related bold timeseries in rapid fMRI. *Neuroimage* 21, 1690–1700. doi: 10.1016/j.neuroimage.2003.12.021

Smith, S. M., Miller, K. L., Salimi-Khorshidi, G., Webster, M., Beckmann, C. F., Nichols, T. E., et al. (2011). Network modelling methods for fMRI. *Neuroimage* 54, 875–891. doi: 10.1016/j.neuroimage.2010.08.063

Song, S., Sjöström, P. J., Reigl, M., Nelson, S., and Chklovskii, D. B. (2005). Highly nonrandom features of synaptic connectivity in local cortical circuits. *PLoS Biol.* 3, e68. doi: 10.1371/journal.pbio.0030068

Stephan, K. E., Weiskopf, N., Drysdale, P. M., Robinson, P. A., and Friston, K. J. (2007). Comparing hemodynamic models with DCM. *Neuroimage* 38, 387–401. doi: 10.1016/j.neuroimage.2007.07.040

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8, e47314. doi: 10.7554/eLife.47314.028

Van Hartevelt, T. J., Cabral, J., Deco, G., Møller, A., Green, A. L., Aziz, T. Z., et al. (2014). Neural plasticity in human brain connectivity: the effects of long term deep brain stimulation of the subthalamic nucleus in Parkinson's disease. *PLoS ONE* 9, e86496. doi: 10.1371/journal.pone.0086496

Vanni, S., Sharifian, F., Heikkinen, H., and Vigário, R. (2015). Modeling fMRI signals can provide insights into neural processing in the cerebral cortex. *J. Neurophysiol.* 114, 768–780. doi: 10.1152/jn.00332.2014

Vitay, J., Dinkelbach, H. Ü., and Hamker, F. H. (2015). ANNarchy: a code generation approach to neural simulations on parallel hardware. *Front. Neuroinform.* 9, 19. doi: 10.3389/fninf.2015.00019

Welvaert, M., Durnez, J., Moerkerke, B., Berdoolaege, G., and Rosseel, Y. (2011). neurosim: an r package for generating fMRI data. *J. Stat. Softw.* 44, 1–18. doi: 10.18637/jss.v044.i10

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 25 April 2022
doi: 10.3389/fncom.2022.847336

Extracting Dynamical Understanding From Neural-Mass Models of Mouse Cortex

Pok Him Siu[1], *Eli Müller*[1], *Valerio Zerbi*[2,3], *Kevin Aquino*[1] *and Ben D. Fulcher*[1]*

[1] School of Physics, The University of Sydney, Camperdown, NSW, Australia, [2] Neural Control of Movement Lab, D-HEST, ETH Zurich, Zurich, Switzerland, [3] Neuroscience Center Zurich, University and ETH Zurich, Zurich, Switzerland

New brain atlases with high spatial resolution and whole-brain coverage have rapidly advanced our knowledge of the brain's neural architecture, including the systematic variation of excitatory and inhibitory cell densities across the mammalian cortex. But understanding how the brain's microscale physiology shapes brain dynamics at the macroscale has remained a challenge. While physiologically based mathematical models of brain dynamics are well placed to bridge this explanatory gap, their complexity can form a barrier to providing clear mechanistic interpretation of the dynamics they generate. In this work, we develop a neural-mass model of the mouse cortex and show how bifurcation diagrams, which capture local dynamical responses to inputs and their variation across brain regions, can be used to understand the resulting whole-brain dynamics. We show that strong fits to resting-state functional magnetic resonance imaging (fMRI) data can be found in surprisingly simple dynamical regimes—including where all brain regions are confined to a stable fixed point—in which regions are able to respond strongly to variations in their inputs, consistent with direct structural connections providing a strong constraint on functional connectivity in the anesthetized mouse. We also use bifurcation diagrams to show how perturbations to local excitatory and inhibitory coupling strengths across the cortex, constrained by cell-density data, provide spatially dependent constraints on resulting cortical activity, and support a greater diversity of coincident dynamical regimes. Our work illustrates methods for visualizing and interpreting model performance in terms of underlying dynamical mechanisms, an approach that is crucial for building explanatory and physiologically grounded models of the dynamical principles that underpin large-scale brain activity.

Keywords: brain dynamics, dynamical systems, neural mass model, mouse cortex, cell densities

OPEN ACCESS

Edited by:
Kelly Shen,
Simon Fraser University, Canada

Reviewed by:
Spase Petkoski,
INSERM U1106 Institut de Neurosciences des Systèmes, France
Jorge F. Mejias,
University of Amsterdam, Netherlands

***Correspondence:**
Ben D. Fulcher
ben.fulcher@sydney.edu.au

Received: *02 January 2022*
Accepted: *22 March 2022*
Published: *25 April 2022*

Citation:
Siu PH, Müller E, Zerbi V, Aquino K and Fulcher BD (2022) Extracting Dynamical Understanding From Neural-Mass Models of Mouse Cortex. Front. Comput. Neurosci. 16:847336. doi: 10.3389/fncom.2022.847336

1. INTRODUCTION

Recent advances in neuroimaging have produced intricate maps revealing the complexity of the brain's microscale circuits, with whole-brain coverage. Analyzing and integrating these data have uncovered new patterns of brain organization, including the systematic spatial variation of gene expression (Burt et al., 2018; Fulcher et al., 2019), cytoarchitecture (Goulas et al., 2016), neuron densities (Erö et al., 2018), cortical thickness (Wagstyl et al., 2015), axonal connectivity (Oh et al., 2014), cognitive function (Margulies et al., 2016), and local dynamical properties (Shafiei et al., 2020). Existing evidence suggests that, to a good first approximation, these properties vary together

along a dominant hierarchical axis in mouse and human (Burt et al., 2018; Fulcher et al., 2019; Wang, 2020).

To understand the functional role of observed physiological patterns, like systematic spatial variations in brain architecture, we need a way of simulating their effect on whole-brain dynamics. Physiologically based brain models achieve this, using methods from statistical physics to capture the dynamics of large populations of neurons and their interactions (Deco et al., 2008; Breakspear, 2017). Neural population models can capture the complex spatiotemporal dynamics in modern neuroimaging datasets, including persistent activity, intermittent oscillations, and multi-stability (Robinson et al., 2016; Noori et al., 2020; Froudist-Walsh et al., 2021; Mejías and Wang, 2022), and have successfully reproduced a wide range of experimental phenomena, from the alpha rhythm to seizure dynamics (Mejias et al., 2016; Breakspear, 2017; Schneider et al., 2021; Sip et al., 2022). The physiological formulation of these models means that their variables and parameters encode interpretable and biologically measurable properties of neural circuits, like the strengths and timescales of interactions between neuronal populations. This allows them to provide a unique mechanistic account of whole-brain dynamics that can be validated against both physiological experiments and the dynamical patterns observed in neuroimaging experiments.

While most existing brain models involve dynamical rules that are spatially uniform (e.g., the same model parameters in all brain areas), recent work has begun to investigate the effect of non-uniform dynamical rules, constrained by emerging brain-atlas datasets. An early example is the work of Chaudhuri et al. (2015), which incorporated a variation in recurrent excitation corresponding to that of measured spine count in the macaque. More recent work in human has incorporated spatial heterogeneity in model parameters with: the MRI-derived T1w:T2w map (Demirtas et al., 2019); T1w:T2w, the first principal component of gene transcription, and an inferred excitation:inhibition ratio (Deco et al., 2021); a linear combination of T1w:T2w and the principal resting-state functional connectivity (FC) gradient (Kong et al., 2021); a fitted parametric variation that recapitulated an interpretable hierarchical variation (Wang et al., 2019); and a spatial variation in excitability with a spatial map of epileptogenicity in modeling seizure dynamics and spread (Jirsa et al., 2017; Courtiol et al., 2020). These papers have reported improved out-of-sample model fits to empirical data, evaluated according to a range of summary statistics of the resulting dynamics (most typically FC), and provided insights into how spatial variation in biological mechanisms (like recurrent excitation) may underpin whole-brain dynamical regimes. While these studies demonstrate the promise of producing more accurate predictions of measured brain dynamics by incorporating regional heterogeneity—constraining to physiological data, or through large-scale parameter fitting (Wang et al., 2019)—the resulting models are correspondingly complex and challenging to interpret in terms of the mechanisms which underpin their dynamics. The tools of dynamical systems have the potential to reveal the dynamical features that improve model fits to data, including the bifurcation structure that defines the accessible dynamical regimes and the range of such regimes that different brain areas can access, including their vicinity to critical points (Deco and Jirsa, 2012; Deco et al., 2013; Cocchi et al., 2017; Demirtas et al., 2019; Wang et al., 2019). In this work, we show that analyzing the dynamical response of individual brain regions to inputs using bifurcation diagrams provides an understanding of model behavior in terms of accessible dynamical regimes, an approach that is particularly valuable for understanding the increased complexity of spatially non-uniform models.

The mouse is an ideal organism to develop comprehensively constrained physiologically based models of brain dynamics, but models of the mouse brain have been relative few compared to the large number of studies of human cortex. Existing models of mouse-brain dynamics on the macroscale have taken a variety of approaches, from phenomenological—connectome-coupled Kuramoto oscillators (Choi and Mihalas, 2019; Allegra Mascaro et al., 2020) and network diffusion models (Shadi et al., 2020)—through to neural mass models (Lin et al., 2020) coupled *via* a connectome (Melozzi et al., 2017, 2019) and interacting populations of spiking neural networks (Nunes et al., 2021). Compared to human, there is an abundance of high-resolution, whole-brain physiological data in mouse (Fulcher et al., 2019), including directed tract-tracing axonal connectivity data (Oh et al., 2014; Harris et al., 2019), high-resolution gene-expression maps (Lein et al., 2007), and cell-density atlases (Kim et al., 2017; Erö et al., 2018). High-quality whole-brain neuroimaging data using fMRI in mouse is also available, allowing us to evaluate model predictions in the resting state (Zerbi et al., 2015; Grandjean et al., 2020) and as a result of targeted manipulations (Zerbi et al., 2019; Markicevic et al., 2020, 2022). Prior work has shown that FC is strongly constrained by direct structural pathways (Grandjean et al., 2017), and prior dynamical models have reported the ability of coupled dynamical models to reproduce FC structure, especially when modeling using matching individual structural connectivity (Melozzi et al., 2019). In this work, we develop a neural-mass model of mouse cortical dynamics, and aim to understand the dynamical regimes in which it best captures resting-state fMRI data in mouse. We also aim to characterize the impact of incorporating spatial variations in excitatory and inhibitory cell densities as spatial variations in model parameters from a dynamical systems perspective.

2. METHODS

As illustrated in **Figure 1**, we developed a neural mass model of the right hemisphere of the mouse cortex, across 37 cortical areas, comprising a simple Wilson–Cowan local dynamical model (**Figure 1A**) coupled *via* a directed structural connectome (**Figure 1B**). These regions are shown on the mouse brain in **Figure 1C**, colored by their relative excitatory cell densities (which are incorporated into the model in section 3.2). Of the 38 cortical regions reported in Oh et al. (2014), we excluded the frontal pole (FRP) due to its small size (likely contributing to noisy, outlying values of excitatory and inhibitory cell densities Erö et al., 2018). In visualizing our results, we grouped cortical regions according to six functional labels: Somatomotor, Medial,

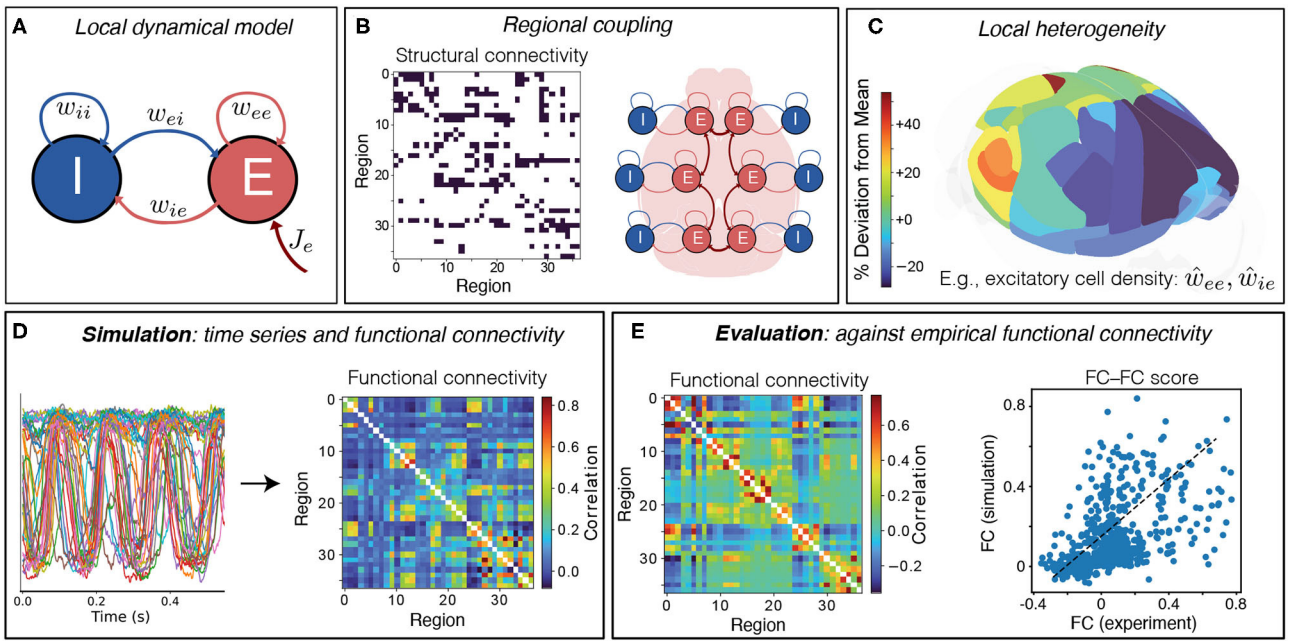


FIGURE 1 | Simulating and evaluating a coupled neural-mass model of mouse cortical dynamics. **(A)** The dynamics of individual brain regions follow the Wilson–Cowan equations (Wilson and Cowan, 1972, 1973) which govern interactions between local excitatory (*E*) and inhibitory (*I*) neural populations. **(B)** Regions are coupled together by connections defined by the AMBCA (Oh et al., 2014), represented as a directed adjacency matrix (connections shown black). A schematic shows how these long-range structural connections couple local cortical regions *via* excitatory projections (Breakspear, 2017). **(C)** Heterogeneity in local model parameters can be introduced as a perturbation that follows the measured variation in excitatory and inhibitory neural densities. Here the variation in excitatory cell density is plotted across the 37 mouse cortical areas as deviations relative to the mean level (green), using `brainrender` (Claudi et al., 2021) and data from Erö et al. (2018). **(D)** Model simulation yields activity time series for each brain region, from which pairwise linear correlations (functional connectivity, FC) are computed. **(E)** Model simulations are evaluated against empirical FC, averaged across 100 mice, as the Spearman correlation between all unique pairwise FC values, yielding an FC–FC score, ρ_{FCFC}.

Temporal, Visual, Anterolateral, and Prefrontal (Harris et al., 2019) (see **Supplementary Table S1** for full list).

As shown in **Figure 1A**, a given brain region consists of both an excitatory (*E*) and an inhibitory (*I*) neural population, whose dynamics are governed by the Wilson–Cowan equations (Wilson and Cowan, 1972, 1973). Brain regions are coupled *via* long-range excitatory projections using a binary, directed connectome from the Allen Mouse Brain Connectivity Atlas (AMBCA) (Oh et al., 2014; Fulcher and Fornito, 2016) (**Figure 1B**). As these data are the result of right-hemisphere viral tracer injections, yielding estimates of ipsilateral cortical connectivity in the right hemisphere, we modeled just the right hemisphere in this work, but note that model of both hemispheres could be developed in future under the assumption of lateral symmetry [e.g., as Melozzi et al. (2017)]. Simulating the model yields dynamics for the *E* and *I* populations; we take the activity time series of the excitatory population to evaluate the similarity of pairwise linear correlation structure as functional connectivity (FC), shown in **Figure 1D**. To assess the goodness of fit, we compare this simulated FC to an empirical FC calculated on a mouse fMRI dataset (**Figure 1E**). The goodness of fit is assessed as a Spearman correlation coefficient computed between all pairs of FC values from the empirical data and the model (**Figure 1E**). Spearman's correlation coefficient was used instead of Pearson's correlation coefficient to capture a potentially nonlinear but monotonic relationship.

As our main aim was to develop tools to understand the distributed dynamics of neural mass models, we favored simplicity in focusing on the Wilson–Cowan (W–C) model relative to alternative models. In addition, its physiological formulation is crucial for mapping to experimental cell-density data, as its parameters encode measurable properties with physical units that can be constrained by such data. The W–C model also exhibits a wide range of dynamical behaviors, including bifurcations, hysteresis, stable fixed-points (attractors), and limit cycles (oscillatory attractors) (Wilson and Cowan, 1972, 1973; Cowan et al., 2016), that are common features of dynamical systems in general, including more complex biophysical neural population models. We use a formulation of the Wilson–Cowan equations based on the mean firing rates of coupled populations of excitatory and inhibitory neurons, as

$$\tau_e \dot{E} = -E + (1-E)S\left[a_e\left(w_{ee}E - w_{ei}I - B_e + J_e\right)\right], \quad (1)$$

$$\tau_i \dot{I} = -I + (1-I)S\left[a_i\left(w_{ie}E - w_{ii}I - B_i\right)\right], \quad (2)$$

where E and I are the mean firing rates of the excitatory and inhibitory populations, respectively (Hz); $S(v) = h/[1+\exp(-v)]$ is the sigmoidal firing-rate function; h (which is set to 1 here) is

the upper bound for the sigmoid function representing a maximal population firing rate (Hz); a_e, a_i control the gradient scaling for the sigmoid function (V^{-1}); w_{xy} are the coupling weights from population y to population x, where x and y correspond to excitatory (e) or inhibitory (i) populations (V s); B_e, B_i are the firing thresholds for excitatory/inhibitory cells (V); J_e is the voltage induced by external current injected into the excitatory cells, defined below as a weighted sum over external inputs (V); and τ_e, τ_i are the time constants of excitation and inhibition respectively (s).

Neural masses, corresponding to cortical areas, were coupled *via* projections between excitatory populations through the external current term, J_e. For a given region a, $J_e^{(a)}(t)$ is computed as

$$J_e^{(a)}(t) = G \sum_b A_{ab} E^{(b)}(t), \tag{3}$$

where G is a global coupling constant (V s), A_{ab} is the adjacency matrix corresponding to the structural connectome (unweighted here), and $E^{(b)}$ is the excitatory activity of region b (Hz). It is helpful to define the quantity

$$J_{\text{tot}}^{(a)}(t) = J_e^{(a)}(t) - B_e = G \sum_b A_{ab} E^{(b)}(t) - B_e, \tag{4}$$

as the total input that includes the constant offset B_e. We will use this to understand the dynamical response of a brain area to its net input in section 3.1.

In addition to this "homogeneous" model, in which the parameters are identical for all brain regions, we also analyze a heterogeneous model (in section 3.2), in which the coupling parameters, w_{ij}, vary across regions. We calibrate this variation to estimated cell-density data (Erö et al., 2018), by making the assumption that local connectivity from excitatory and inhibitory cells is uniform, and thus that coupling strengths from a given population are proportional to the density of cells of that population. Thus, we adjust the coupling parameters corresponding to outputs from the excitatory population, w_{ee} and w_{ie}, according to measured variations in excitatory cell density across cortical areas, and adjust w_{ii} and w_{ei} according to measured variations in inhibitory cell density. Defining nominal parameter values as $\hat{w}_{xy}$ (for x and y taking i and e), we can then define linear parameter perturbations for a given region a as:

$$\begin{aligned} w_{ee}^{(a)} &= \hat{w}_{ee}(1 + R_e^{(a)}), \quad & w_{ie}^{(a)} &= \hat{w}_{ie}(1 + R_e^{(a)}), \\ w_{ii}^{(a)} &= \hat{w}_{ii}(1 + R_i^{(a)}), \quad & w_{ei}^{(a)} &= \hat{w}_{ei}(1 + R_i^{(a)}), \end{aligned} \tag{5}$$

where rescaling factors, R_e and R_i, represent relative variations in excitatory and inhibitory cell density, respectively (see **Figure 1C** for a visualization of how excitatory cell density varies across cortical areas). To map cell-density measurements to corresponding R_e and R_i values, we first z-score normalized raw excitatory and inhibitory cell-density data, as $e^{(a)}$ and $i^{(a)}$, respectively, across all regions, a. We then defined a simple proportional mapping to model parameters *via* a single scaling parameter, $\sigma \geq 0$, as

$$R_e^{(a)} = \sigma e^{(a)}, \quad R_i^{(a)} = \sigma i^{(a)}. \tag{6}$$

In this formulation, setting $\sigma = 0$ sets all $R_e^{(a)} = R_i^{(a)} = 0$ and reproduces the spatially homogeneous model; increasing σ increases the level of variation in coupling parameters across areas. Note that there is much scope for defining more complex mappings involving more new parameters, but defining the mapping from cell densities to model parameters in this simple, single-parameter scheme allows us to more clearly tackle our main aim to investigate how the model's dynamical features are shaped by such variation.

For a given system of coupled ODEs defined above, dynamics were simulated using The Virtual Brain (Sanz-Leon et al., 2015; Melozzi et al., 2017), yielding simulated time series for each region. The system was driven by white noise with a mean $\mu = 0$ and standard deviation $s = 1.3 \times 10^{-5}$ using the Euler–Maruyama method with a fixed time step, $\Delta t = 0.1$ ms, for a total simulation length of 1.2×10^5 ms, (or 2 min at 1,000 Hz). Initial transients of 1 s (1,000 time steps) were removed from all simulations to focus on the model's steady-state dynamics. As our aim was to understand the dynamical properties of the model that enable it to match the statistics of measured fMRI dynamics, we chose not to adjust the model output, $E^{(a)}(t)$, through a simulation of the hemodynamic response function to match the fMRI measurement [but could be done in future using, e.g., a convolution of a canonical hemodynamic response (Boynton et al., 1996) or a biophysical model (Friston et al., 2000; Kim and Ress, 2016)].

fMRI data for 100 wild-type mice are taken from Zerbi et al. (2021), and consisted of blood-oxygen-level-dependent (BOLD) signals recorded from 100 anesthetized mice measured at rest for a period of 15 min using a Biospec 70/16 small animal MR system operating at 7T, equipped with a cryogenic quadrature surface coil for signal detection (Bruker BioSpin AG, Fällanden, Switzerland). The data were processed (see **Supplementary Material** for details) and parcellated using the Allen Common Coordinate Framework (CCF v3). Using time-series data from each of the 37 cortical regions analyzed here, we computed a functional connectivity (FC) matrix for each mouse as pairwise Pearson correlations. These matrices were averaged across mice to yield a group-average FC that was used as the basis of comparison for computing FC–FC scores.

Models were assessed on their ability to reproduce the pairwise linear correlation structure (FC) of empirical mouse fMRI data, as the Spearman correlation between predicted and measured FC values: the FC–FC score, ρ_{FCFC}. While we focused here on reproducing pairwise linear correlations using ρ_{FCFC}, we note that a more comprehensive evaluation of model fit, incorporating aspects of local dynamics and dynamic FC properties, will be important for future investigations to more fully evaluate the rich patterns contained in the dynamics (Cabral et al., 2017; Aquino et al., 2021; Deco et al., 2021). To account for variability in simulated model dynamics due to a finite simulation time and different random seeds, we computed ρ_{FCFC} for 40

repeats of each simulation using different random seeds. Code for reproducing the simulations and analysis presented here is available at https://github.com/DynamicsAndNeuralSystems/MouseBrainModelling.

3. RESULTS

Here, we aim to understand the dynamical principles underlying coupled dynamical models using a neural mass model of the mouse cortex. First, in section 3.1, we investigate the spatially uniform case in which all brain regions are governed by identical dynamical rules. Focusing on model behavior in the vicinity of saddle-node and Hopf bifurcations, we characterize the model's dynamical regimes that best capture empirical FC structure. We then investigate the spatially heterogeneous case in section 3.2, in which regional variations in parameters are introduced according to variations in excitatory and inhibitory cell-density maps (Erö et al., 2018), which shape the model's local bifurcation properties and resulting dynamical regimes.

3.1. What Dynamical Features Drive High Model Performance?

We first characterize the model's dynamical regimes that best capture the pairwise correlation structure of experimental mouse fMRI, with the aim to understand how the positioning of individual nodes (brain regions) around specific types of bifurcations affects the model's ability to capture empirical FC. In this section we focus on a homogeneous model, in which all brain areas are governed by the same dynamical rules, but differ in their inputs from other regions (*via* the connectome). We characterize the model's behavior in each of three regimes: (i) in the vicinity of a single stable equilibrium, which we denote as the "Fixed Point" regime [using parameters adapted from Sanz-Leon et al. (2015)]; (ii) in the vicinity of a bistable region separated by saddle-node bifurcations, which we denote as the "Hysteresis" regime [using parameters from Heitmann et al. (2018)]; and (iii) in the vicinity of a pair of Hopf bifurcations, denoted as the "Limit Cycle" regime [using parameters from Borisyuk and Kirillov (1992)]. Parameter values for each of these three regimes are given in **Supplementary Table S2**. Bifurcation diagrams of excitatory firing, E, as a function of net external input, $J_{\text{tot}} = J_e - B_e$, are plotted for the Fixed Point regime (**Figure 2D**), Hysteresis regime (**Figure 2E**), and Limit Cycle regime (**Figure 2F**). These plots show how stable states of E vary with J_{tot} as solid lines (with unstable states shown for the bistable regime in **Figure 2E** and lower and upper limits of a limit-cycle oscillation in **Figure 2F**). They thus capture the rules underlying the dynamical behavior of individual brain regions in response to their aggregate input from other brain regions, J_{tot}, with each parameter setting defining a qualitatively different set of accessible dynamics, and types of response to inter-regional inputs. Importantly, these basic bifurcation structures, and the insights we gain from them, are not specific to the W–C model but are common features of many dynamical models (Strogatz, 2018).

We can understand the dynamics of an individual brain region in terms of the variation in its inputs over time, $J_{\text{tot}}(t)$ (recalling that J_{tot} is high when a region has many inputs from other high-activity, or high-E, regions). To understand this in more detail, we consider two key parameters that control the range of J_{tot} that can be explored by a given brain region. As per Equation (4), these parameters are: (i) the excitatory firing threshold, B_e, which contributes a constant offset to J_{tot}; and (ii) the global coupling constant, G, which scales each region's response to excitatory inputs from other connected regions. Increasing B_e decreases J_{tot} for all cortical regions, shifting the range of J_{tot} explored by network nodes to the left on the J_{tot}–E bifurcation diagram. We can see this from the annotated levels of input, J_{tot}, corresponding to selected B_e values (when $J_e = 0$) as vertical dashed lines in **Figures 2D–F**, which denote the minimum J_{tot} for selected B_e values. Low $G \approx 0$ removes the effect of inter-regional coupling altogether ($J_e \approx 0$), resulting in a very narrow range of J_{tot} around B_e, while increasing G allows individual regions to respond more strongly to external inputs, and thus span a greater range of J_{tot} values. Given fixed values of B_e and G, the key factor controlling how different brain regions differ in J_{tot} in the homogeneous model is their connected neighbors, with high in-degree regions having more inputs and thus the potential to achieve a higher J_e and J_{tot} than low in-degree regions.

We are now able to analyze how different values of B_e (which adjusts the baseline of J_{tot}) and G (which scales the excitatory inputs, J_e, relative to this baseline) shape the dynamics of a given brain region. For example, some combinations of B_e and G confine all nodes in the network to a fixed-point attractor, whereas others allow some nodes to span one (or multiple) bifurcations. Different choices of G and B_e control the diversity of dynamical features supported by the model, but what types of configurations yield high FC–FC scores, ρ_{FCFC}? Our results, comparing across a range of both G and B_e, are shown as heat maps for the Fixed Point (**Figure 2A**), Hysteresis (**Figure 2B**), and Limit Cycle (**Figure 2C**) regimes. To visualize the correspondence between points in G–B_e space and the resulting range of $J_{\text{tot}}(t)$ (and hence accessible dynamical regimes) they correspond to in the model simulation (range taken across time and nodes), we annotated this range in **Figures 2D–F** for key selected points in each corresponding heat map—labeled as "i," "ii," etc. For example, points toward the left of the G–B_e heat map correspond to low G and thus narrowing the range of J_{tot}, while points near the top of the heat map correspond to high B_e and hence low baseline inputs; hence points labeled "i" correspond to low and narrow ranges of J_{tot}, as annotated to the bifurcation diagrams in **Figures 2D–F**.

We first note a wide range of ρ_{FCFC} in all cases, indicating that model performance depends strongly on the local response to inputs, G and B_e, and thus the types of dynamical regimes available to the nodes of the coupled network. We also see that each dynamical regime exhibits characteristic regions of G–B_e space in which there is high FC–FC correspondence (colored red in **Figures 2A–C**), which reaches as high as $\rho_{\text{FCFC}} = 0.52$ (for the Fixed-Point regime), $\rho_{\text{FCFC}} = 0.50$ (Hysteresis regime), and $\rho_{\text{FCFC}} = 0.56$ (Limit-Cycle regime). All three model regimes can capture FC better than the direct correlation between SC and FC, $\rho_{\text{SCFC}} = 0.42$, indicating a benefit of accounting for distributed dynamics *via* coupled dynamical equations in capturing FC.

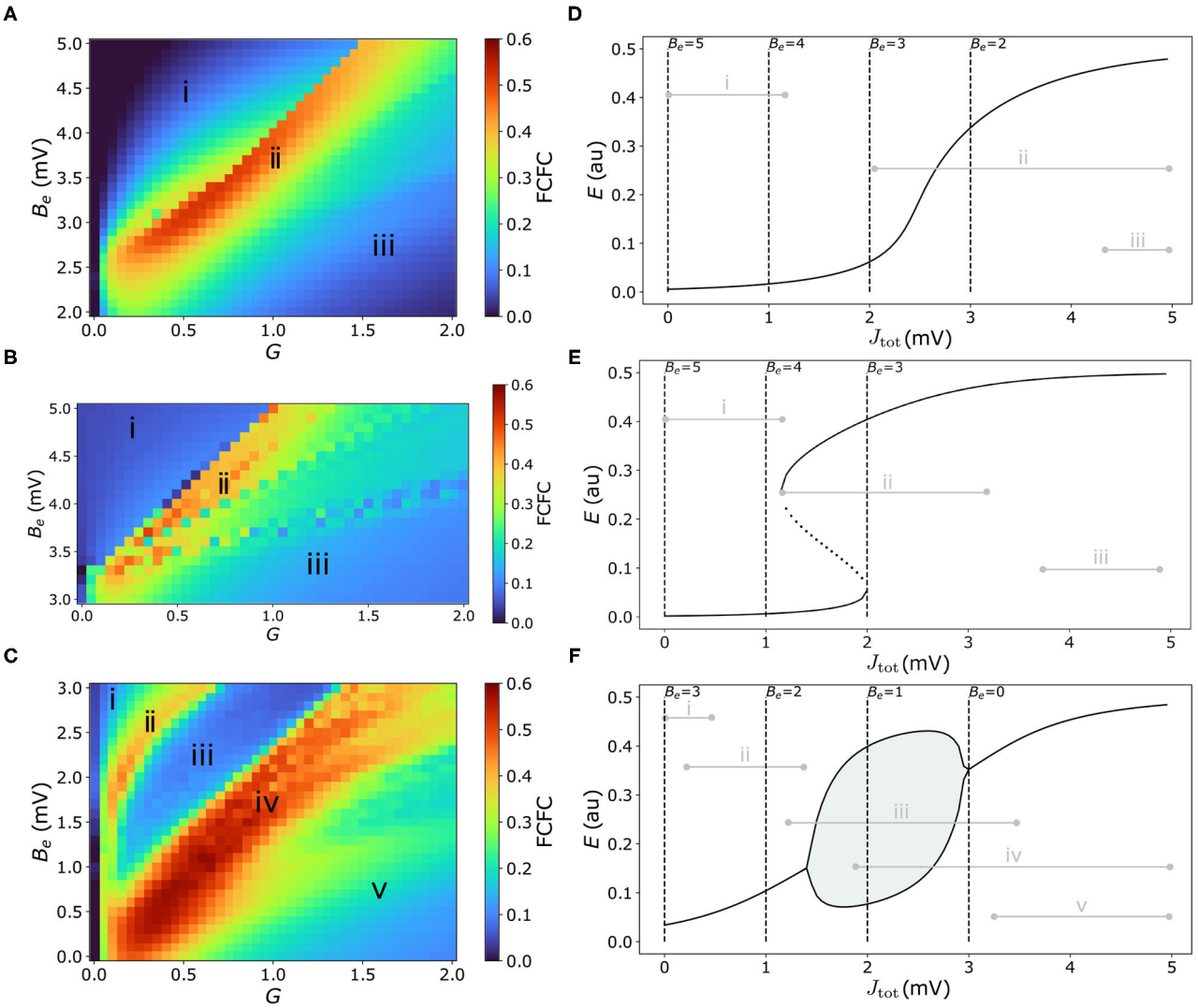


FIGURE 2 | Model performance is highly sensitive to the types of dynamical features available to the coupled dynamical network, with high FC–FC found near bifurcations and where external inputs have strong dynamical responses. **(A–C)** FC–FC score between model and data is plotted as a heat map in G–B_e space for the three model regimes considered here (see text): **(A)** "Fixed-point" regime, **(B)** "Hysteresis" regime, and **(C)** "Limit-cycle" regime. Corresponding J_{tot}–E bifurcation diagrams [cf. Equation (4)] for each regime are shown in the right-hand panels **(D–F)**, showing stable E fixed points (solid), unstable E fixed points (dotted), and minima and maxima of limit-cycle oscillations (solid lines with shading). Dashed vertical lines represent the minimum J_{tot} corresponding to selected B_e values. Gray horizontal lines represent the range of J_{tot} values across regions and time for a sample simulation from the corresponding point in G–B_e space annotated in **(A–C)**. Parameter values for each regime are in **Supplementary Table S2**.

Furthermore, model performance is consistent with, or higher than recently reported results for mouse cortex using a reduced Wong–Wang model (Wong and Wang, 2006) in a bistable regime [and using a Balloon–Windkessel BOLD filter (Friston et al., 2000) and a linear correlation, ρ_{FCFC}]: $0.35 \lessapprox \rho_{FCFC} \lessapprox 0.50$ (Melozzi et al., 2019).

To understand how the model can produce high FC–FC, $\rho_{FCFC} = 0.52 \pm 0.03$, in the Fixed Point regime (**Figure 2A**), we start by exploring the qualitatively different types of input–output responses in **Figure 2D**. At high excitatory firing threshold, B_e, and low coupling, G (labeled "i" in **Figures 2A,D**), nodes can only access the relatively flat, low-E steady-state branch, weakening inter-regional communication across the brain and leading to poor FC–FC. A similar suppression of inter-regional communication, and resulting low ρ_{FCFC}, occurs when the model is confined to the upper branch at low B_e and high G (labeled "iii" in **Figures 2A,D**). In the intermediate region, labeled "ii" in **Figures 2A,D**, we obtain high FC–FC scores, up to a maximum $\rho_{FCFC} = 0.52 \pm 0.03$ (at $B_e = 3.3$ mV, $G = 0.65$ mVs). Here, brain areas can access the sharp gradient of the sigmoid-like stable branch in E, and are thus highly sensitive in their response to variations in the activity of neighboring brain regions. This gives

us the somewhat surprising result that this very simple model, in a regime in which regions respond to the aggregate activity of their neighbors (but without any complex local dynamical features like bifurcations or oscillations) can produce high $\rho_{\mathrm{FCFC}} = 0.52$, consistent with results reported recently using more complex models (Melozzi et al., 2019). This is qualitatively consistent with direct structural connections providing a strong constraint on the resulting FC (Grandjean et al., 2017), with non-direct interactions providing a more minor perturbation (Robinson, 2012).

We next investigated a "Saddle Node" model regime [using parameters from Borisyuk and Kirillov (1992)], that involves a pair of saddle-node bifurcations with an intermediate bistable region, shown in **Figures 2B,E**. We obtained qualitatively similar results to the Fixed-Point regime analyzed above: ρ_{FCFC} is low when nodes are confined to relatively flat low-E branch (at low G and high B_e, labeled "i" in **Figures 2B,E**) or the high-E branch (high G and low B_e, labeled "iii" in **Figures 2B,E**), where responses to external inputs are weak. Stronger FC–FC scores (e.g., a maximum $\rho_{\mathrm{FCFC}} = 0.50 \pm 0.14$ at $B_e = 3.7\,\mathrm{mV}$ and $G = 0.35\,\mathrm{mVs}$) again arise in the intermediate region, where the local activity response is most sensitive to driving inputs, J_{tot} (labeled "ii" in **Figures 2B,E**). The difference now is the increased diversity of supported dynamics: regions coexist between the stable low-E and high-E states, and can switch between them. This bistability leads to a greater dynamical repertoire of regions in the network, including longer-timescale switching (cf. **Supplementary Figure S2**), but this is not reflected in an improved ρ_{FCFC}.

Finally, we investigated model dynamics in the neighborhood of a stable limit cycle, separated by two Hopf bifurcations [model parameters from Heitmann et al. (2018)], shown as a J_{tot}–E bifurcation diagram in **Figure 2F**. As for the two regimes studied above, when nodes are confined to a relatively flat stable branch, labeled "i" and "v" in **Figures 2C,F**, FC–FC scores are low. For a similar reason, we also find low FC–FC when nodes are confined to a limit-cycle oscillation (labeled "iii" in **Figures 2C,F**), where nodes have a restricted ability to respond to their inputs in a way that their neighbors can meaningfully respond to [since nodes are coupled *via* E, cf. Equation (3)]. But the heat map in **Figure 2C** reveals two regions of G–B_e space with high ρ_{FCFC}, labeled "ii" and "iv." In the region labeled "ii" (e.g., $\rho_{\mathrm{FCFC}} = 0.38 \pm 0.03$ at $B_e = 2.8\,\mathrm{mV}$, $G = 0.45\,\mathrm{mVs}$), nodes sit on a stable branch which has a small but sufficient curvature to enable local activity to respond, albeit weakly, to inputs from connected regions. But the best fits to data, reaching $\rho_{\mathrm{FCFC}} = 0.56 \pm 0.04$ (at $B_e = 1.5\,\mathrm{mV}$, $G = 0.7\,\mathrm{mVs}$), are found in the region labeled "iv" in **Figures 2C,F**. In this region of B_e–G space, nodes can access two distinctive types of dynamics: the limit-cycle regime (at low J_{tot}) and the high-E fixed-point attractor (at high J_{tot}). High FC–FC scores are also obtained when nodes can also access the low-E stable branch (at high G and high B_e). Importantly, the high-E branch at high J_{tot} has a relatively sharp dependence on J_{tot}, a feature that is common to obtaining high-ρ_{FCFC} scores in all three model regimes. Together, our analyses in this section demonstrate the importance of a model that allows nodes to respond sensitively to inputs from their network neighbors for reproducing FC.

3.1.1. Interpreting Simulated Dynamics in Terms of Bifurcation Diagrams

Bifurcation diagrams provide an understanding of the dynamical regimes accessed by individual nodes, and the way in which they respond to changes in inputs, information that can guide understanding of the complex distributed dynamics that result from a full model simulation. For the Limit-Cycle regime, simulated multivariate time series and corresponding FC matrices are plotted in **Figure 3** for points labeled "ii," "iii," and "iv" in **Figures 2C,F**. In "ii," nodes are confined to the low-E stable branch and, accordingly, the dynamics consist of noisy deviations from a low-E stable fixed point (**Figure 3D**). These perturbations can drive changes in structurally connected nodes, yielding weak pairwise correlations shown in **Figure 3A**. In "iii," when nodes are mostly confined to the limit cycle and ρ_{FCFC} is low, most nodes exhibit oscillations (with some longer-timescale deviations, cf. **Figure 3E**), and a minority of other nodes (situated near the low-J_{tot} Hopf bifurcation) move between noisy deviations from the low-E stable branch and oscillatory limit-cycle dynamics. This results in very high pairwise correlations between groups of synchronized oscillatory nodes, $r > 0.8$, such that the underlying structural connections play less of a role in shaping the pairwise correlation structure, resulting in a low FC–FC score. In "iv," with the highest ρ_{FCFC}, the Hopf bifurcations facilitate complex spatiotemporal dynamics shown in **Figure 3F**. While many nodes spend most of the simulation near the high-E stable branch (those with high J_{tot}), we observe periods of time during which groups of nodes (near the Hopf bifurcation) display synchronized oscillations, embedded in globally complex and distributed dynamics on longer timescales. These analyses demonstrate how analyzing the response of local nodes to inputs, as ranges of J_{tot} in a bifurcation diagram (as in **Figures 2D–F**), can ground an understanding of the complex distributed dynamics that result from the full coupled model, which can be visualized effectively as heat maps (**Figure 3**).

3.1.2. Resolving Inter-regional Differences in Inputs

The variation in qualitative dynamics across individual brain areas in the multivariate time series plotted in **Figures 3D–F** indicates that different network elements are accessing different dynamical regimes permitted by the model, resulting from substantial variability in the $J_{\mathrm{tot}}(t)$ experienced by different nodes. Since all nodes are governed by the same dynamical rules, and, hence, the same bifurcation diagrams, we can annotate $J_{\mathrm{tot}}(t)$ ranges onto a common bifurcation diagram to understand how the dynamics of individual regions are governed by different types of inputs from their connected neighbors. That is, rather than plotting just the overall range of J_{tot} (from the minimum to the maximum across all nodes), as in **Figures 2D–F**, we can resolve the individual ranges of J_{tot} experienced by each individual node on the J_{tot}–E bifurcation diagram. An example is shown for the Limit-Cycle regime at "iv" in **Figure 4A** [where we have plotted J_e instead of J_{tot}, equivalently, for a fixed $B_e =$

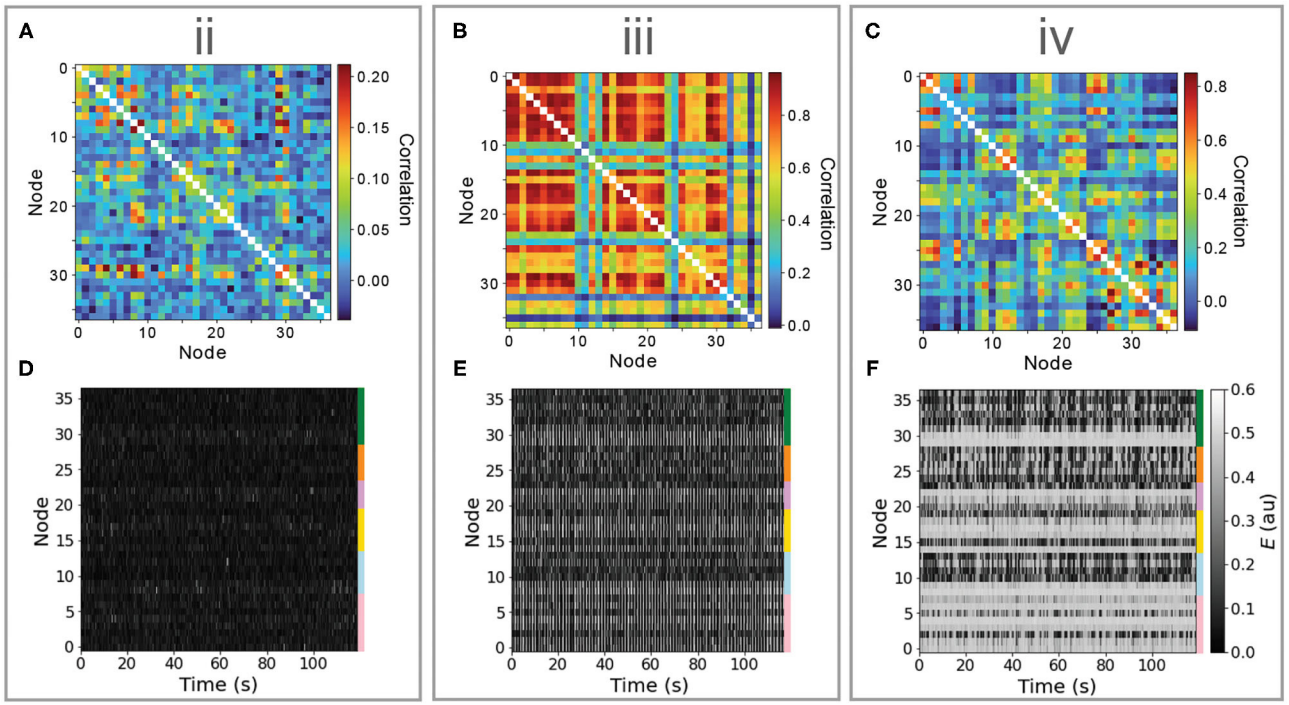


FIGURE 3 | Different dynamical features of the limit-cycle regime yield very different dynamics, including noisy deviations about a stable fixed point, synchronous oscillations, and a complex distributed dynamics featuring intermittent synchronization with high FC–FC. Here, we investigate simulated time series (lower row) and functional connectivity matrices (upper row) for three regions in B_e–G space annotated "ii," "iii," and "iv" in **Figure 2**. **(A–C)** Simulated functional connectivity matrices are plotted for "ii," "iii," and "iv," respectively. **(D–F)** Simulated E time series are plotted as a node × time heat map (or "carpet plot" Aquino et al., 2020) for all brain regions for "ii," "iii," and "iv," respectively. Colored bars label the six cortical divisions listed in **Supplementary Table S1**. In all plots, nodes are ordered as per **Supplementary Table S1**.

1.5 mV, cf. Equation (4)]. We see how, even with fixed dynamical rules, the range of J_e experienced by individual nodes varies markedly. Some regions have low J_e across the simulation, like the dorsal retrosplenial area, RSPd (annotated in **Figure 4A**), and, therefore, only display oscillations, as plotted in **Figure 4B**. Other regions with high J_e across the simulation, like the posterior parietal association areas, PTLp (annotated in **Figure 4A**), are confined to the stable high-E branch across the full simulation and display dynamics consistent with noisy deviations from a fixed point, as shown in **Figure 4B**. Regions like the ventral retrosplenial area, RSPv (annotated in **Figure 4A**), span the Hopf bifurcation, and thus exhibit more complex patterns that contain both oscillatory dynamics and noisy excursions about a stable fixed-point, depending on fluctuations in inputs, $J_e(t)$. The short samples of $E(t)$ for six annotated Medial regions in **Figure 4B** reveal some of these dynamics, including dynamic phase relationships between the oscillatory populations. These findings demonstrate the usefulness of interpreting the dynamics of coupled mass models in terms of time-varying inputs to the constituent populations.

3.2. Understanding Heterogeneity in Local Dynamical Rules

Above, we used bifurcation diagrams to show that complex distributed dynamics in a neural-mass model can be understood in terms of the responses of individual regions to inputs from their connected neighbors. Despite equivalent local dynamical rules, and hence identical bifurcation diagrams for all brain regions, we found substantial inter-regional variability in accessible dynamical regimes and resulting activity dynamics, due to differences in structural connectivity and resulting $J_e(t)$. In this section, we aim to understand the effect of varying the local dynamical rules themselves, by incorporating spatial heterogeneity in the properties of local cortical circuits (*via* a corresponding variation in model parameters). Specifically, we varied excitatory and inhibitory coupling strengths of individual brain areas according to excitatory and inhibitory cell-density data (Erö et al., 2018). We focused on the Limit Cycle regime of the W–C model described above, which displayed the richest dynamical repertoire and highest ρ_{FCFC}. As described in section 2, we used relative variations in excitatory and inhibitory cell densities across cortical areas to define a corresponding variation in R_e and R_i, which proportionally adjust coupling parameters—w_{ii}, w_{ie}, w_{ei}, w_{ee}—across brain areas. Setting $R_e = R_i = 0$ for all areas recovers the homogeneous model studied above [see Equation (5) for details]. This simple formulation allows us to understand how varying the excitatory and inhibitory coupling parameters across areas, in accordance with underlying excitatory and inhibitory cell densities, shape the dynamical responses of individual areas to inputs, and, hence, the resulting model dynamics.

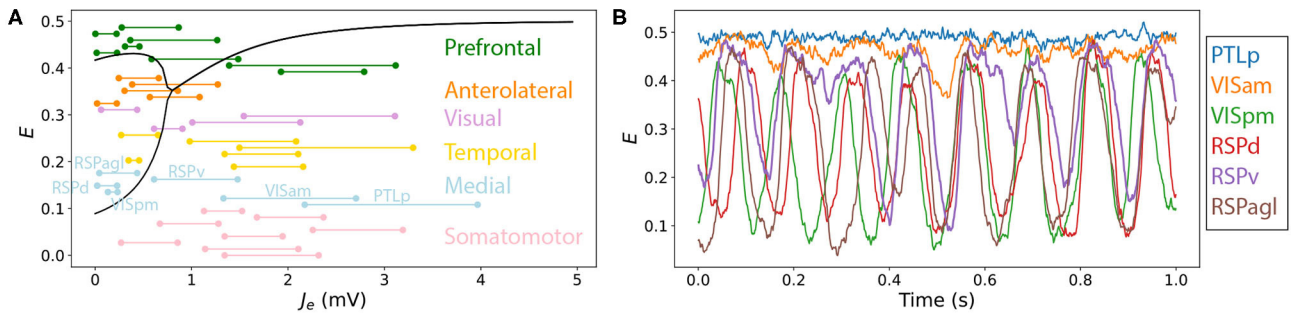


FIGURE 4 | Resolving different ranges of inputs, J_e, experienced by different network nodes allows us to understand their variable dynamical behavior in a coupled network model. Here we focus on the point labeled "iv" in the Limit-Cycle regime (**Figures 2C,F**), $B_e = 1.5$ mV, $G = 0.7$ mVs, in which nodes differ substantially in their inputs, J_e, and hence their resulting dynamics. **(A)** Bifurcation diagram for E and a function of J_e (as **Figure 2F**), with ranges of net excitatory drive, J_e, across the model simulation annotated for each brain region (colored according to the six labeled divisions). All regions are ordered according to **Supplementary Table S1**, and are labeled for the six Medial regions, which are plotted in **(B)**. **(B)** E time series for the six Medial regions—PTLp, VISam, VISpm, RSPd, RSPv, and RSPagl—shown for the final 1 s of the simulation.

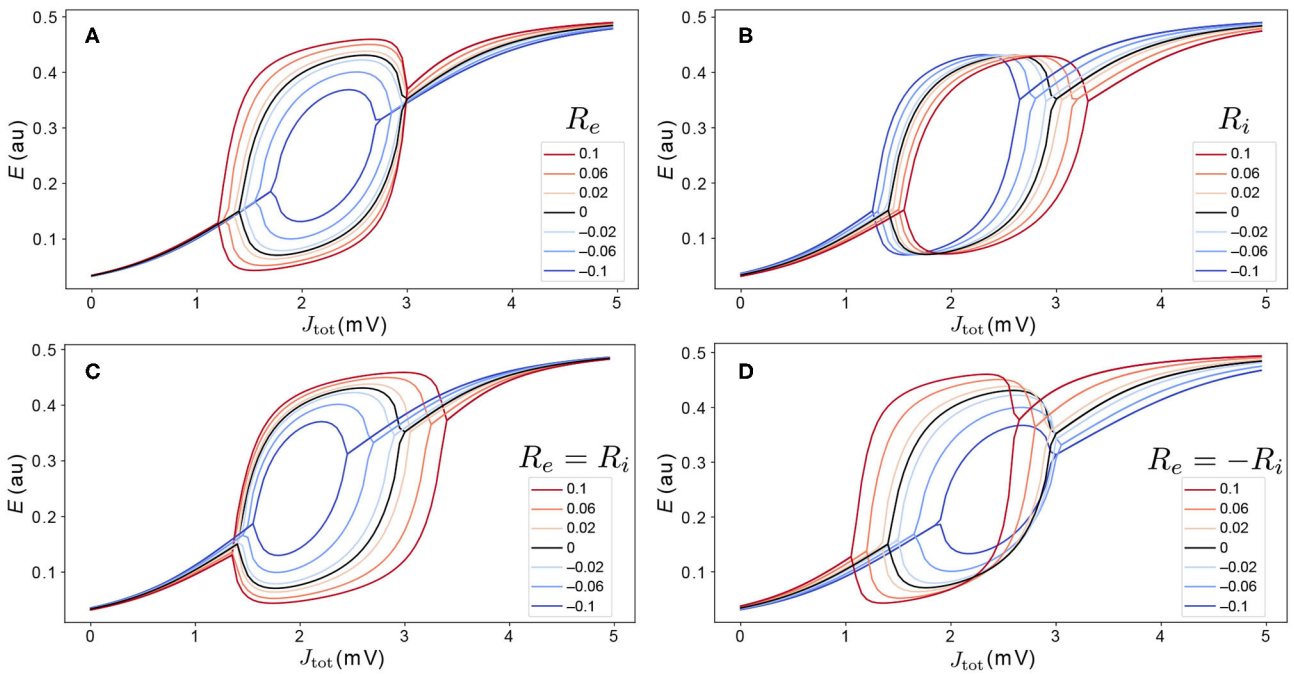


FIGURE 5 | Variations in excitatory and inhibitory cell density modify the dynamical regimes accessible to cortical regions. We model the effect of variations in excitatory and inhibitory cell density *via* perturbation parameters R_e and R_i, respectively, as defined in Equation (5). Relative to the nominal bifurcation diagram, $R_e = R_i = 0$ (black), we investigate variations in $-0.1 \leq R_e \leq 0.1$ and $-0.1 \leq R_i \leq 0.1$. Four types of variation were investigated: **(A)** R_e only ($R_i = 0$); **(B)** R_i only ($R_e = 0$); **(C)** R_e and R_i, such that $R_e = R_i$; and **(D)** R_e and R_i, such that $R_e = -R_i$. The legend indicates values of R_e.

3.2.1. Levels of Excitation and Inhibition Perturb Bifurcation Diagrams

To understand how variations in R_e and R_i affect model dynamics, we first analyze how these parameters shape the $J_{\rm tot}$–E bifurcation diagrams for an individual area. The effect of ±10% variations to coupling parameters (corresponding to the ranges $-0.1 < R_e < 0.1$ and $-0.1 < R_i < 0.1$), are shown as $J_{\rm tot}$–E bifurcation diagrams in **Figure 5**, varying just R_e (**Figure 5A**), just R_i (**Figure 5B**), R_e and R_i together with $R_e = R_i$ (**Figure 5C**), and R_e and R_i such that $R_e = -R_i$ (**Figure 5D**). We find that even these relatively small, ≈ 10%, perturbations have a substantial effect on the dynamical responses of individual areas, affecting: (i) the range of $J_{\rm tot}$ over which model exhibits stable oscillations; (ii) the oscillation amplitudes themselves; and (iii) steady-state activity levels. As shown in **Figure 5**, cortical areas with a higher excitatory cell density, R_e, have higher-amplitude oscillations, a wider range of $J_{\rm tot}$ over which stable oscillations are exhibited, and, for the

same J_{tot}, increased activity, E, in the upper branch. Different changes result from modifying the inhibitory cell density, shown in **Figure 5B**: increasing R_i shifts the same bifurcation and fixed-point structure to higher J_{tot} (equivalent to raising the firing threshold, B_e). That is, regions with higher inhibitory cell density, R_i, require a greater aggregate input, J_e, to produce the same dynamics. Varying both R_e and R_i, shown in **Figures 5C,D**, yields combinations of the individual perturbations from R_e and R_i individually. These results demonstrate how relatively small variations in excitatory and inhibitory coupling parameters can have large effects on the bifurcation structure and dynamical regimes exhibited by local cortical regions. The effects are more dramatic for R_e and R_i values in the range from -0.5 to 0.5, where the Hopf bifurcations can be removed altogether from the Limit Cycle regime (**Supplementary Figure S3**), or additional stable states can be added *via* saddle-node bifurcations in the Hysteresis regime (**Supplementary Figure S4**).

3.2.2. Understanding Mouse Cortical Model Dynamics Constrained by Excitatory and Inhibitory Cell Densities

In the heterogeneous model, individual different brain areas differ both in their J_{tot} values that they receive from their coupled neighbors (due to differences in their structural connections), but also have different dynamical rules, due to different individual combinations of R_e and R_i values. As demonstrated above for the homogeneous model, this understanding of the dynamical responses of individual brain areas to inputs from across the network is crucial to guiding understanding of the complex, distributed dynamics of the full coupled model. In this section, we explore how the impact of local variations in R_e and R_i can be visualized and used to understand the dynamics of the full coupled model. Recall that our heterogeneous model is formulated with a single new parameter, σ, that defines how strongly relative differences in excitatory and inhibitory cell densities are mapped to corresponding changes in the model's coupling parameters (Equation 6). For the Limit-Cycle regime, we investigated how FC–FC scores change as we introduce a greater degree of inter-areal heterogeneity, σ. The variation in ρ_{FCFC} as a function of σ across the range $0 \leq \sigma \leq 1$ is shown in **Figure 6E**. We did not find a substantial increase in ρ_{FCFC} when incorporating heterogeneity, $\sigma > 0$, although there was a modest improvement relative to the homogeneous model ($\sigma = 0$) for $\sigma = 0.2$, yielding $\rho_{\text{FCFC}} = 0.60 \pm 0.05$. Testing this result against a null distribution (obtained by repeating the procedure but with randomly permuted excitatory and inhibitory cell-density data) using a permutation test yielded $p \approx 0.15$, indicating that $\rho_{\text{FCFC}} = 0.60$ does not constitute a significant improvement relative to the homogeneous model (see section 2 for details). As we discuss later, this result may be contributed to the small number of regions in the model, the simplicity of the dynamical equations, or the dominance of SC in constraining FC in the anesthetized mouse (Grandjean et al., 2017).

While we did not find evidence of a significant improvement in FC–FC score from a simple incorporation of heterogeneity, our main aim was to demonstrate how tools from dynamical systems can help to understand the complex coupled dynamics of such a spatially heterogenous model. We used the point, $\sigma = 0.2$, inferred above as a suitable example for this purpose. With $\sigma = 0.2$, we plotted J_e–E bifurcation diagrams for all brain regions on the same plot in **Figure 6A**. The plot shows how differences in excitatory and inhibitory cell densities results in different bifurcation diagrams, that correspond to similar qualitative changes as analyzed in **Figure 5** above. Specifically, brain regions now differ substantially in their: critical values, J_e, that separate limit cycle from fixed-point dynamics; ranges of J_e in which oscillations are stable; oscillation amplitudes; and fixed-point activity levels, E, in the upper branch (for a given J_e). Compared to the homogeneous model, two regions with the same input, J_e, no longer indicates that they will be subject to the same dynamical rules.

To understand how these changes in local dynamical rules affect the resulting cortical dynamics, we next plotted the range of J_e that each node experiences across the simulation. As shown in **Figure 6B**, this can be represented as a horizontal line, distinguishing J_e values corresponding to what stable dynamical feature—"limit cycle" or "fixed point"—according to each region's individual bifurcation structure. This results in a richer dynamical landscape for the model: some brain regions can access both stable limit cycle and fixed-point dynamics, others can only access the high-E fixed-point equilibrium, while others can access just the limit-cycle attractor. It is useful to connect the range of dynamical regimes each region accesses across the simulation, shown in **Figure 6B**, with the E dynamics themselves, shown in **Figure 6C**. We can clearly see the high-E regions on the upper stable branch, as well as the more complex intermittent oscillations of regions that can access limit-cycle dynamics. The functional connectivity matrix from this simulation is shown in **Figure 6D**. This representation of pairwise correlations in the model dynamics hides much of the richness of the individual time series themselves (**Figure 6C**), and the dynamical rules that underlie them (**Figures 6A,B**). The ability to represent qualitative dynamical regimes of individual regions in a coupled network model—as J_e–E bifurcation diagrams with individual ranges of J_e explored for each region—provides a powerful illustration of the dynamics supported by the coupled components of a complex networked dynamical model.

4. DISCUSSION

In this article, we developed a neural-mass model of the mouse cortex. We showed how bifurcation diagrams can be used to understand how regional differences in dynamics result from differences in inputs, J_{tot}, and delineated the types of dynamical regimes that yield good fits to experimental functional connectivity. We first analyzed a homogeneous model in which all regions are governed by identical dynamical rules to show how regional variations in dynamics result from differences in inputs (driven by differences in structural connectivity). We then extended this treatment to a heterogeneous model in which the bifurcation structures themselves vary across regions due to variation in local excitatory and inhibitory cell densities. Our results provide a useful framework for understanding the

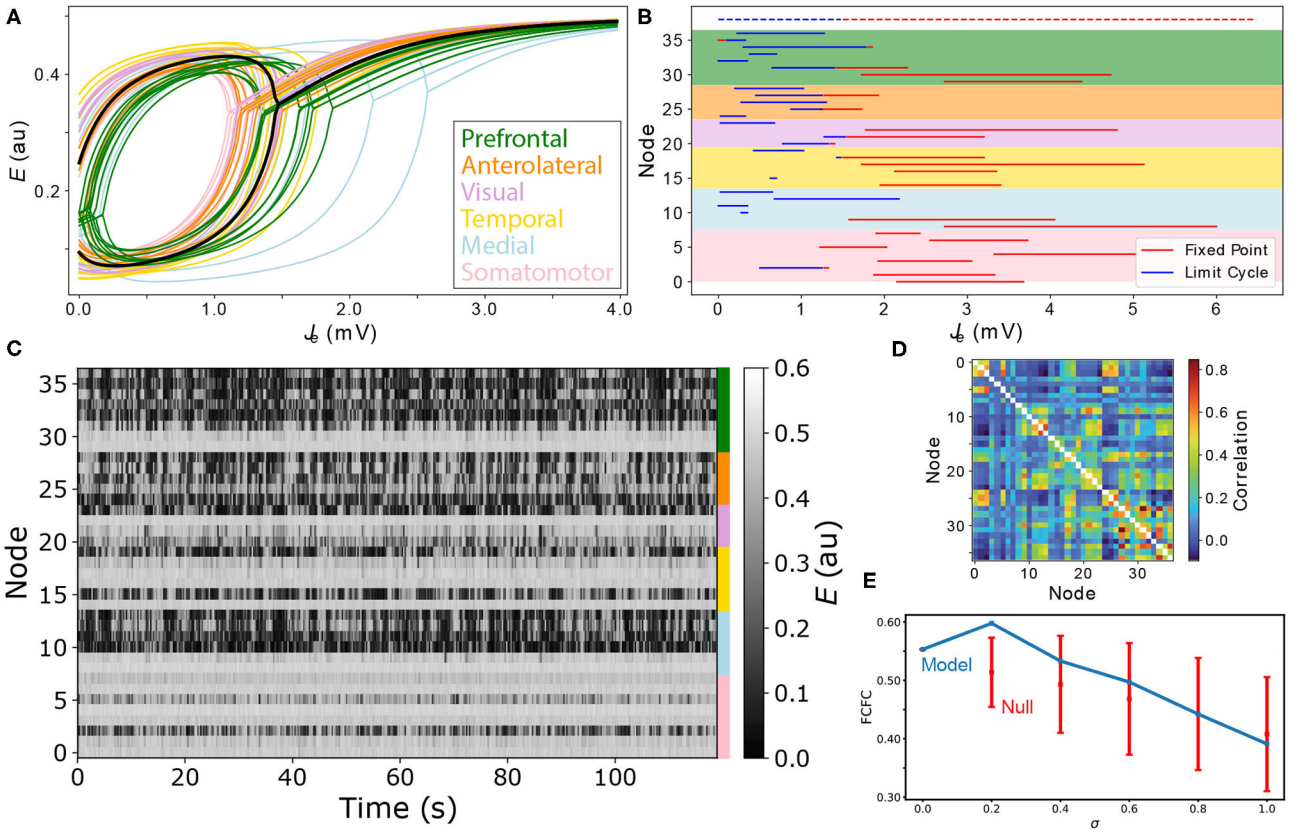


FIGURE 6 | Modeling spatial variation in local excitatory and inhibitory cell densities produces complex distributed dynamics. **(A)** Bifurcation diagrams are plotted for all cortical areas according to their excitatory and inhibitory cell densities. Regions are colored according to their labeled anatomical grouping and the homogeneous case ($R_e = R_i = 0$) is shown in black for comparison. **(B)** The type of equilibrium dynamics displayed by a given cortical region, limit cycle (blue) or fixed point (red), is plotted as a function of $J_e - B_e$ for all cortical regions for the range of $J_e - B_e$ they experience across the model simulation. Nodes are ordered as per **Supplementary Table S1** and shading reflects the six anatomical groupings labeled in A. Dashed lines shown at the top correspond to the uniform case ($R_e = R_i = 0$) for comparison. **(C)** Simulated time series for all brain areas are plotted as a heat map. Colors annotated to the right label the six anatomical groupings listed in A. **(D)** Simulated functional connectivity matrix. **(E)** FC–FC score as a function of the scaling parameter, σ, Equation (6). Results are shown for the model constrained by excitatory and inhibitory cell-density data (blue) and the permutation-based null distribution shown as mean ± standard deviation (red).

mechanisms that underlie complex simulated model dynamics, using a combination of local bifurcation diagrams (annotated with ranges of inputs for different regions) and visualizations of the multivariate time-series dynamics [as "carpet plots" (Aquino et al., 2020)]. These analyses will be particularly important for understanding how the brain's microscale circuits give rise to the complex distributed dynamics observed in brain-imaging experiments. A common scientific goal of modeling a system is to accurately reproduce important properties of it, while also gaining an understanding of how it does so. While successful approaches have been demonstrated for maximizing goodness of fit [sometimes optimizing large numbers of parameters (Wang et al., 2019; Kong et al., 2021; Wischnewski et al., 2021)], obtaining understanding is a key challenge for complex nonlinear models of brain dynamics. The analyses and visualizations demonstrated in this work aim to provide an understanding of the model dynamics in terms of the dynamical regimes that individual regions can access, shaped by their inputs from coupled neighbors. Key analyses include: (i) assessing the role of input parameters B_e and G in shaping empirical FC fits in terms of corresponding ranges spanned across J_e–E bifurcation diagrams (**Figure 2**); (ii) annotating of J_e for all cortical regions onto a common bifurcation diagram (**Figure 4A**); (iii) analyzing perturbations to bifurcation diagrams due to variations in local circuit properties (**Figure 5**); and (iv) annotating region-specific qualitative equilibrium dynamics across ranges of J_e for all regions in a single plot (**Figure 6B**). As whole-brain models develop to incorporate whole-brain datasets—including whole-brain maps of gene-expression and cell types (Fulcher et al., 2019; Yao et al., 2021)—these types of analyses will be crucial to understanding how this complexity shapes the underlying dynamical mechanisms, both at the level of individual brain regions, and their distributed interactions.

While many studies focus on determining an optimal working point, i.e., structural connectome scaling G, we find that the offset (B_e in the present model) is also critical in determining how local regions respond to inputs, and hence the resulting ρ_{FCFC}. We also found strong fits to empirical FC whenever brain regions were

able to respond to inputs with sufficient gain, likely reflecting the strong role of direct structural connections in shaping FC in the anesthetized mouse (Grandjean et al., 2017). In particular, even in the Fixed-Point regime, in which the model exhibits the most constrained dynamical repertoire, we report $\rho \approx 0.52$, consistent with other published results in the literature [FC–FC scores up to ≈ 0.5 (Melozzi et al., 2019) using a Wong–Wang model (Wong and Wang, 2006)]. Only a small improvement was found when the model operated near a Hopf bifurcation, $\rho_{\mathrm{FCFC}} = 0.56$. This highlights the ability of simple dynamical features to capture aspects of measured dynamics, consistent with prior comparisons demonstrating high performance of simple models (Messé et al., 2014, 2015; Nozari et al., 2021). The results also demonstrate the importance of comparing model performance against simpler benchmarks, and justifying increased model complexity only if it accompanies enhanced explanatory power.

Incorporating spatial variations in local dynamical rules according to whole-brain maps has immense potential in allowing us to connect new physiological understanding of neural circuits to the whole-brain dynamics that they enable. In this work, we incorporated spatial variations in excitatory and inhibitory cell densities as a corresponding perturbation to coupling parameters between E and I populations, with a single scaling parameter, σ. However, there are alternative ways in which this heterogeneity could be implemented and constrained in future, for example, by allowing σ to differ for excitatory (σ_e) and inhibitory (σ_i) populations. Incorporating more detailed physiological data into correspondingly more complex biophysical models (e.g., incorporating multiple inhibitory cell types), brings further parametric freedom that needs to be properly constrained from a combination of physiological and neuroimaging data. Our approach for assessing the improvement in ρ_{FCFC} after incorporating cell-density data involved a permutation approach against randomized assignment of the data to regions (preserving the match between e and i, but permuting their assignment to brain regions), and did not reveal a significant improvement relative to null gradients ($p \approx 0.15$). This may be due to the relatively small number of brain regions included, and the focus on FC–FC as an evaluation metric rather than a more comprehensive set of evaluations. Other ways of assessing the improvement of the spatially heterogeneous model could also be explored, such as testing the $e:i$ ratio against alternative spatial gradients [as Deco et al. (2021)], or taking an optimization approach to estimate the optimal e and i gradients, and then assess their similarity to the measured excitatory and inhibitory cell-density data [as Wang et al. (2019)].

As our aim here was to demonstrate methods for understanding the dynamics of coupled neural models with heterogeneity using a simple modeling approach, many aspects of the model could be improved in future work. First, we have focused here on a specific simple biophysical model, the Wilson–Cowan model (Wilson and Cowan, 1972, 1973; Cowan et al., 2016), that allowed us to incorporate variations in excitatory and inhibitory cell-density data. We have focused on the behavior of the model in three specific dynamical regimes (a fixed point with gain, hysteresis, and limit cycle), but the results should be qualitatively applicable to those same dynamical regimes of other models. However, we note that other models with different dynamical features may display different behavior, such as the Wong–Wang model (Wong and Wang, 2006; Deco et al., 2013, 2021; Murray et al., 2017; Demirtas et al., 2019; Wang et al., 2019), or models that incorporate cortico–thalamic interactions (Wilson and Cowan, 1973; Robinson et al., 2015; Lin et al., 2020; Müller et al., 2020). We also note that while our aim here was to understand the model dynamics directly, it is common practice to simulate a hemodynamic response, such as the Balloon–Windkessel model (Friston et al., 2000) or a more sophisticated hemodynamic response function (Aquino et al., 2014). Simulating a slower hemodynamic response would introduce challenges in mapping bifurcation diagrams in E to the corresponding BOLD dynamics, and could lead to substantial qualitative differences between the dynamics of the neural model and the HRF-filtered dynamics. As a result, our specific conclusions about model performance in different dynamical regimes may not generalize to different choices of hemodynamic responses, but this could be achieved in future work by attempting to construct an effective bifurcation diagram of the dynamics of the BOLD forward solution as a function of the model parameters. We note, however, the body of evidence showing improved performance of linear models over nonlinear, biophysically informed models, in capturing the dynamical properties of fMRI data (Messé et al., 2014, 2015), and a recent finding that the performance of nonlinear neural mass models can drop when including HRF (Nozari et al., 2021). This suggests that, in the absence of thoroughly validated neural-mass models at the level of population neural activity (Lin et al., 2020), and a clearly demonstrated improvement of a BOLD forward model, neural mass models may be more conservatively viewed as a phenomenological means of capturing different types of dynamics and dynamical interactions, for which our simple approach, here, is valid and useful.

We also highlight our relatively simple treatment of structural connectivity, as a binary adjacency matrix, even though estimates of axonal connectivity strengths vary over at least four orders of magnitude (Oh et al., 2014). It remains an open question what greater structural connection "strengths" (approximated by the number of axons connecting two brain areas), corresponds to dynamically, e.g., faster connection speeds, a stronger effect on local population mean dynamics, or some alternative type of response. While the model here does not include time delays (assuming fast inter-regional interactions on the timescale of neural dynamics), they are likely to be crucial in shaping the brain's distributed dynamics (Petkoski and Jirsa, 2019) and should be explored in future work. We next note a major simplifying assumption in using a neural-mass model, which involves representing the spatially continuous cortical sheet as a set of 37 discrete cortical areas, abstracted away from their physical embedding (Robinson, 2019). Given the spatial resolution of modern mouse-brain maps, and the often continuous spatial variation they reveal, it will be important to develop models that accurately capture this physical continuity, e.g., using a neural field approach (Robinson et al., 2003).

Finally, we highlight the limitation of evaluating our model with respect to its ability to match only the linear correlation

structure, FC, of the empirical dataset. fMRI data have a much richer dynamical structure than is captured by the static FC, including the dynamics of FC across a recording (Cabral et al., 2017; Demirtas et al., 2019; Aquino et al., 2021; Deco et al., 2021) and the organization of regional timescales (Sethi et al., 2017; Shafiei et al., 2020). For example, despite producing very different patterns in simulated time series, we found similar fits, ρ_{FCFC}, across the Fixed-Point, Hysteresis, and Limit-Cycle regimes of our homogeneous model, and when incorporating heterogeneity. The more complex distributed dynamics, including intermittent synchronization seen in carpet plots from the Limit Cycle regime (**Figure 3F**) and when incorporating regional heterogeneity (**Figure 6C**), qualitatively match the types of patterns seen in empirical fMRI dynamics better than in the fixed-point regime. This highlights the simplicity of the FC–FC score, ρ_{FCFC}, in capturing only the pairwise linear correlation structure in the data, and indicates the need for future work to perform a more comprehensive evaluation. This should include similar visualizations of model performance across B_e–G space (as **Figures 2A–C**), where the most distinctive models features for reproducing a greater range of characteristics of fMRI dynamics may be more clearly distinguished.

With the increasing availability of high-resolution neuroscience data, in space and time, the need for tools to provide interpretable accounts of their dynamics is pressing. Our work demonstrates a range of useful tools to analyze the behavior of coupled dynamical models of brain dynamics, helping them to provide understanding of the dynamical mechanisms that underpin their predictions. Our results emphasize the importance of benchmark comparison (e.g., a simple fixed-point model yields high FC–FC), visualization (e.g., very different dynamical patterns exhibited in carpet plots can yield similar correlation structures in FC), and proper statistical testing (e.g., while the heterogeneous model yields improved FC–FC, it is not significantly better than repeating the process on randomized data), practices that may help guide progress in the field.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author/s.

ETHICS STATEMENT

The animal study was reviewed and approved by the Ethical Committee of the Canton Zurich, Switzerland.

AUTHOR CONTRIBUTIONS

BF and EM contributed to conception and design of the study. PS performed all simulations and analysis, supervised by BF and EM. Mouse fMRI data were processed by VZ. PS wrote an initial draft of the manuscript, which was refined for submission by BF and EM. All authors contributed to manuscript revision.

FUNDING

This work was supported by the Physics Foundation, School of Physics, The University of Sydney.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fncom.2022.847336/full#supplementary-material

REFERENCES

Allegra Mascaro, A. L., Falotico, E., Petkoski, S., Pasquini, M., Vannucci, L., Tort-Colet, N., et al. (2020). Experimental and computational study on motor control and recovery after stroke: toward a constructive loop between experimental and virtual embodied neuroscience. *Front. Syst. Neurosci.* 14, 31. doi: 10.3389/fnsys.2020.00031

Aquino, K. M., Fulcher, B., Oldham, S., Parkes, L., Gollo, L., Deco, G., et al. (2021). On the intersection between data quality and dynamical modelling of large-scale fMRI signals. *NeuroImage.* 119051. doi: 10.1016/j.neuroimage.2022.119051

Aquino, K. M., Fulcher, B. D., Parkes, L., Sabaroedin, K., and Fornito, A. (2020). Identifying and removing widespread signal deflections from fMRI data: Rethinking the global signal regression problem. *NeuroImage* 212, 116614. doi: 10.1016/j.neuroimage.2020.116614

Aquino, K. M., Robinson, P. A., Schira, M. M., and Breakspear, M. J. (2014). Deconvolution of neural dynamics from fMRI data using a spatiotemporal hemodynamic response function. *NeuroImage* 94, 203–215. doi: 10.1016/j.neuroimage.2014.03.001

Borisyuk, R. M., and Kirillov, A. B. (1992). Bifurcation analysis of a neural network model. *Biol. Cybern.* 66, 319–325.

Boynton, G. M., Engel, S. A., Glover, G. H., and Heeger, D. J. (1996). Linear systems analysis of functional magnetic resonance imaging in human V1. *J. Neurosci.* 16, 4207–4221.

Breakspear, M. J. (2017). Dynamic models of large-scale brain activity. *Nat. Neurosci.* 20, 340–352. doi: 10.1038/nn.4497

Burt, J. B., Demirtas, M., Eckner, W. J., Navejar, N. M., Ji, J. L., Martin, W. J., et al. (2018). Hierarchy of transcriptomic specialization across human cortex captured by structural neuroimaging topography. *Nat. Neurosci.* 27, 889. doi: 10.1038/s41593-018-0195-0

Cabral, J., Kringelbach, M. L., and Deco, G. (2017). Functional connectivity dynamically evolves on multiple time-scales over a static structural connectome: models and mechanisms. *NeuroImage* 160, 84–96. doi: 10.1016/j.neuroimage.2017.03.045

Chaudhuri, R., Knoblauch, K., Gariel, M.-A., Kennedy, H., and Wang, X.-J. (2015). A large-scale circuit mechanism for hierarchical dynamical processing in the primate cortex. *Neuron* 88, 419–431. doi: 10.1016/j.neuron.2015.09.008

Choi, H., and Mihalas, S. (2019). Synchronization dependent on spatial structures of a mesoscopic whole-brain network. *PLoS Comput. Biol.* 15, e1006978. doi: 10.1371/journal.pcbi.1006978

Claudi, F., Tyson, A. L., Petrucco, L., Margrie, T. W., Portugues, R., and Branco, T. (2021). Visualizing anatomically registered data with brainrender. *eLife* 10, e65751. doi: 10.7554/eLife.65751

Cocchi, L., Gollo, L. L., Zalesky, A., and Breakspear, M. J. (2017). Criticality in the brain: a synthesis of neurobiology, models and cognition. *Progr. Neurobiol.* 158, 132–152. doi: 10.1016/j.pneurobio.2017.07.002

Courtiol, J., Guye, M., Bartolomei, F., Petkoski, S., and Jirsa, V. K. (2020). Dynamical mechanisms of interictal resting-state functional connectivity in epilepsy. *J. Neurosci.* 40, 5572–5588. doi: 10.1523/JNEUROSCI.0905-19.2020

Cowan, J. D., Neuman, J., and van Drongelen, W. (2016). Wilson–cowan equations for neocortical dynamics. *J. Math. Neurosci.* 6, 1. doi: 10.1186/s13408-015-0034-5

Deco, G., and Jirsa, V. K. (2012). Ongoing cortical activity at rest: criticality, multistability, and ghost attractors. *J. Neurosci.* 32, 3366–3375. doi: 10.1523/JNEUROSCI.2523-11.2012

Deco, G., Kringelbach, M. L., Arnatkeviciute, A., Oldham, S., Sabaroedin, K., Rogasch, N. C., et al. (2021). Dynamical consequences of regional heterogeneity in the brain's transcriptional landscape. *Sci. Adv.* 7, eabf4752. doi: 10.1126/sciadv.abf4752

Deco, G., Ponce-Alvarez, A., Mantini, D., Romani, G. L., Hagmann, P., and Corbetta, M. (2013). Resting-state functional connectivity emerges from structurally and dynamically shaped slow linear fluctuations. *J. Neurosci.* 33, 11239–11252. doi: 10.1523/JNEUROSCI.1091-13.2013

Deco, G., Robinson, P. A., Jirsa, V. K., Breakspear, M. J., and Friston, K. J. (2008). The dynamic brain: from spiking neurons to neural masses and cortical fields. *PLoS Comput. Biol.* 4, e1000092. doi: 10.1371/journal.pcbi.1000092

Demirtas, M., Burt, J. B., Helmer, M., Ji, J. L., Adkinson, B. D., Glasser, M. F., Van Essen, D. C., et al. (2019). Hierarchical heterogeneity across human cortex shapes large-scale neural dynamics. *Neuron* 101, 1181–1194.e13. doi: 10.1016/j.neuron.2019.01.017

Erö, C., Gewaltig, M.-O., Keller, D., and Markram, H. (2018). A cell atlas for the mouse brain. *Front. Neuroinf.* 12, e17727. doi: 10.3389/fninf.2018.00084

Friston, K. J., Mechelli, A., Turner, R., and Price, C. J. (2000). Nonlinear responses in fMRI: the balloon model, volterra kernels, and other hemodynamics. *NeuroImage* 42, 649–662. doi: 10.1016/j.neuroimage.2008.04.262

Froudist-Walsh, S., Bliss, D. P., Ding, X., Rapan, L., Niu, M., Knoblauch, K., et al. (2021). A dopamine gradient controls access to distributed working memory in the large-scale monkey cortex. *Neuron* 109, 3500–3520.e13. doi: 10.1016/j.neuron.2021.08.024

Fulcher, B. D., and Fornito, A. (2016). A transcriptional signature of hub connectivity in the mouse connectome. *Proc. Natl. Acad. Sci. U.S.A.* 113, 1435–1440. doi: 10.1073/pnas.1513302113

Fulcher, B. D., Murray, J. D., Zerbi, V., and Wang, X.-J. (2019). Multimodal gradients across mouse cortex. *Proc. Natl. Acad. Sci. U.S.A.* 116, 4689–4695. doi: 10.1073/pnas.1814144116

Goulas, A., Uylings, H. B. M., and Hilgetag, C. C. (2016). Principles of ipsilateral and contralateral cortico-cortical connectivity in the mouse. *Brain Struct. Funct.* 252, 1–15. doi: 10.1007/s00429-016-1277-y

Grandjean, J., Canella, C., Anckaerts, C., Ayrancı, G., Bougacha, S., Bienert, T., et al. (2020). Common functional networks in the mouse brain revealed by multi-centre resting-state fMRI analysis. *NeuroImage* 205, 116278. doi: 10.1016/j.neuroimage.2019.116278

Grandjean, J., Zerbi, V., Balsters, J., Wenderoth, N., and Rudina, M. (2017). The structural basis of large-scale functional connectivity in the mouse. *J. Neurosci.* 37, 0438–17–8101. doi: 10.1523/JNEUROSCI.0438-17.2017

Harris, J. A., Mihalas, S., Hirokawa, K. E., Whitesell, J. D., Choi, H., Bernard, A., et al. (2019). Hierarchical organization of cortical and thalamic connectivity. *Nature* 575, 195–202. doi: 10.1038/s41586-019-1716-z

Heitmann, S., Aburn, M. J., and Breakspear, M. (2018). The brain dynamics toolbox for matlab. *Neurocomputing* 315, 82–88. doi: 10.1016/j.neucom.2018.06.026

Jirsa, V. K., Proix, T., Perdikis, D., Woodman, M. M., Wang, H., Gonzalez-Martinez, J., et al. (2017). The virtual epileptic patient: individualized whole-brain models of epilepsy spread. *NeuroImage* 145, 377–388. doi: 10.1016/j.neuroimage.2016.04.049

Kim, J. H., and Ress, D. (2016). Arterial impulse model for the BOLD response to brief neural activation. *NeuroImage* 124, 394–408. doi: 10.1016/j.neuroimage.2015.08.068

Kim, Y., Yang, G. R., Pradhan, K., Venkataraju, K. U., Bota, M., García del Molino, L. C., et al. (2017). Brain-wide maps reveal stereotyped cell-type-based cortical architecture and subcortical sexual dimorphism. *Cell* 171, 456–469.e22. doi: 10.1016/j.cell.2017.09.020

Kong, X., Kong, R., Orban, C., Wang, P., Zhang, S., Anderson, K., et al. (2021). Sensory-motor cortices shape functional connectivity dynamics in the human brain. *Nat. Commun.* 12, 6373. doi: 10.1038/s41467-021-26704-y

Lein, E., Hawrylycz, M. J., Ao, N., Ayres, M., Bensinger, A., Bernard, A., et al. (2007). Genome-wide atlas of gene expression in the adult mouse brain. *Nature* 445, 168–176. doi: 10.1038/nature05453

Lin, I.-C., Okun, M., Carandini, M., and Harris, K. D. (2020). Equations governing dynamics of excitation and inhibition in the mouse corticothalamic network. *bioRxiv Preprint. 2020.06.03.132688*. doi: 10.1101/2020.06.03.132688

Margulies, D. S., Ghosh, S. S., Goulas, A., Falkiewicz, M., Huntenburg, J. M., Langs, G., et al. (2016). Situating the default-mode network along a principal gradient of macroscale cortical organization. *Proc. Natl. Acad. Sci. U.S.A.* 113, 12574–12579. doi: 10.1073/pnas.1608282113

Markicevic, M., Fulcher, B. D., Lewis, C., Helmchen, F., Rudin, M., Zerbi, V., et al. (2020). Cortical excitation:inhibition imbalance causes abnormal brain network dynamics as observed in neurodevelopmental disorders. *Cereb. Cortex* 30, 4922–4937. doi: 10.1093/cercor/bhaa084

Markicevic, M., Sturman, O., Bohacek, J., Rudin, M., Zerbi, V., Fulcher, B. D., et al. (2022). Neuromodulation of striatal D1 cells shapes BOLD fluctuations in anatomically connected thalamic and cortical regions. *bioRxiv Prepint 2022.03.11.483972*. doi: 10.1101/2022.03.11.483972

Mejias, J. F., Murray, J. D., Kennedy, H., and Wang, X.-J. (2016). Feedforward and feedback frequency-dependent interactions in a large-scale laminar network of the primate cortex. *Sci. Adv.* 2, e1601335. doi: 10.1126/sciadv.1601335

Mejías, J. F., and Wang, X.-J. (2022). Mechanisms of distributed working memory in a large-scale network of macaque neocortex. *eLife* 11, e72136. doi: 10.7554/eLife.72136

Melozzi, F., Bergmann, E., Harris, J. A., Kahn, I., Jirsa, V., and Bernard, C. (2019). Individual structural features constrain the mouse functional connectome. *Proc. Natl. Acad. Sci. U.S.A.* 116, 26961–26969. doi: 10.1073/pnas.1906694116

Melozzi, F., Woodman, M. M., Jirsa, V. K., and Bernard, C. (2017). The virtual mouse brain: a computational neuroinformatics platform to study whole mouse brain dynamics. *eNeuro* 4, ENEURO.0111–17.2017. doi: 10.1523/ENEURO.0111-17.2017

Messé, A., Rudrauf, D., Benali, H., and Marrelec, G. (2014). Relating structure and function in the human brain: relative contributions of anatomy, stationary dynamics, and non-stationarities. *PLoS Comput. Biol.* 10, e1003530. doi: 10.1371/journal.pcbi.1003530

Messé, A., Rudrauf, D., Giron, A., and Marrelec, G. (2015). Predicting functional connectivity from structural connectivity via computational models using MRI: an extensive comparison study. *NeuroImage* 111, 65–75. doi: 10.1016/j.neuroimage.2015.02.001

Müller, E. J., Munn, B. R., and Shine, J. M. (2020). Diffuse neural coupling mediates complex network dynamics through the formation of quasi-critical brain states. *Nat. Commun.* 11, 6337. doi: 10.1038/s41467-020-19716-7

Murray, J. D., Jaramillo, J., and Wang, X.-J. (2017). Working memory and decision-making in a frontoparietal circuit model. *J. Neurosci.* 37, 12167–12186. doi: 10.1523/JNEUROSCI.0343-17.2017

Noori, R., Park, D., Griffiths, J. D., Bells, S., Frankland, P. W., Mabbott, D., et al. (2020). Activity-dependent myelination: a glial mechanism of oscillatory self-organization in large-scale brain networks. *Proc. Natl. Acad. Sci. U.S.A.* 117, 13227–13237. doi: 10.1073/pnas.1916646117

Nozari, E., Bertolero, M. A., Stiso, J., Caciagli, L., Cornblath, E. J., He, X., et al. (2021). Is the brain macroscopically linear? A system identification of resting state dynamics. *arXiv [Preprint]. arXiv:2012.12351*. doi: 10.48550/arXiv.2012.12351

Nunes, R. V., Reyes, M. B., Mejias, J. F., and de Camargo, R. Y. (2021). Directed functional and structural connectivity in a large-scale model for the mouse cortex. *Netw. Neurosci.* 5, 874–889. doi: 10.1162/netna00206

Oh, S. W., Harris, J. A., Ng, L., Winslow, B., Cain, N., Mihalas, S., et al. (2014). A mesoscale connectome of the mouse brain. *Nature* 508, 207–214. doi: 10.1038/nature13186

Petkoski, S., and Jirsa, V. K. (2019). Transmission time delays organize the brain network synchronization. *Philosoph. Trans. R. Soc. Math. Phys. Eng. Sci.* 377, 20180132. doi: 10.1098/rsta.2018.0132

Robinson, P. A. (2012). Interrelating anatomical, effective, and functional brain connectivity using propagators and neural field theory. *Phys. Rev. E* 85, 011912. doi: 10.1103/PhysRevE.85.011912

Robinson, P. A. (2019). Physical brain connectomics. *Phys. Rev. E* 99, 012421. doi: 10.1103/PhysRevE.99.012421

Robinson, P. A., Postnova, S., Abeysuriya, R. G., Kim, J. W., Roberts, J. A., McKenzie-Sell, L., et al. (2015). "A multiscale working brain model," in *Validating Neuro-Computational Models of Neurological and Psychiatric Disorders*. (Cham: Springer), 107–140.

Robinson, P. A., Rennie, C. J., Rowe, D. L., O'Connor, S. C., Wright, J. J., Gordon, E., et al. (2003). Neurophysical modeling of brain dynamics. *Neuropsychopharmacology* 28, S74–S79. doi: 10.1038/sj.npp.1300143

Robinson, P. A., Zhao, X., Aquino, K. M., Griffiths, J. D., Sarkar, S., and Mehta-Pandejee, G. (2016). Eigenmodes of brain activity: neural field theory predictions and comparison with experiment. *NeuroImage* 142, 79. doi: 10.1016/j.neuroimage.2016.04.050

Sanz-Leon, P., Knock, S. A., Spiegler, A., and Jirsa, V. K. (2015). Mathematical framework for large-scale brain network modeling in The Virtual Brain. *NeuroImage* 111, 385–430. doi: 10.1016/j.neuroimage.2015.01.002

Schneider, M., Broggini, A. C., Dann, B., Tzanou, A., Uran, C., Sheshadri, S., et al. (2021). A mechanism for inter-areal coherence through communication based on connectivity and oscillatory power. *Neuron* 109, 4050–4067.e12. doi: 10.1016/j.neuron.2021.09.037

Sethi, S. S., Zerbi, V., Wenderoth, N., Fornito, A., and Fulcher, B. D. (2017). Structural connectome topology relates to regional BOLD signal dynamics in the mouse brain. *Chaos Interdiscipl. J. Nonlin. Sci.* 27, 047405. doi: 10.1063/1.4979281

Shadi, K., Dyer, E., and Dovrolis, C. (2020). Multisensory integration in the mouse cortical connectome using a network diffusion model. *Netw. Neurosci.* 4, 1030–1054. doi: 10.1162/netna00164

Shafiei, G., Markello, R. D., Vos de Wael, R., Bernhardt, B. C., Fulcher, B. D., and Misic, B. (2020). Topographic gradients of intrinsic dynamics across neocortex. *eLife* 9, e62116. doi: 10.7554/eLife.62116

Sip, V., Guye, M., Bartolomei, F., and Jirsa, V. (2022). Computational modeling of seizure spread on a cortical surface. *J. Comput. Neurosci.* 50, 17–31. doi: 10.1007/s10827-021-00802-8

Strogatz, S. H. (2018). *Nonlinear Dynamics and Chaos with Student Solutions Manual: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press.

Wagstyl, K., Ronan, L., Goodyer, I. M., and Fletcher, P. C. (2015). Cortical thickness gradients in structural hierarchies. *NeuroImage* 111, 241–250. doi: 10.1016/j.neuroimage.2015.02.036

Wang, P., Kong, R., Kong, X., Liégeois, R., Orban, C., Deco, G., et al. (2019). Inversion of a large-scale circuit model reveals a cortical hierarchy in the dynamic resting human brain. *Sci. Adv.* 5, eaat7854. doi: 10.1126/sciadv.aat7854

Wang, X.-J. (2020). Macroscopic gradients of synaptic excitation and inhibition in the neocortex. *Nat. Rev. Neurosci.* 21, 169–178. doi: 10.1038/s41583-020-0262-x

Wilson, H. R., and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophys. J.* 12, 1–24.

Wilson, H. R., and Cowan, J. D. (1973). A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik* 13, 55–80.

Wischnewski, K. J., Eickhoff, S. B., Jirsa, V. K., and Popovych, O. V. (2021). Towards an efficient validation of dynamical whole-brain models. *Sci Rep.* 12, 4331. doi: 10.1038/s41598-022-07860-7

Wong, K.-F., and Wang, X.-J. (2006). A recurrent network mechanism of time integration in perceptual decisions. *J. Neurosci.* 26, 1314–1328. doi: 10.1523/JNEUROSCI.3733-05.2006

Yao, Z., van Velthoven, C. T. J., Nguyen, T. N., Goldy, J., Sedeno-Cortes, A. E., Baftizadeh, F., et al. (2021). A taxonomy of transcriptomic cell types across the isocortex and hippocampal formation. *Cell* 184, 3222–3241.e26. doi: 10.1016/j.cell.2021.04.021

Zerbi, V., Floriou-Servou, A., Markicevic, M., Vermeiren, Y., Sturman, O., Privitera, M., et al. (2019). Rapid reconfiguration of the functional connectome after chemogenetic locus coeruleus activation. *Neuron* 103, 702–718.e5. doi: 10.1016/j.neuron.2019.05.034

Zerbi, V., Grandjean, J., Rudin, M., and Wenderoth, N. (2015). Mapping the mouse brain with rs-fMRI: an optimized pipeline for functional network identification. *NeuroImage* 123, 11–21. doi: 10.1016/j.neuroimage.2015.07.090

Zerbi, V., Pagani, M., Markicevic, M., Matteoli, M., Pozzi, D., Fagiolini, M., et al. (2021). Brain mapping across 16 autism mouse models reveals a spectrum of functional connectivity subtypes. *Mol. Psychiatry* 26, 1–11. doi: 10.1038/s41380-021-01245-4

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroinformatics

ORIGINAL RESEARCH
published: 16 May 2022
doi: 10.3389/fninf.2022.752471

The Case for Optimized Edge-Centric Tractography at Scale

Joseph Y. Moon[1], Pratik Mukherjee[2]*, Ravi K. Madduri[3], Amy J. Markowitz[2], Lanya T. Cai[2], Eva M. Palacios[2], Geoffrey T. Manley[2] and Peer-Timo Bremer[1]*

[1] Lawrence Livermore National Laboratory, Livermore, CA, United States, [2] Department of Radiology and Biomedical Imaging, University of California, San Francisco, San Francisco, CA, United States, [3] Argonne National Laboratory, Lemont, IL, United States

The anatomic validity of structural connectomes remains a significant uncertainty in neuroimaging. Edge-centric tractography reconstructs streamlines in bundles between each pair of cortical or subcortical regions. Although edge bundles provides a stronger anatomic embedding than traditional connectomes, calculating them for each region-pair requires exponentially greater computation. We observe that major speedup can be achieved by reducing the number of streamlines used by probabilistic tractography algorithms. To ensure this does not degrade connectome quality, we calculate the identifiability of edge-centric connectomes between test and re-test sessions as a proxy for information content. We find that running PROBTRACKX2 with as few as 1 streamline per voxel per region-pair has no significant impact on identifiability. Variation in identifiability caused by streamline count is overshadowed by variation due to subject demographics. This finding even holds true in an entirely different tractography algorithm using MRTrix. Incidentally, we observe that Jaccard similarity is more effective than Pearson correlation in calculating identifiability for our subject population.

Keywords: connectomes, identifiability, tractography, diffusion MRI, optimization, EDI, edge-centric

OPEN ACCESS

Edited by:
John David Griffiths, University of Toronto, Canada

Reviewed by:
Eric K. Neumann, Independent Researcher, Cambridge, United States
Gabriel Girard, Center for Biomedical Imaging (CIBM), Switzerland
Javier Guaje, Indiana University, United States

****Correspondence:***
Joseph Y. Moon
moon15@llnl.gov
Pratik Mukherjee
pratik.mukherjee@ucsf.edu

Received: *03 August 2021*
Accepted: *22 April 2022*
Published: *16 May 2022*

Citation:
Moon JY, Mukherjee P, Madduri RK, Markowitz AJ, Cai LT, Palacios EM, Manley GT and Bremer P-T (2022) The Case for Optimized Edge-Centric Tractography at Scale. Front. Neuroinform. 16:752471. doi: 10.3389/fninf.2022.752471

1. INTRODUCTION

The structural connectome is a powerful framework for analyzing macro-scale circuity of the living human brain and associating this connectivity with behavioral traits and health outcomes. Streamlines (also called fiber tracks or samples) are computationally reconstructed from each seed voxel in the white-to-gray matter boundary and connect exactly two regions of the brain. Structural connectome analysis, or connectomics, may have the power to distinguish autism spectrum disorder, estimate patient age and gender, and even predict cognitive ability (Betzel et al., 2014; Ingalhalikar et al., 2014; Contreras et al., 2015; Roine et al., 2015). Furthermore, there is a significant expectation that connectomics will provide crucial insights into otherwise difficult-to-probe neurological conditions, such as traumatic brain injury (TBI) and other cognitive disorders.

However, the anatomic validity of connectomes based on diffusion MRI has been inconsistent (Maier-Hein et al., 2017; Jeurissen et al., 2019). Tractography algorithms based on local fiber orientation may reconstruct large numbers of erroneous streamlines without additional constraints from ground-truth observation. Furthermore, the reconstructed streamline density may differ greatly from actual streamline density at each voxel, even when adjusted with filtering techniques such as SIFT (Smith et al., 2014, 2015). Owen et al. (2015) propose edge density imaging (EDI), which maps the number of region-to-region edges that pass through every white matter voxel. EDI is generated by edge-centric tractography, which reconstructs streamlines as edge bundles between individual pairs of cortical and subcortical regions. Each edge bundle is confined

to its own anatomically-plausible volume, which helps to exclude invalid streamlines. This has the advantage of normalizing connections between regions and improving inter-subject reproducibility, particularly between regions with high edge density (Owen et al., 2016).

However, progress in EDI and edge-centric tractography has been hampered by the computational cost of generating an order of magnitude more streamlines than before. A traditional connectome will simply seed a specific quantity of streamlines per voxel in the white-to-gray matter boundary and determine in which region each streamline terminates. This can be accomplished with a few tens of millions of streamlines and may take at most a few hours on modern computers. Edge-centric tractography must be repeated for each region-pair, such that each voxel (in the white-to-gray matter boundary) will reconstruct streamlines for every single region-pair that its streamlines could possibly intersect. Even when excluding anatomically-implausible region-pairs, this process can easily require billions of streamlines and consume many nodes on the most advanced high performance computing (HPC) platforms. Creating and curating edge-centric connectomes for a few dozen patients, even at a research facility, may take weeks and requires dedicated personnel familiar with computational neuroscience. As a result, processing hundreds or thousands of patients for a large-scale study has been cost-prohibitive. Here, we exploit the Department of Energy's vast HPC capabilities to examine the probabilistic variation of edge-centric tractography and the predictability of its computations. This aspect of EDI has not been studied carefully because of the sheer scale of computational and human resources required to analyze a large number of connectomes.

In particular, we focus on the probabilistic algorithms underpinning edge-centric tractography, particularly PROBTRACKX2 (Behrens et al., 2003) and MRTrix (Tournier et al., 2019). Though our thesis only applies to probabilistic algorithms, we include the results of a deterministic algorithm from MRTrix to demonstrate the robustness of identifiability as a quality metric. To account for the potential of crossing tracks, imprecise white-to-gray matter boundary estimations, and uncertainty induced by the lack of spatial resolution in the MRI scans, the research standard has been to compute 1,000 streamlines per voxel per region-pair (Owen et al., 2015). The unofficial publication standard is as many as 5,000 streamlines. However, the advantage of 1,000 streamlines per voxel remains unclear and the results presented below suggest that there may be little practical benefit in computing more than 1 streamline per voxel per region-pair. This simple but significant change implies an immediate reduction in computational cost by up to three orders of magnitude without significant loss of information.

The tractability of structural connectomes to matrix analysis has resulted in a variety of proposed techniques to assess their reliability (Imms et al., 2019). But since most of these techniques target specific medical or anatomical conditions, it is difficult to use them as universal metrics. In this work, we utilize a more general notion of identifiability, introduced by Amico and Goñi (2018). Conceptually, identifiability measures how well one can identify the connectome of a specific patient among a cohort of participants given an independently computed connectome from a prior MRI scan. Identifiability provides a generic measure of the information content of structural connectomes that is independent of any particular health condition or metric. We use a multi-center cohort of participants admitted for orthopedic, i.e., non-head related, injuries in order to demonstrate that a large streamline count does not improve identifiability in a general population. More specifically, we find that connectomes computed using 1 streamline per voxel per region-pair are as descriptive as connectomes that were generated with significantly higher streamline counts. Furthermore, the random variance induced by the probabilistic tractography is often as big as any changes observed for higher streamline counts. These two facts combined imply that many standard analyses will perform just as well with connectomes generated from a small number of streamline count than what is currently considered the standard. Reducing streamline count drastically reduces the computational resources required, making edge-centric structural connectomes accessible to a much wider range of researchers and potentially paving the way for real-time connectome analysis in a clinical setting.

2. METHODS

The edge-centric tractography workflow consists of three major steps (Payabvash et al., 2019): (1) calculating the probability distributions of fibers within each voxel from the raw MRI data, (2) parcellating the brain into structurally relevant regions, and (3) estimating how strongly each pair of regions are connected. The main focus of this paper is to analyze heuristics for the connectivity between brain regions using different streamline counts and use that information to estimate the accuracy of different levels of optimization. These heuristics must, in essence, estimate the likelihood that reconstructed connectomes match the real-world connectome. Since computing this likelihood directly is challenging, the accepted approach is to use uniform random sampling. Specifically, we begin with a large number of streamlines at each seed voxel and subsequently approximate the likelihood values by dividing the number of successful streamlines by the total number of streamlines. The likelihood values are then normalized by the volume of the regions and inserted into the connectome. Each cell of this upper-triangular matrix represents the connectivity of a region-to-region pair.

When we increase the streamline count, this process will converge to the true connectome as defined by the given parcellation, local fiber directions, and tractography algorithm. As the fiber directions form a very high dimensional sampling space and a complex distribution, common wisdom would suggest that a very large number of streamlines are required for an accurate estimate. The exact origin of the accepted publication standard of streamlines, between 1,000 and 5,000 streamlines per voxel. remains unclear. But these numbers are likely the result of similar concerns regarding accuracy. However, while more streamlines undoubtedly add more information to the connectome, doing so repeatedly for every single region-pair generates enormous amounts of redundant data. If we

use 82 cortical and subcortical regions in the commonly-used Desikan-Killiany parcellation, this results in 6642 potential region-pairs. Even when we curate the number of plausible region-pairs in the same way as Payabvash et al. (2019), we have nearly 1,000 region-pairs to consider for each seed voxel. Given between 10,000 and 100,000 seed voxels in the white-to-gray matter boundary (depending on subject anatomy, image resolution, and voxel density), this can result 10 to 100 billion streamline computations. We contend that this is far in excess of requirements for most use cases.

It is well-known that the physical aspects associated with an MRI procedure, i.e., measurement noise, patient motion, etc., as well as the constant change of the human brain add significant uncertainties to the measurements made on the brain which affect the generated connectome (Burgess et al., 2016). Therefore, it is unproductive to compute the connectome to a precision that is significantly higher than the maximal resolution implied by the inherent uncertainties. However, quantitatively assessing the "quality" of a connectome is not straight forward. There are two significant challenges. The first challenge is the requirement of a sufficient number of comparable MRI scans and the resources to compute their corresponding connectomes at different streamline counts. The second challenge is that there is no agreed-upon comparison metric between connectomes to understand the level of differences relevant in practice.

Here we address the first problem through a collaboration with the Transforming Research and Clinical Knowledge in Traumatic Brain Injury (TRACK-TBI) consortium.[1] TRACK-TBI is a longitudinal, observational study of TBI carried out at 18 Level 1 Trauma Centers across the United States. It includes brain-injured subjects along with a matched cohort of orthopedic injury control subjects. All participants were followed for 12 months following injury, and MRIs were collected from a subset of both the brain-injured and orthopedic injury cohorts. To avoid potential bias from the actual brain injuries, we are using a cohort of 88 orthopedic injury control subjects all between ages 18 and 71 (mean 37.8 yr; SD 13.7 yr; 30 female). All patients have no indication of head trauma based on clinical screening. We utilize diffusion-weighted MR imaging for each patient at two time points: 2 weeks and 6 months after injury. MR imaging is conducted with 3T scanners at 11 sites across the United States. All images are acquired using a uniform single-shell sampling scheme. All sites use the same acquisition parameters, insofar as possible across GE, Philips, and Siemens platforms (Palacios et al., 2017). Diffusion MRI and T1-weighted MRI pre-processing and post-processing are as reported in Owen et al. (2015, 2016). This process ultimately provides NIfTI diffusion tensor images with b = 1,000 s/mm^2, divided into 2.7-mm isotropic voxels in a 128 L $\times$128 W $\times$ 72 H matrix.

Given a total of 176 MRI scans we utilize MaPPeRTrac (Moon et al., 2020), a new portable and parallel computing pipeline that enables us to exploit large-scale computing facilities for the necessary tractography computations.[2] Our pipeline accomplishes the tractography workflow using the software components shown in **Table 1**.

TABLE 1 | Software components of MaPPeRTrac.

Pre-processing	**BET, DTIFIT, FLIRT (Jenkinson et al., 2002)**
Segmentation	Freesurfer (Desikan et al., 2006)
Fiber tensor estimation	BEDPOSTX2 (Behrens et al., 2003)
Probabilistic tractography	PROBTRACKX2 (Behrens et al., 2003)
Alternative prob. and deterministic tractography	MRTrix3 (Tournier et al., 2019)

Figure 1 gives a rough illustration of how we convert NIfTI images to connectomes matrices. When running Freesurfer, we parcellate the brain with the Desikan-Killiany atlas. For the PROBTRACKX2 pipeline, we use BEDPOSTX2 to estimate fiber orientation directions (FOD). We then run PROBTRACKX2 for each region-pair while adjusting streamline count between 1 and 1,000 streamlines per voxel and using the gray-white matter boundary as the seeding volume. All other software components are left to their default values. Our tractography workflow is portable across most scientific HPC clusters with Slurm, Cobalt, or Grid Engine job scheduling. However, to process these particular subjects, we used machines running the TOSS 3 operating system with Slurm scheduling. Further details can be found in **Table 2** (Moon et al., 2020).

Our software can also conduct tractography using the MRTrix library, as shown in **Figure 2**. It is important to note that we ran a traditional tractography algorithm using MRTrix. Since MRTrix lacks the ability to track the number of streamlines passing through each voxel, as opposed to just the start and end regions, it cannot be used to generate EDI. Our main intention with MRTrix is to show the generalizability of the claim that extremely high streamline counts fail to provide unique information, regardless of algorithm details and parameters. We conducted these experiments with the same of number of streamlines as edge-centric tractography to demonstrate this point.

Our MRTrix pipeline uses the same pre-processing tools and Freesurfer parcellation as the PROBTRACKX2 pipeline (Tournier et al., 2007). However, we convert the parcellation to five-tissue-type (5TT) format in order to use the Anatomically-Constrained Tractography (ACT) framework (Smith et al., 2012). This framework will more accurately terminate streamlines. We then estimate the response function for each white-matter voxel using the (Tournier et al., 2013) iterative algorithm, since this is the recommended approach for single-shell data. Having specified a mask using the whole diffusion-weighted image, we run the spherical deconvolution algorithm proposed by Tournier et al. (2007) on the response function estimation to generate the FOD. After normalizing the FOD to correct for intensity outliers, we use this FOD as the input for either the iFOD2 algorithm for probabilistic tractography or the SD_STREAM algorithm for deterministic tractography. The iFOD2 algorithm conducts second-order integration of estimated fiber orientations to determine principle streamline direction (Tournier et al., 2010).

[1]https://tracktbi.ucsf.edu
[2]https://github.com/LLNL/MaPPeRTrac

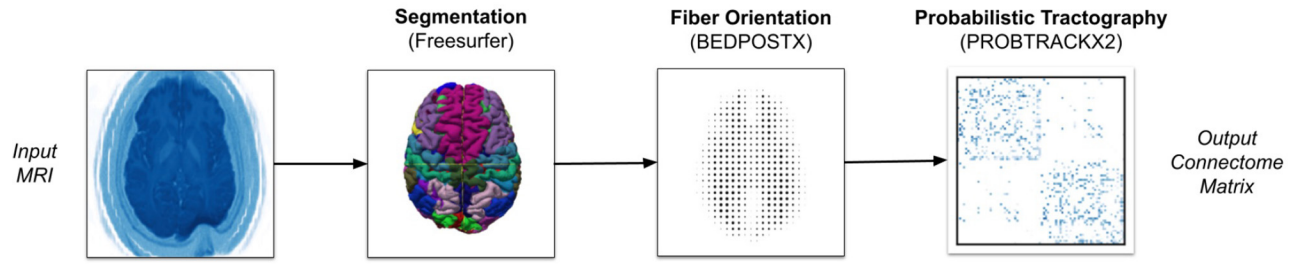


FIGURE 1 | Overview of MaPPeRTrac pipeline.

TABLE 2 | Hardware used to run MaPPeRTrac.

System	CPU	Clock speed	Cores/ node	RAM/ node	GPU
Quartz	Intel Xeon E5-2695	2.10–3.30 GHz	36	128 GB	n/a
Pascal	Intel Xeon E5-2695	2.10–3.30 GHz	36	256 GB	NVIDIA Tesla P100

The SD_STREAM algorithm performs Newton optimization to orient streamlines toward local peaks in the fiber orientation (Tournier et al., 2012). Like with PROBTRACKX2, we seed streamlines at the center of each voxel in the gray-white matter boundary and adjust streamline count between 1 and 1,000 streamlines per voxel. But whereas our PROBTRACKX2 pipeline seeded only the starting region in each region-pair, our MRTrix pipeline must combine all gray-white matter boundary volumes to create a single seeding volume. Since masking was performed during spherical deconvolution on our FOD, we do not apply another mask during tractography. Unless previously indicated, all MRTrix parameters are left to their default values. The hardware and subject data are identical to those used with PROBTRACKX2.

Our goal is to optimize tractography such that computation is minimized without losing any information content. Information content in this context refers to any biomarkers extrapolated from the connectome which may relate to various psychiatric disorders. These biomarkers are essentially patterns in the connectome matrix which are valuable insofar as they can be associated with patient outcomes, such as depressive disorder or Alzheimer's disease. However, despite significant advances, most studies of structural connectomes in a clinical context remain limited to a small number of patients. As a result, it is difficult to point to any set of best practices for tractography optimization in studies with dozens or hundreds of patients.

As previously mentioned, we use the notion of identifiability introduced by Amico and Goñi (2018). Whereas they measured identifiability in functional connectomes, we extend the concept to structural connectomes in order to estimate the information content across different streamline counts. Identifiability assumes that the connectome must capture unique characteristics of the individual, or at least distinct enough to make accurate medical and/or scientific predictions. Given the evidence for this assertion (Finn et al., 2015), we should be able to identify individual patients within a cohort of similar patients as long as each patient's unique characteristics are borne out in their connectome. Identifiability formalizes this concept and provides a quantitative measure of how well we can identify connectomes.

$$A_{ij} = corr(p_i, q_j) \tag{1}$$

$$I_{self} = \frac{1}{N}\sum A_{ii} \quad \text{and} \quad I_{others} = \frac{1}{N^2 - N}\sum^{i \neq j} A_{ij} \tag{2}$$

$$I_{diff} = (I_{self} - I_{others}) * 100 \tag{3}$$

The identifiability score for each patient is computed by comparing their connectome at one timepoint p to every connectome generated at different timepoints, q. As discussed in more detail below we have experimented with various forms of connectome metrics such as correlation, L2 distance, and Jaccard similarity. Equation (1) shows that this results in an $N \times N$ matrix $\mathbf{A}$, composed of correlations between the two timepoints where N is the number of patients. The average of diagonal elements, I_{self}, measures correlation between connectomes of the same patient. The average of off-diagonals, I_{others}, measures correlation between connectomes of different patients. These can be expressed as in Equation (2). Identifiability I_{diff}, as seen in Equation (3), is measured as the difference between I_{self} and I_{others}.

Amico and Goñi (2018) improve identifiability by reducing connectome dimensionality. If we perform principal component analysis (PCA) reconstruction with m components, then the best possible identifiability we can extract from the data is

$$I_{diff}* = arg\max_{m \epsilon M} I_{diff}(m) \tag{4}$$

We express identifiability as Equation (4) in all subsequent sections, as it represents the strongest identification ability for any set of connectomes.

Identifiability can be used to compare the success of different procedures at preserving the connectomes' information content. However, larger study populations will necessarily have lower identifiability, since each patient must self-identify out of a

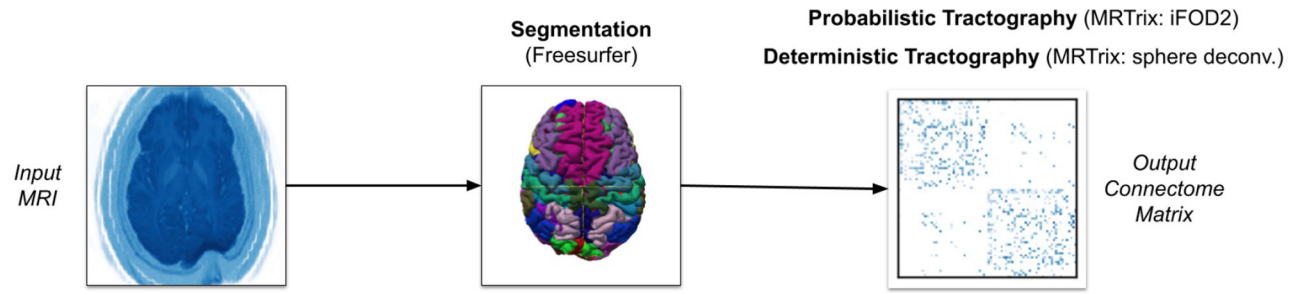


FIGURE 2 | Overview of alternative pipeline with MRTrix.

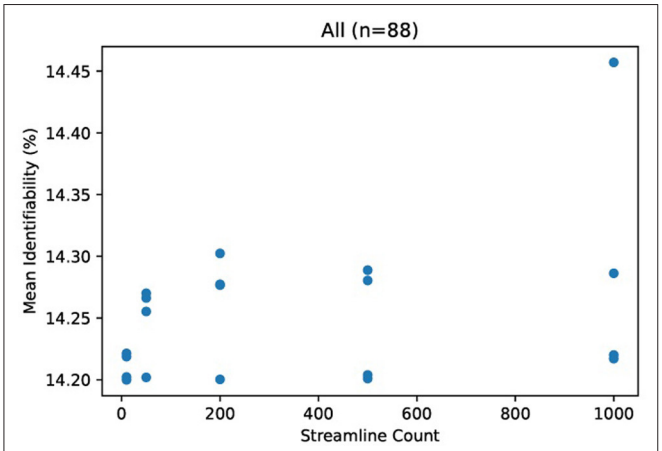


FIGURE 3 | Mean identifiability with all patients ($q = 4$ random seeds per streamline count, $k = 20$ subset size, $r = 10$ repetitions). For each streamline count, there are q data points with tractography running a unique random seed. Each data point represents the average of r repetitions of k randomly selected patients in order to normalize for dataset size.

wider pool of candidates. To mitigate this, we calculate the mean identifiability of repeated k-fold validation with fixed-size subsets. We randomly select a subset of k patients out of n total population, calculate identifiability of the subset, repeat this r times, and average the repetitions. The resulting mean identifiability enables comparison between differently-sized populations.

$$A_{ij} = \frac{|p_i - q_j|}{|p_i| + |q_j|} \tag{5}$$

$$A_{ij} = \frac{p_i}{|p_i|} \cdot \frac{q_j}{|q_j|} \tag{6}$$

$$A_{ij} = \frac{\sum_k min(p_{ik}, q_{jk})}{\sum_k max(p_{ik}, q_{jk})} \tag{7}$$

It is possible to calculate identifiability using correlation metrics other than Pearson correlation. The comparison between test and retest connectomes (see Equation 1) can be expressed using any linear correlation algorithm. For example, Equation (5) demonstrates a comparison using L2 distance, normalized against each connectome. We also examine the normalized dot product (Equation 6) and the Jaccard similarity coefficient (Equation 7). We experiment with multiple correlation metrics to help demonstrate the robustness of our optimization argument.

3. RESULTS

We re-ran probabilistic tractography with the same MRI scans for twenty iterations: at five streamline counts with four samples, each initialized with different random seeds. Note that we do not present median or standard deviation for these figures—this is because the cost of computation is so high that generating more than four samples per streamline count would be prohibitive. The five streamline counts are 10, 50, 200, 500, and 1,000 streamlines per voxel per region-pair. In the following figures, each data point represents the mean identifiability at a particular streamline count and random seed. Our tractography workflow re-calculates streamlines for every region pair, so each white matter voxel at the gray-white matter boundary will actually originate many more streamlines than this number suggests. We do not observe a relationship between mean identifiability and streamline count, especially considering stochastic variation and the narrow Y-axis. Since identifiability is the total percentage difference in correlation between I_{self} and I_{others} (see Equation 3), small stochastic variations of fractions of a percent have little impact. However, even stochastic variation appears to have a greater impact than streamline count. This suggests that connectomes generated with low streamline counts contain just as much information as high streamline counts, at least for identification tasks.

Due to the small number of data points (related to the extreme cost of compute), it would be unhelpful to evaluate correlation metrics between streamline count and identifiability such as coefficient of determination or error bars. We do not deny that correlation may exist between identifiability and streamline count. Because we argue that this correlation is not significant compared to variations due to demographics, we instead consider the absolute variations of identifiability within a category and between categories. In **Figure 3**, we see variation within all

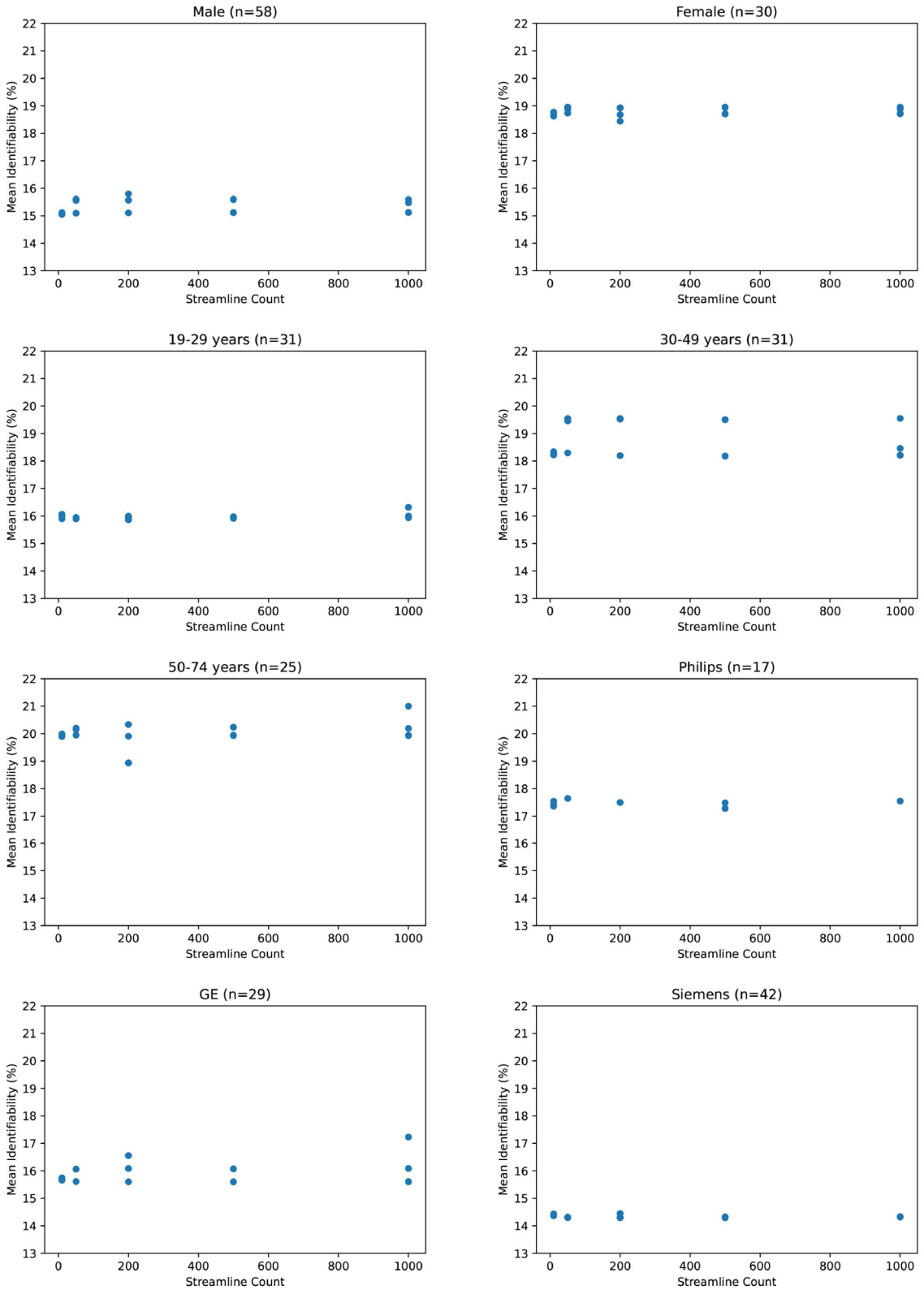


FIGURE 4 | Mean identifiability by category ($q = 4$ random seeds per streamline count, $k = 20$ subset size, $r = 10$ repetitions).

subjects of just 0.25 percent. In comparison, most categories in **Figure 4** differ from each other by much greater than 1 percent.

If we zoom in to individual categories, we see that mean identifiability does not strongly vary with streamline count no matter how patients are grouped together. Variation within each category is an average of 0.6 percent. The greatest variation is within 50–74 year olds at 2.1 percent, but this variation shows no positive relationship streamline count and identifiability. In addition, we observe that certain categories present stronger differences than others. Male and female identifiability differ by 3.9 percent, the youngest and oldest patients by 4.1 percent on average, and various MRI platforms by less than 1 percent. Though this does not confirm that identifiability is reading population differences between categories, it does suggest that those differences would be more significant than any increase in identifiability from a higher streamline count.

One could argue that by comparing connectomes only against other connectomes at the same streamline count, identifiability

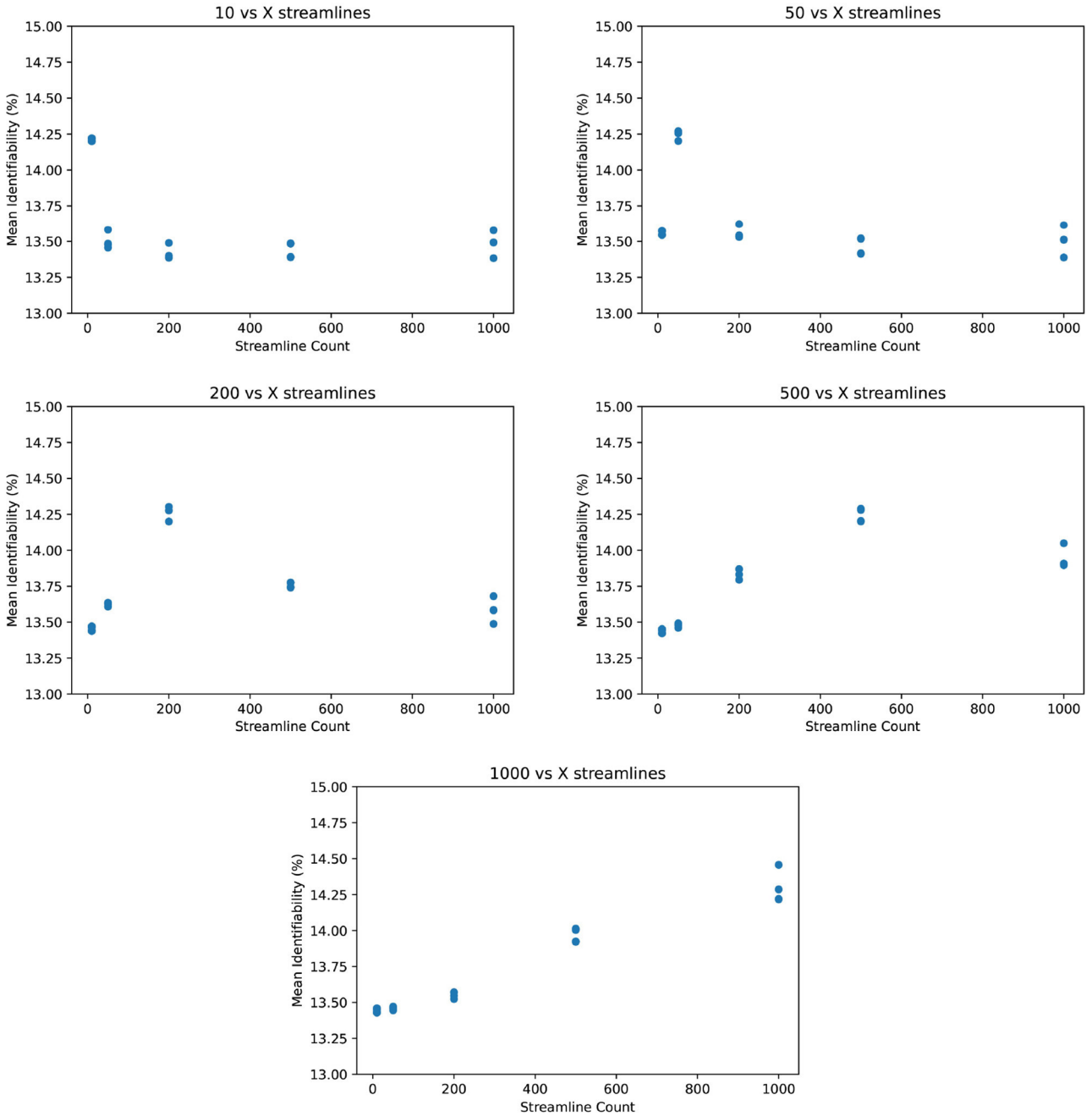


FIGURE 5 | Mean identifiability across streamline counts ($q = 4$ random seeds per streamline count, $k = 20$ subset size, $r = 10$ repetitions).

is biased by processing artifacts unique to that streamline count. Considering this, we compared identifiability with test connectomes p_i against retest connectomes q_j from different streamline counts. **Figure 5** appears to confirm this bias because identifiability is higher when the test and retest share the same streamline count. But to some degree, this is expected, as information particular to that streamline count is shared between its tests and retests, whereas those from different streamline counts may not carry that information. Nevertheless, the degree of bias does not seem to be significant compared to the overall success in identification. Again note the narrow Y-axis - even identifiability as low as 13% is more than sufficient to distinguish a retest from all 87 other retest connectomes.

We observe the same trend of weak correlation between streamline count and identifiability in **Figure 6**. Incidentally, we find that L2 distance yields somewhat better identification power than Pearson correlation. The normalized dot product appears relatively weak in comparison. However, the Jaccard similarity coefficient demonstrates significantly stronger identifiability than Pearson correlation. This is particularly unusual since Jaccard similarity discards much information from its inputs by only selecting the maximum and minimum of the test and retest values. Although we use Pearson correlation in all other figures due to its prevalence in existing literature, **Figure 6** suggests that there may be room for improving the identifiability algorithm.

For sake of completeness, we examine the edge-centric connectomes using alternative graph metrics common in neuroimaging literature. Details of these graph metrics for the purpose of investigating test-retest reliability have been described by Imms et al. (2019). For each connectome, we (1) calculate each graph metric at each streamline count, (2) normalize the graph metric at each streamline count against the value of the graph metric at 1,000 streamlines, and (3) plot each normalized graph metric in **Figure 7**. The resulting plots demonstrate no added value above 1 streamlines per voxel per region-pair, similar to our results for identifiability.

We ran the same subjects with MRTrix to generate traditional connectomes, again using five streamline counts with four samples each and k-fold validation. The results in **Figure 8** demonstrate the same trend—an extremely slight variation in identifiability with streamline count. In fact, the relationship between streamline count and identifiability appears so tenuous that higher counts have slightly lower identifiability. In **Figure 9**, it is unsurprising to see the deterministic algorithm sees no variation with streamline count at all. This indicates that the deterministic algorithm used by MRTrix is conducting needless

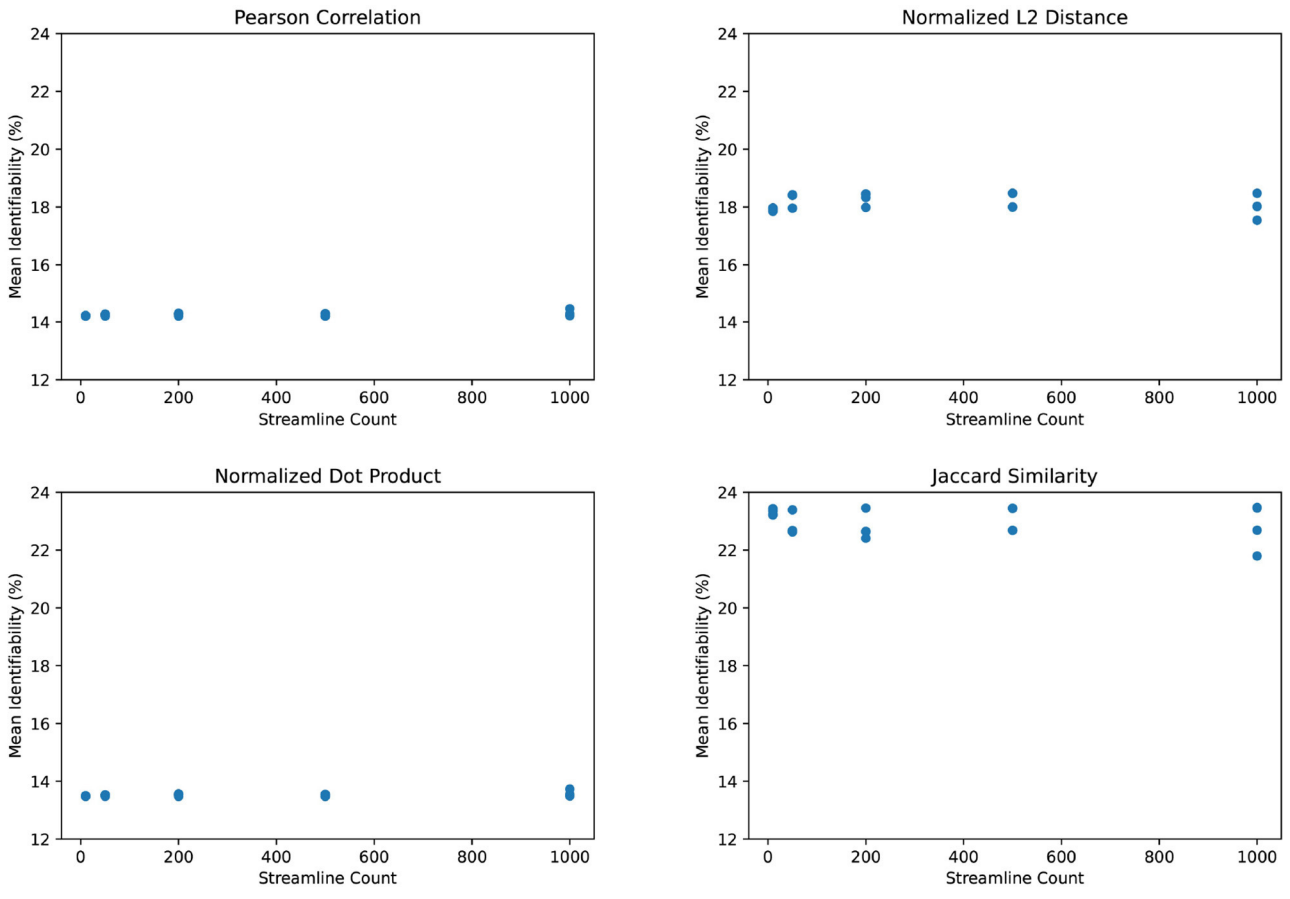


FIGURE 6 | Mean identifiability by correlation metric ($q = 4$ random seeds per streamline count, $k = 20$ subset size, $r = 10$ repetitions).

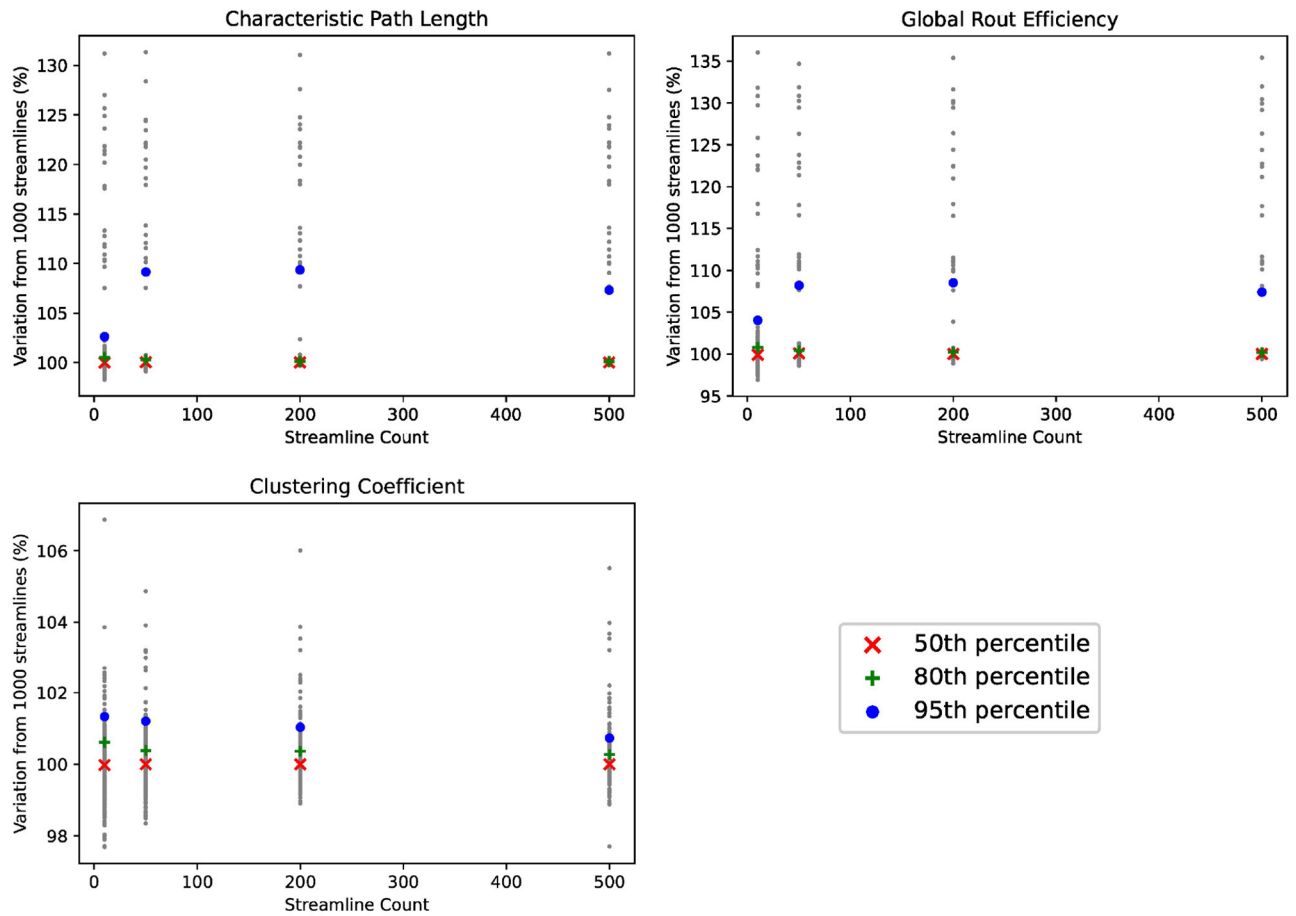


FIGURE 7 | Comparison of graph metrics.

computation beyond the first streamline per voxel, since there is no remaining decision space for tractography to explore.

4. DISCUSSION

By comparing edge-centric connectomes with the concept of identifiability, we find that probabilistic and deterministic algorithms do not significantly benefit from high streamline counts. This has major ramifications for the computational cost and availability of edge-centric tractography, as similar results can be achieved with a fraction of the streamlines. However, there is a major risk that optimization would lose information not captured by identifiability. The ability to identify a patient is necessary to connectome analysis—otherwise one could argue that a connectome is indistinguishable and therefore dominated by noise and external variables. But even if we could perfectly identify patients from connectomes, this may not be sufficient for more complex analyses.

There is also the risk that we did not compute sufficient samples. To address this, we re-ran probabilistic tractography on all patients with five streamline counts and four different random seeds, for a total of twenty iterations. With that amount of data, streamline count does not appear to significantly influence identifiability. Even if correlation can be established, the slope of such a curve is so flat as to be swamped by noise and subject demographics. However, it is remotely possible that running far more than twenty iterations would show strong variation. We do not pursue this possibility owing to the computational expense of tractography with high streamline counts - generating our data already consumed over 300,000 CPU hours.

We also find that Jaccard similarity outperforms more commonly used connectome correlation metrics such as Pearson correlation in the calculation of identifiability. Though we are surprised that this is the case, it is possible that Jaccard similarity increases the weight of low-frequency information by effectively binarizing the non-shared values. When calculating identifiability, high-frequency values, such as dense contiguous sections of the brain, may often match to the wrong subject. Subjects are better distinguished by low-frequency areas with unique structures. Given an incorrect match, choosing a minimum or maximum of the test and retest value in low-frequency areas will create a strongly fluctuating test-retest variation since values tend not to overlap. And whereas Pearson correlation and other metrics would dilute this variation by

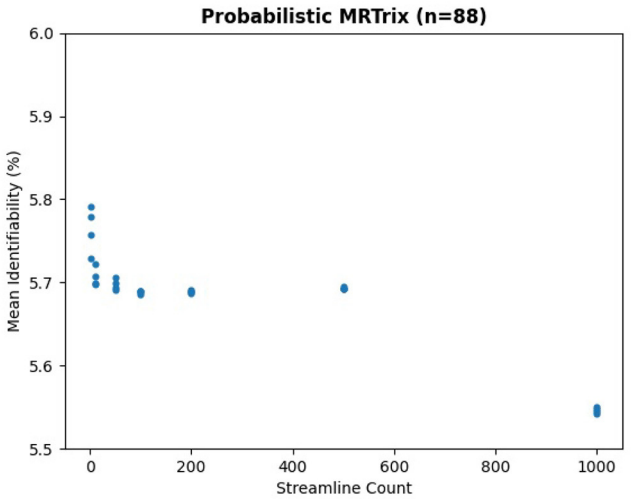


FIGURE 8 | Mean identifiability with all patients using probabilistic MRTrix (q = 4 random seeds per streamline count, k = 20 subset size, r = 10 repetitions, iFOD2 algorithm).

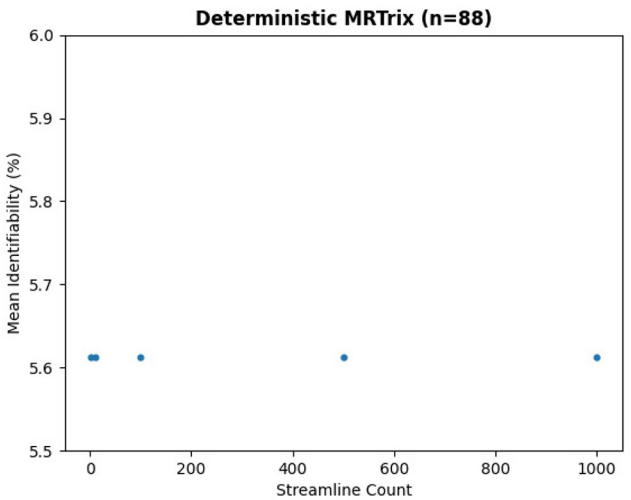


FIGURE 9 | Mean identifiability with all patients using deterministic MRTrix (SD_STREAM algorithm).

the weight of high-frequency areas, Jaccard similarity would provide consistent test-retest variation in high-frequency areas since it does not combine the test and retest in each voxel. As a result, Jaccard similarity improves identifiability similarly to PCA reconstruction, by pruning low-information data. However, this is mostly speculation and would require further study beyond the scope of this paper.

There is also the concern that our findings lack external physiological data. Brains do not exist in a vacuum, so key markers such as clinical survey results, blood pressure, and body weight may influence connectome analysis in subtle ways. We mitigate this to an extent by categorizing patients by age and gender and find that nothing in these categories undermines our argument regarding streamline count. Furthermore, it has been demonstrated that tractography is highly sensitive to choice of processing method. If the method itself diverges from ground truth, there is little that reproducibility can do to recover accurate results. Ideally, we would approach ground truths using phantom studies on the MRI processing techniques (Nath et al., 2020) or histological studies on *ex vivo* specimins (Schilling et al., 2018, 2019). However, we do not possess further anatomical or physiological data for this patient population, so the influence of other external variables remains unexplored.

We are also limited to using a particular set of acquisition and pre-processing parameters. Previous studies have used a broad array of parameters on the same subjects to make generalizable observations (Côté et al., 2013). Though our narrow parameters may appear to limit the generalizability of our findings, we contend that differences between scans of the patients are subtle enough that the ability of distinguish between them is more significant than the ability to compare alternative parameters on the same data. For example, a slightly different parcellation would result in changes to the overall structure of the connectome matrix, but identifiability would not greatly change since the relative differences between connectomes would be much less affected. Since we can even find the same results with two entirely different tractography softwares, PROBTRACKX2 and MRTrix, then minor changes on tractography parameters are unlikely to change our overall findings.

5. CONCLUSIONS

Progress in EDI connectomics has been limited by the steep computational cost of probabilistic white matter fiber tractography. Creating diverse datasets with large numbers of patients requires optimizations of the tractography workflow. However, excessive optimization may degrade the connectome's information content. To measure the extent to which we can optimize tractography, we use identifiability as an approximate measure of the average information content in a set of connectomes. Identifiability is a quantifiable metric for identification tasks predictiveness using a patient's test and retest, based on MRI conducted 6 months apart. This enables us to optimize computation by determining whether information is lost.

Edge-density probabilistic tractography is computationally expensive because it simulates massive quantities of white-matter fiber streamlines. We find that the number of streamlines can be greatly reduced from current practice. This optimization appears to have no impact on identifiability; ergo, it does not degrade the connectome's information content for most purposes. Reducing the number of streamlines yields direct linear efficiencies, such that using half the streamlines takes approximately half the time to compute. Existing literature uses between 1,000 and 5,000 streamlines per voxel per region-pair to ensure a well-converged solution. We find that identifiability is stable with as few as 1 streamlines per voxel per region-pair.

We find that low streamline counts perform just as well as high streamline counts even when analyzing our study population

with different demographics. These findings hold true for male and female patients, different age ranges, different correlation metrics, and all three common MRI hardware platforms. The choice of population makes a far greater impact than any decision on streamline count. In fact, variations in mean identifiability due to streamline count are even less than those from stochastic variation due to probabilistic tractography.

Using low streamline counts promises to greatly accelerate study of EDI and edge-centric connectomes. High streamline counts do not appear to harm identifiability in any scenario, and will likely continue to be the standard for small-scale studies. But by reducing the computational cost of tractography, this simple optimization will enable hundreds to thousands of edge-centric connectomes to be generated on systems that previously handled a few dozen. Many open neuroimaging questions related to EDI cannot be answered with small-scale studies alone, particularly those on subtle population differences such as behavioral disorders. As the field of connectomics grows, optimizations such as these will be necessary to keep up with the large amount of clinical data and computational resources applied to human brain research as well as foster clinical applications that require faster results for real-time patient care.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

ETHICS STATEMENT

Ethical review and approval was not required for the study on human participants in accordance with the local legislation and institutional requirements. The patients/participants provided their written informed consent to participate in this study.

AUTHOR CONTRIBUTIONS

Code, experimentation, and writing were primarily conducted by JM under the supervision of P-TB. RM and LC contributed significant technical and editorial collaboration, particularly with the Mappertrac software. The patient data was prepared by the UCSF authors, led by GM. PM and EP contributed most of the Section 2, with significant assistance from AM. All authors assisted with the creation of this paper. All authors contributed to the article and approved the submitted version.

FUNDING

The research was funded by the United States Department of Energy under the DOE Office of Science, Advanced Scientific Computing Research. Support was organized under The Co-Design for Artificial Intelligence and Computing at Scale for Extremely Large, Complex Datasets projects (Grant #KJ040301). This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

REFERENCES

Amico, E., and Goñi, J. (2018). The quest for identifiability in human functional connectomes. *Sci. Rep.* 8, 8254. doi: 10.1038/s41598-018-25089-1

Behrens, T., Woolrich, M., Jenkinson, M., Johansen-Berg, H., Nunes, R., Clare, S., et al. (2003). Characterization and propagation of uncertainty in diffusion-weighted mr imaging. *Mag. Reson. Med.* 50, 1077–1088. doi: 10.1002/mrm.10609

Betzel, R. F., Byrge, L., He, Y., Goñi, J., Zuo, X.-N., and Sporns, O. (2014). Changes in structural and functional connectivity among resting-state networks across the human lifespan. *Neuroimage* 102, 345–357. doi: 10.1016/j.neuroimage.2014.07.067

Burgess, G. C., Kandala, S., Nolan, D., Laumann, T. O., Power, J. D., Adeyemo, B., et al. (2016). Evaluation of denoising strategies to address motion-correlated artifacts in resting-state functional magnetic resonance imaging data from the human connectome project. *Brain Connect.* 6, 669–680. doi: 10.1089/brain.2016.0435

Contreras, J. A., Goñi, J., Risacher, S. L., Sporns, O., and Saykin, A. J. (2015). The structural and functional connectome and prediction of risk for cognitive impairment in older adults. *Curr. Behav. Neurosci. Rep.* 2, 234–245. doi: 10.1007/s40473-015-0056-z

Côté, M.-A., Girard, G., Bore, A., Garyfallidis, E., Houde, J.-C., and Descoteaux, M. (2013). Tractometer: Towards validation of tractography pipelines. *Med. Image Anal.* 17, 844–857. doi: 10.1016/j.media.2013.03.009

Desikan, R. S., Segonne, F., Fischl, B., Quinn, B. T., Dickerson, B. C., Blacker, D., et al. (2006). An automated labeling system for subdividing the human cerebral cortex on mri scans into gyral based regions of interest. *Neuroimage* 31, 968–980. doi: 10.1016/j.neuroimage.2006.01.021

Finn, E. S., Shen, X., Scheinost, D., Rosenberg, M. D., Huang, J., Chun, M. M., et al. (2015). Functional connectome fingerprinting: identifying individuals using patterns of brain connectivity. *Nat. Neurosci.* 18, 1664–1671. doi: 10.1038/nn.4135

Imms, P., Clemente, A., Cook, M., D'Souza, W., Wilson, P. H., Jones, D. K., et al. (2019). The structural connectome in traumatic brain injury: a meta-analysis of graph metrics. *Neurosci. Biobehav. Rev.* 99, 128–137. doi: 10.1016/j.neubiorev.2019.01.002

Ingalhalikar, M., Smith, A., Parker, D., Satterthwaite, T. D., Elliott, M. A., Ruparel, K., et al. (2014). Sex differences in the structural connectome of the human brain. *Proc. Natl. Acad. Sci. U.S.A.* 111, 823–828. doi: 10.1073/pnas.1316909110

Jenkinson, M., Bannister, P., Brady, M., and Smith, S. (2002). Improved optimization for the robust and accurate linear registration and motion correction of brain images. *Neuroimage* 17, 825–841. doi: 10.1006/nimg.2002.1132

Jeurissen, B., Descoteaux, M., Mori, S., and Leemans, A. (2019). Diffusion MRI fiber tractography of the brain. *NMR Biomed*. 32, e3785. doi: 10.1002/nbm.3785

Maier-Hein, K. H., Neher, P. F., Houde, J.-C., Côté, M.-A., Garyfallidis, E., Zhong, J., et al. (2017). The challenge of mapping the human connectome based on diffusion tractography. *Nat. Commun.* 8, 1349. doi: 10.1038/s41467-017-01285-x

Moon, J. Y., Bremer, P.-T., Mukherjee, P., Markowitz, A. J., Palacios, E. M., Rodriguez, A., et al. (2020). MaPPeRTrac: a massively parallel, portable, and reproducible tractography pipeline. *bioRxiv [Preprint]*. doi: 10.1101/2020.12.23.424191

Nath, V., Schilling, K. G., Parvathaneni, P., Huo, Y., Blaber, J. A., Hainline, A. E., et al. (2020). Tractography reproducibility challenge with empirical data (traced): the 2017 ISMRM diffusion study group challenge. *J. Mag. Reson. Imaging* 51, 234–249. doi: 10.1002/jmri.26794

Owen, J. P., Chang, Y. S., and Mukherjee, P. (2015). Edge density imaging: mapping the anatomic embedding of the structural connectome within the white matter of the human brain. *Neuroimage* 109, 402–417. doi: 10.1016/j.neuroimage.2015.01.007

Owen, J. P., Wang, M. B., and Mukherjee, P. (2016). Periventricular white matter is a nexus for network connectivity in the human brain. *Brain Connect.* 6, 548–557. doi: 10.1089/brain.2016.0431

Palacios, E., Martin, A., Boss, M., Ezekiel, F., Chang, Y., Yuh, E., et al. (2017). Toward precision and reproducibility of diffusion tensor imaging: a multicenter diffusion phantom and traveling volunteer study. *Am. J. Neuroradiol.* 38, 537–545. doi: 10.3174/ajnr.A5025

Payabvash, S., Palacios, E. M., Owen, J. P., Wang, M. B., Tavassoli, T., Gerdes, M., et al. (2019). White matter connectome edge density in children with autism spectrum disorders: potential imaging biomarkers using machine-learning models. *Brain Connect.* 9, 209–220. doi: 10.1089/brain.2018.0658

Roine, U., Roine, T., Salmi, J., Nieminen-von Wendt, T., Tani, P., Leppämäki, S., et al. (2015). Abnormal wiring of the connectome in adults with high-functioning autism spectrum disorder. *Mol. Autism* 6, 65. doi: 10.1186/s13229-015-0058-4

Schilling, K., Gao, Y., Stepniewska, I., Janve, V., Landman, B., and Anderson, A. (2018). Anatomical accuracy of standard-practice tractography algorithms in the motor system - a histological validation in the squirrel monkey brain. *Mag. Reson. Imaging* 55, 7–25. doi: 10.1016/j.mri.2018.09.004

Schilling, K. G., Nath, V., Hansen, C., Parvathaneni, P., Blaber, J., Gao, Y., et al. (2019). Limits to anatomical accuracy of diffusion tractography using modern approaches. *Neuroimage* 185, 1–11. doi: 10.1016/j.neuroimage.2018.10.029

Smith, R., Tournier, J.-D., Calamante, F., and Connelly, A. (2014). The effects of sift on the reproducibility and biological accuracy of the structural connectome. *Neuroimage* 104, 253–265. doi: 10.1016/j.neuroimage.2014.10.004

Smith, R. E., Tournier, J.-D., Calamante, F., and Connelly, A. (2012). Anatomically-constrained tractography: improved diffusion mri streamlines tractography through effective use of anatomical information. *Neuroimage* 62, 1924–1938. doi: 10.1016/j.neuroimage.2012.06.005

Smith, R. E., Tournier, J.-D., Calamante, F., and Connelly, A. (2015). Sift2: enabling dense quantitative assessment of brain white matter connectivity using streamlines tractography. *Neuroimage* 119, 338–351. doi: 10.1016/j.neuroimage.2015.06.092

Tournier, J.-D., Calamante, F., and Connelly, A. (2007). Robust determination of the fibre orientation distribution in diffusion mri: non-negativity constrained super-resolved spherical deconvolution. *Neuroimage* 35, 1459–1472. doi: 10.1016/j.neuroimage.2007.02.016

Tournier, J.-D., Calamante, F., and Connelly, A. (2012). Mrtrix: diffusion tractography in crossing fiber regions. *Int. J. Imaging Syst. Technol.* 22, 53–66. doi: 10.1002/ima.22005

Tournier, J.-D., Calamante, F., and Connelly, A. (2013). Determination of the appropriate b value and number of gradient directions for high-angular-resolution diffusion-weighted imaging. *NMR Biomed.* 26, 1775–1786. doi: 10.1002/nbm.3017

Tournier, J.-D., Smith, R., Raffelt, D., Tabbara, R., Dhollander, T., Pietsch, M., et al. (2019). MRTRIX3: a fast, flexible and open software framework for medical image processing and visualisation. *Neuroimage* 202, 116137. doi: 10.1016/j.neuroimage.2019.116137

Tournier, J. D., Calamante, F., Connelly, A., et al. (2010). "Improved probabilistic streamlines tractography by 2nd order integration over fibre orientation distributions," in *Proceedings of the International Society for Magnetic Resonance in Medicine* (Hoboken, NJ: John Wiley & Sons, Inc.)

Conflict of Interest: GM discloses grants from the United States Department of Defense—TBI Endpoints Development Initiative (Grant #W81XWH-14-2-0176), TRACK-TBI Precision Medicine (Grant #W81XWH-18-2-0042), and TRACK-TBI NETWORK (Grant #W81XWH-15-9-0001); NIH-NINDS—TRACK-TBI (Grant #U01NS086090); and the National Football League (NFL) Scientific Advisory Board—TRACK-TBI LONGITUDINAL. The United States Department of Energy supports GM for a precision medicine collaboration. One Mind has provided funding for TRACK-TBI patients stipends and support to clinical sites. He has received an unrestricted gift from the NFL to the UCSF Foundation to support research efforts of the TRACK-TBI NETWORK. He has also received funding from NeuroTruama Sciences LLC to support TRACK-TBI data curation efforts. Additionally, Abbott Laboratories has provided funding for add-in TRACK-TBI clinical studies. AM receives funding from the Department of Defense TBI Endpoints Development Initiative (Grant #W81XWH-14-2-0176) and TRACK-TBI NETWORK (Grant #W81XWH-15-9-0001). She also receives salary support from the United States Department of Energy precision medicine collaboration and the philanthropic organization, One Mind. JM and P-TB are employed by Lawrence Livermore National Security, LLC.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroinformatics

ORIGINAL RESEARCH
published: 20 May 2022
doi: 10.3389/fninf.2022.724336

EDEN: A High-Performance, General-Purpose, NeuroML-Based Neural Simulator

Sotirios Panagiotou[1,2]*, *Harry Sidiropoulos*[2], *Dimitrios Soudris*[1], *Mario Negrello*[2]* and *Christos Strydis*[2,3]*

[1] School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece, [2] Department of Neuroscience, Erasmus Medical Center, Rotterdam, Netherlands, [3] Quantum and Computer Engineering Department, Delft University of Technology, Delft, Netherlands

Modern neuroscience employs *in silico* experimentation on ever-increasing and more detailed neural networks. The high modeling detail goes hand in hand with the need for high model reproducibility, reusability and transparency. Besides, the size of the models and the long timescales under study mandate the use of a simulation system with high computational performance, so as to provide an acceptable time to result. In this work, we present EDEN (Extensible Dynamics Engine for Networks), a new general-purpose, NeuroML-based neural simulator that achieves both high model flexibility and high computational performance, through an innovative model-analysis and code-generation technique. The simulator runs NeuroML-v2 models directly, eliminating the need for users to learn yet another simulator-specific, model-specification language. EDEN's functional correctness and computational performance were assessed through NeuroML models available on the NeuroML-DB and Open Source Brain model repositories. In qualitative experiments, the results produced by EDEN were verified against the established NEURON simulator, for a wide range of models. At the same time, computational-performance benchmarks reveal that EDEN runs from one to nearly two orders-of-magnitude faster than NEURON on a typical desktop computer, and does so without additional effort from the user. Finally, and without added user effort, EDEN has been built from scratch to scale seamlessly over multiple CPUs and across computer clusters, when available.

Keywords: computational neuroscience, biological neural networks, simulation, High-Performance Computing, code morphing, interoperability, NeuroML, software

OPEN ACCESS

Edited by:
John David Griffiths,
University of Toronto, Canada

Reviewed by:
Boris Marin,
Federal University of ABC, Brazil
Richard C. Gerkin,
Arizona State University, United States
Shailesh Appukuttan,
UMR9197 Institut des Neurosciences Paris Saclay (Neuro-PSI), France

***Correspondence:**
Sotirios Panagiotou
s.panagiotou@erasmusmc.nl
Mario Negrello
m.negrello@erasmusmc.nl
Christos Strydis
c.strydis@erasmusmc.nl

Received: *12 June 2021*
Accepted: *24 March 2022*
Published: *20 May 2022*

Citation:
Panagiotou S, Sidiropoulos H, Soudris D, Negrello M and Strydis C (2022) EDEN: A High-Performance, General-Purpose, NeuroML-Based Neural Simulator. Front. Neuroinform. 16:724336. doi: 10.3389/fninf.2022.724336

1. INTRODUCTION

Simulation of biological neural networks is an essential tool of modern neuroscience. However, there are currently certain challenges associated with the development and *in silico* study of such networks. The neural models in use are diverse and heterogeneous; there is no single set of mathematical formulae that is commonly used by the majority of existing models. In addition, the biophysical mechanisms that make up models are constantly being modified, and reused in various combinations in new models. These factors mandate the use of general-purpose neural simulators in common practice. At the same time, the network sizes and levels of modeling detail employed in modern neuroscience translate to a constant increase in the volume of required computations.

Thus, neuroscience projects necessitate high-performance tools for simulations to finish in a practical amount of time and for models to fit into available computer memory.

Although there already exists a rich arsenal of simulators targeting neuroscience, the aforementioned challenges of neural simulation remain an open problem. On one hand, there are hand-written codes that push the processing hardware to the limit but they are difficult or impossible to extend in terms of model support, because of their over-specialization. They offer great computational performance by executing solely the numerical calculations required by the model's dynamics. On the other hand, there are general-purpose simulators that readily support most types of models, however, their computational efficiency is much less than that of hand-written codes. Hence, there is a significant gap in efficiency between general-purpose neural simulators and the computational capabilities that modern hardware platforms can achieve.

Besides, simulation of large networks often requires deploying neural models on multiple processor cores or, even, on computer clusters. Existing general-purpose simulators do not manage the technicalities of parallelization, model decomposition, and communication automatically. Thus, significant engineering effort is spent on setting up the simulators to run on multi-core and multi-node systems, which further obstructs scientific work.

A further problem is that, presently, each neural simulator uses its own model-specification language. Thus, models written for one simulator are difficult and laborious to adapt for another, which hampers the exchange and reuse of models across the neuroscience community. In this context, if a new simulator were to support only its own modeling language, this would fragment the modeling community further and would add a serious barrier to the simulator's adoption as well as the reuse of existing models.

1.1. The EDEN Simulator

To address the challenges in *in silico* neuroscience, we designed a new general-purpose neural simulator, called EDEN (Extensible Dynamics Engine for Networks). EDEN directly runs models described in NeuroML, achieves leading computational performance through a novel architecture, and handles parallel-processing resources—both on standalone personal computers as well as on computer clusters—automatically.

EDEN employs an innovative *model-analysis* and *code-generation technique*[1] through which the model's variables and the mathematical operations needed for the simulation are converted into a set of individual *work items*. Each work item consists of the data that represent a part of the neural network, and the calculations to simulate this part of the network over time. The calculations for the individual work items can then be run *in parallel* within each simulation step, allowing distribution of the computational load among many processing elements. This technique enables *by-design support* for general neural models, and at the same time offers significant performance benefits over conventional approaches. The need for model generality with user-provided formulae is *directly* addressed *via automatic code generation*; but the architecture also supports hand-optimized implementations that apply for specific types of neurons. At the same time, reducing the complex structure of biophysical mechanisms inside a neuron into an explicitly laid out set of essential, model-specific calculations allows compilers to perform large-scale optimizations. What is more, traditional simulators perform best with specific kinds of neuron models (e.g., multi-compartmental or point neurons) and worse with other ones. In contrast, EDEN's approach allows selecting the implementation that works best for each part of the network, at run time.

We adopted the *NeuroML v2* standard (Cannon et al., 2014) as our simulator's modeling language. NeuroML v2 is the emerging, standard cross-tool specification language for general neural-network models. By following the standard, we stay compatible with the entire NeuroML-software ecosystem: EDEN's simulation functionality is complemented by all the existing model-generation and results-analysis tools, and the ecosystem gets the most value out of EDEN as an interoperable simulator. Furthermore, positioning the simulator as a plug-compatible tool in the NeuroML stack allows us to focus our efforts on EDEN's features as a simulator (namely, computational performance, model generality, and usability). Finally, supporting an established modeling language makes user adoption much easier, compared to introducing a new simulator-specific language.

Another aspect that was taken into account in EDEN's design is *usability*. In addition to the benefits gained through NeuroML support, EDEN addresses usability through *automatic management* of multi-processing resources. This means that EDEN can distribute processing for a simulation across the processor cores of a personal computer–or even a computer cluster–fully automatically. Thus, users can fully exploit their modern computer hardware and deploy simulations of large networks on high-performance clusters, with no additional effort.

To evaluate all aforementioned features of EDEN, we employed: (1) qualitative benchmarks showing simulation fidelity to the standard neural simulator NEURON; and (2) quantitative benchmarks showing far superior simulation speed compared to NEURON, for networks of non-trivial size. The results of these benchmarks are expanded on in Section 3.

The contributions of this work are, thus, as follows:

- A novel neural simulator called EDEN supporting high model generality, computational performance, and usability by design.
- A novel model-analysis/code-generation technique that allows extracting the required calculations from a neural-network model, and casting them into efficient work items that can be run in parallel to simulate the network.
- A qualitative evaluation of EDEN, demonstrating NEURON-level fidelity, for a diverse set of neural models.
- A quantitative evaluation of EDEN, demonstrating simulation speeds of real-world neural networks (sourced from literature) up to close to two orders-of-magnitude faster than NEURON, when run on an affordable, 6-core desktop computer.

[1] Code generation as a general technique is prevalent in high-performance neural simulators—see Blundell et al. (2018)—but these simulators either analyse neuron models at a shallower level than our work does, or they support a narrow subset of neuron models, as we explain in the following.

TABLE 1 | Qualitative comparison between EDEN and other state-of-the-art neural simulators: NEURON (McDougal et al., 2017), CoreNEURON (Kumbhar et al., 2019), JLEMS (Cannon et al., 2014), BRIAN2 (Stimberg et al., 2019), GeNN (Yavuz et al., 2016), NEST (Gewaltig and Diesmann, 2007), and Arbor (Akar et al., 2019).

	EDEN	(Core) NEURON	Arbor	jLEMS	BRIAN2	NEST	GeNN
Supported models and features							
LIF, AdEx, Izhikevich cells	✓	✓	Only LIF	✓	✓	✓	✓
Custom artificial cells	✓	✓	×	✓	✓	Partially *via* NestML	Partially *via* NineML
Highly detailed multi-compartmental cells	✓	✓	✓	×	Not practical	×	×
Native NeuroML support	✓	×	×	✓	×	×	×
Overall support compared to EDEN	Baseline	✓	×	×	×	×	×
Performance							
Machine-wide parallelism	✓	Manual	✓	×	Only for simple cases	✓	✓
Cluster-wide parallelism	✓	Manual	✓	×	×	✓	×
Cluster-wide auto-parallelization of detailed networks with graded synapses	✓	×	×	×	×	×	×
Overall performance compared to EDEN	Baseline	×	×	×	×	✓†	✓†

†*Only for artificial-cell models that NEST and GeNN support.*

1.2. Qualitative Comparison of Neural Simulators

In **Table 1**, we present a qualitative comparison between our proposed simulator EDEN, and the most popular, actively developed simulators in the computational-neuroscience field. In line with the scope of this article, we consider the more general-purpose simulators that can be used in a batch-mode, brain-modeling setting. The table consists of two parts, the top half dealing with coverage of neuron models and features, and the bottom half dealing with aspects of computational performance. **Figure 1** also summarizes a qualitative comparison between the usability, range of supported models and computational performance of the various simulators. The characteristics and relative advantages of each simulator are further laid out in the following paragraphs.

NEURON (McDougal et al., 2017) is the popular standard simulator for biological and hybrid[2] neural networks. It supports the richest set of model features among neural-simulation packages. A characteristic feature of NEURON is that everything about the model can be changed dynamically while the model is being simulated. This allows simulation of certain uncommon models, but it negatively impacts the simulator's computational efficiency. CoreNEURON (Kumbhar et al., 2019) is a new simulation kernel for NEURON that improves computational performance and memory usage at the cost of losing the ability to alter the model during simulation. It does not affect setting up the simulator and the model, which are still performed in the same way. Due to the underlying architectural design, the user has to add custom communication code to allow parallel simulation with NEURON, though there is ongoing effort to standardize and automate the needed user code (Dura-Bernal et al., 2019).

Compared to NEURON, EDEN only supports the NeuroML gamut of models. However, EDEN has much higher computational performance that also automatically scales up with available processor cores and computational nodes. Also, setting up a neural network in NEURON requires the connection logic to be programmed in its own scripting language. This is a cumbersome task and, what is more, NEURON's script interpreter is slow and non-parallel, often resulting in model setup taking more time than the actual simulation. In contrast, EDEN can load networks from any neural-network generation tool that can export to NeuroML[3], thus leveraging the capabilities and computational performance of these tools.

Another simulator for biological neural networks is Arbor (Akar et al., 2019) which aims at high performance as well as model flexibility. Its architecture somewhat resembles the object model used by NEURON, which facilitates porting models, written in NEURON, to Arbor. However, compared to NEURON, it supports a smaller set of mechanisms. Regarding hybrid networks, modeling artificial cells is difficult; only linear integrate-and-fire (LIF) neurons are readily supported, and the user has to modify and rebuild the Arbor codebase for introducing new artificial-cell types. In addition, neuron populations connected by graded synapses cannot be distributed across machines for parallel simulation, which restricts scalability when running cutting-edge biological-neuron simulations. Compared to Arbor, EDEN supports about the same range of biophysical models but also supports *all types* of abstract-neuron models, while Arbor only supports LIF abstract neurons. This limitation prevents Arbor from supporting many hybrid networks. There is also a difference in usability: To set up a

[2]Neural networks with mixed populations of both artificial and biophysically modeled neurons.

[3]Common NeuroML-compatible model-generation tools: NetPyNE (Dura-Bernal et al., 2019), neuroConstruct (Gleeson et al., 2007), NeuroMLlite (https://github.com/NeuroML/NeuroMLlite).

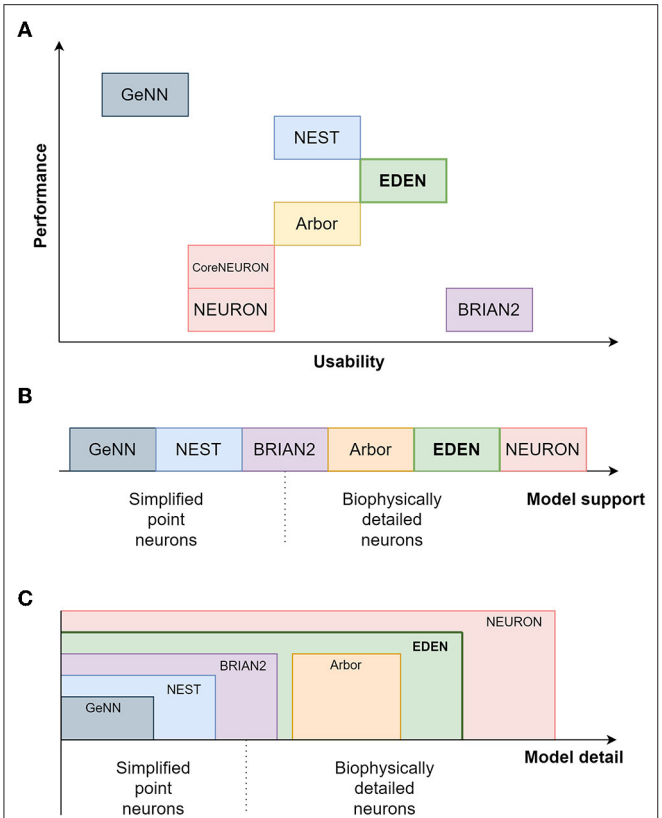


FIGURE 1 | A relative comparison of the characteristics of EDEN and the established neural simulators. **(A)** compares the simulators on the performance and usability plane; **(B)** shows the ordering between simulators regarding the level of model detail; and **(C)** shows the level of modeling detail supported by each simulator.

network model, the network-generation logic must be captured as Arbor-specific programming code. EDEN, instead, avoids simulator-specific programming by using a cross-tool file format.

In the space of artificial-cell-based spiking neural networks (SNNs), there are various specialized simulators in common use. jLEMS (Cannon et al., 2014) is the reference simulator for the LEMS side of NeuroML v2. It supports custom point-neuron dynamics through LEMS, which itself is a hierarchical-dynamics description language that co-evolved with NineML (Raikov et al., 2011). It was not designed for high performance and supports only simplified point neurons. BRIAN2 (Stimberg et al., 2019) is a simulator originally designed for point neurons, that focuses on usability and user productivity. It supports custom point-neuron dynamics, written in mathematical syntax. Its support for multi-compartmental cells is a work in progress; currently, all compartments must have the same set of equations. NEST (Gewaltig and Diesmann, 2007) and GeNN are general-purpose simulators for networks of point neurons and achieve high performance through a library of optimized codes for specific neuron types. Setting up the network is done through a custom programming language for NEST, and by extending the simulator with custom C++ code for GeNN. For NEST and GeNN, the way to add custom point-neuron types without modifying the C++ code is by writing the neuron's internal dynamics in a simulator-specific language; however, this method is not enough to capture all aspects of the model (such as multiple pre-synaptic points on the same neuron in NEST). Compared to abstract-cell simulators, EDEN has an advantage in model generality, since it also supports biophysically detailed multi-compartmental neurons, and hybrid networks of physiological and abstract cells. Although EDEN is not as computationally efficient as the high-end abstract-cell simulators, it readily supports *user-defined dynamics* inside the cells and synapses, whereas said high-performance simulators have to be modified to support new cell and synapse types. EDEN also supports non-aggregable synapses [i.e., not just types that can be aggregated into a single instance as per (Lytton, 1996)], and any combination of synapse types being present on any type of cell; which are also not supported by high-performance abstract cell simulators.

An important point to stress is that, the differences in supported model features, combined with the different model-description languages, make it difficult to reproduce the exact same neural network (and its output) across all simulators; this is especially the case for biologically detailed models. Thus, although there is much previous work on performance-driven neural simulation, our work is one of the first to directly compare performance with NEURON on physiological models that are drawn from existing literature, rather than employing synthetic ones. This further underscores the point that EDEN is a general-purpose tool that can be readily used with existing NeuroML models as well as in new NeuroML projects.

In the literature, the designers of CoreNEURON and Arbor have each reported utilizing the cores of a whole High-Performance Computing (HPC) node, to achieve up to an order of magnitude of speedup over NEURON. While the models and the machines used in those cases are not identical to ours for allowing a strict comparison, our demonstrated speedup of 1 to nearly 2 orders of magnitude over NEURON on a 6-core PC shows that EDEN is more than competitive against the state of the art in terms of computational performance. Furthermore, the fact that a regular desktop PC has been used for achieving such speedups makes the results highly relevant for a typical neuroscientist's computational resources.

2. METHODS AND MATERIALS

2.1. EDEN Overview

The architecture of EDEN can be visualized as a processing pipeline, which is illustrated in **Figure 2**. The pipeline details and the reasons for allowing EDEN to deliver high performance, model flexibility, and usability are explained in this section. While current neural simulators primarily focus on either computational performance or model generality, EDEN simultaneously achieves both objectives with a novel approach: it generates efficient code kernels that are tailored for the neuron models at hand.

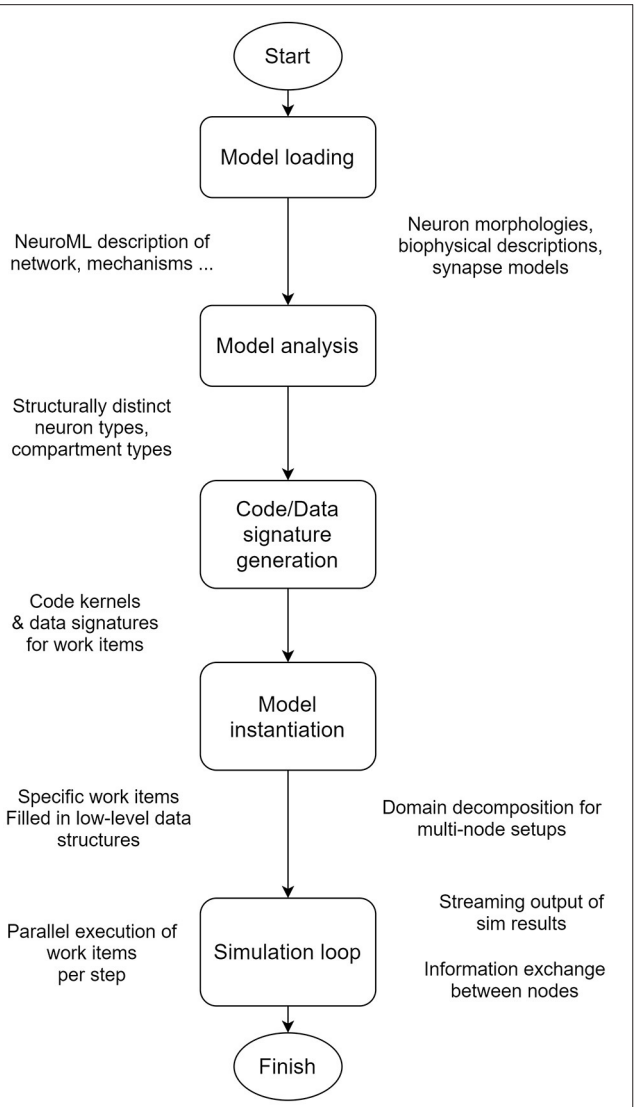


FIGURE 2 | EDEN's processing pipeline. The whole model is analyzed in order to extract the computationally similar parts of neurons, and to generate optimized code and data representations for them, on the fly.

EDEN performs time-driven simulation of any sort of neural network that can be described in NeuroML. To enable simulation of complex, and often heterogeneous, networks with high performance, EDEN first performs model- and workload-analysis steps so as to divide the simulation workload into independent, parallelizable components and, subsequently, determines *efficient code and data representations* for simulating each one of them. Finally, EDEN employs automatic code generation to convert these components to parallel-executable tasks (called work items). Code generation boosts computational efficiency by adapting performance-critical code to the specific model being simulated and to the specific hardware platform being used. Task parallelization boosts computational efficiency even further by distributing the simulation work across multiple CPU cores in a given computer, and across multiple computers in a high-performance cluster.

2.2. Usability Through Native NeuroML Support

Choosing NeuroML as EDEN's input format allows us to focus on our core part of high-performance numerical simulation and, at the same time, leverage the existing NeuroML-compatible, third-party tools for design, visualization, and analysis of neural networks. Adopting the standard also improves the simulator's usability, as the end user does not need to learn one more simulator-specific modeling language. In practice, directly supporting the NeuroML standard also allowed us to verify the simulator's results against the standard NEURON simulator, for numerous available models. As we will see in Section 3, the same NeuroML description can be used for both simulators and run automatically. Otherwise, porting all these models separately to both simulators would have taken an impractical amount of effort, making verification and comparison much more difficult to achieve.

2.3. Performance and Flexibility Through Code Generation

Neural models, especially biophysical ones, are commonly described through a comprehensive, complex hierarchy of mechanisms. Neural-simulation programmers have to consider this cornucopia of mechanisms and their combinations so as to form neural models. All the while, the formulations behind the mechanisms are constantly evolving, thus, allowing for no single set of mathematical equations to cover most (or even a few of the) neural models.

The resulting complexity—in both setting up a model and running the simulation algorithm—has steered general-purpose neural-simulation engines to adopt *object-oriented models* of the neural networks being run. Each type of programming object, then, captures a respective physiological mechanism, and the hierarchy of mechanisms in the model is represented by an equivalent object hierarchy. By adopting NeuroML, EDEN takes the same object-oriented approach at the model input.

Although this approach does help simplify the *programming model* by mitigating the conceptual and programming complexity of working with sophisticated models, it is detrimental to the *execution model* since it is an inefficient way to run the simulations on modern computer hardware. The object-oriented data structure of a model in use has to be traversed, every time the equations of the model are evaluated and the model's state is advanced. The traversal logic in use enforces a certain ordering among the calculations that are needed to advance the model's state. Also, the object-oriented model's pointer-based data structures make control flow and data-access patterns unpredictable, slowing down the processing and memory subsystems of the computer, respectively.

For example, NEURON advances the state of the network in successive stages: Within the scope of one parallel thread, the processing stages of (a) evaluating current and conductivity for all membrane mechanisms present, (b) solving the cable

equation for all neurons, and (c) advancing the internal state of all membrane mechanisms are performed in strict sequence. Since each part of these three stages pertains to a specific compartment of the network, and yet processing of these stages for the same compartment is separated in time by processing for the whole network, this ordering is detrimental to data locality. In accelerator-enabled implementations of this technique, namely CoreNEURON and Arbor, the mechanisms with identical mathematical structure are grouped together and executed in an even stricter sequence within the original phases of processing. This exacerbates the impact to locality and introduces synchronization overhead that increases with model complexity, as parallelization is only applied across identical instances of each mechanism type.

Now, starting from the computer-architecture part of the problem, HPC resources are designed so that the maximum amount of computations can be done independently and simultaneously. Thus, fully utilizing them requires streamlined algorithms and *flat data structures*. In many cases, neural-simulation codes have been custom-tailored for the HPC hardware at hand. Although such codes improve simulation speed and supported network size by orders of magnitude compared to general-purpose simulators, they make inherent model assumptions that *prevent* them from supporting other models. The result is that these manually optimized codes, as well as the knowledge behind them, are abandoned after the specific experiment they were developed for is concluded.

To avoid the pitfalls of these two approaches, EDEN consciously refrains from imposing a specific execution model, so that it can support both model generality and high-performance characteristics. Both of them are simultaneously achieved through a novel approach: efficient code kernels that are tailored for the neuron models at hand are automatically generated, while supporting the whole NeuroML gamut of network models. The specific processing stages that EDEN undergoes to achieve this (see **Figure 2**) are as follows:

1. Analyse all types of neurons in a given model.
2. Deduce the parts of the neural network that have a similar mathematical structure.
3. Produce efficient code kernels, each custom-made to simulate a different part of the network.
4. Iteratively run the code kernels to simulate the network.

This code-generation approach used by EDEN has manifold benefits: First, the simulation can be performed without traversing the model's hierarchy of mechanisms at run time, since the set of required calculations has already been determined at setup time. Second, since the generated code contains only the necessary calculations to simulate a whole compartment or neuron, the compiler is given much more room for code optimization compared to code generation for individual mechanisms. Third, the minimal set of constraints that EDEN's backend places on the code of work items allows incorporating hand-written code kernels that have been optimized for specific neural models. This is also made possible due to the model-analysis stage, which isolates groups of neurons and/or compartments with an identical mathematical structure; when a hard-coded kernel is available for a detected neuron type, it can be employed for the specific cell population, to further boost performance. Thus, EDEN's model-decomposition and code-generation architecture delivers high computational performance for a general class of user-provided neuron models, and it also permits *extensions* in both the direction of model generality and computational performance.

For this first version of EDEN, a *polymorphic kernel generator*[4] that supports the full gamut of NeuroML models was implemented. The specifics of the code kernels are customized for each neuron type; still, the generator's format covers any type of neuron, whether it is a rate-based model, an integrate-and-fire neuron or a complex biological neuron, or whether the interaction is event-based, graded, or mixed. Thus, this implementation provides a baseline of computational efficiency, for all neural models. It can also work in tandem with specialized kernels. Two ways of extending EDEN with such specialized high-performance codes are described below.

The simplest way to integrate an existing code in EDEN is to directly use it just for the models that the code supports. Programming-wise, the neural network to be run is checked whether it can be run on the new code, and if this is the case, the original new code is generated as a work item, and the simulation data is accordingly allocated and initialized for the model. By running the same code on the same data, extended EDEN should perform as well as the original EDEN code, for the supported family of models.

Alternatively, if the specialized code applies to only a part of the desired network, it can interface with work items from EDEN's general-purpose implementation (or other extensions) for the part of the network that it does not cover. Some modification is then necessary to make the code exchange information (such as synaptic communication) in the same way as the work items it is connected to, but the gains in model generality are immediate.

Following these methods, the usefulness of the optimized code is extended with the least possible effort, simulation can utilize multiple computational techniques at the same time, and the details of each technique do not affect the rest of the EDEN codebase.

2.4. EDEN Concepts

2.4.1. Work Items

The fundamental units of work executed per each simulation step in EDEN are called "work items." The work items are parts of the model that can be processed in parallel within a simulation step, to advance the state of the simulated model. Within a time-step, each work item is responsible for updating a small part of the entire model. Each work item is associated with a single part of the model data being simulated, and a single code block being run. That code block is responsible for updating the mutable part of its model data over time, but it may also update other parts as well, so that it can send information to other parts of the model. One such case is transmission of spike events to post-synaptic

[4]Polymorphic means that it adapts to the neuron's structure, instead of handling just one type.

components. Then, data-access collision with the work item that is assigned to the post-synaptic component is avoided by double buffering; the work item receiving the information reads it on the next time-step, while leaving the alternate buffer available for other work items to write to. In the case multiple other work items may write simultaneously, atomic memory accesses are used.

In this first version of EDEN, each work item involves simulating exactly one neuron, but the design allows further variations—for example, to consolidate simple neurons in batches, or to split large neurons in parts—as long as the calculations for each work item are independent.

2.4.2. Code and Data Signatures

EDEN generates compact code and data representations to run the simulation, by composition of the multiple underlying parts. The details of how this works are explained below.

Each simulated mechanism is defined by its dynamics, the fixed parameters and state variables of the dynamics, and the variables through which it influences other mechanisms, and is influenced by other mechanisms. The external variables influencing the mechanism are called *requirements*, and the values it, in turn, presents for other mechanisms to use, are called *exposures*. Then, to simulate the mechanism, the required actions are:

- to evaluate all variables involved in the dynamical equations (called "assigned" henceforth, in EDEN as well as NEURON parlance). This is the "evaluation" step of the simulation code.
- then, to advance the simulation's state based on the dynamics, and the current values of the assigned variables. This is the "update" step of the simulation code.

The whole set of code and data for simulating a mechanism is collectively called a *signature* in EDEN parlance. Examples of code-data signatures are shown in **Figure 3**, for simple cases of a post-synaptic component and an ion channel. Each of them consists of the code for running the "evaluation" and "update" steps (also called *code signature*), and the data representing the mechanism (also called *data signature*). The signature representation is used in EDEN both for simple mechanisms and composite ones. In fact, the signatures of smaller mechanisms are successively merged to form the signatures of higher-order parts of the neurons, eventually forming signatures for whole compartments or even entire neurons. The code of such signatures is then run in parallel, in order to simulate the whole neural network.

In order to combine the signatures representing two mechanisms, the interfaces (i.e., requirements and exposures) through which the mechanisms interact have to be determined. The hierarchical structure of the provided neural models helps in this, since it delineates the interfaces through which the "parent" mechanism interacts with its "children," and the "siblings" interact with each other.

Code generation starts from the simple, closed-form mechanisms present (for example, Hodgkin-Huxley rate functions or plasticity factors of mechanisms). The hierarchy of mechanisms present in a neuron is traversed, and signatures are incrementally formed for each level of the hierarchy.

The specific steps to merge two signatures are then, in terms of code and data:

- The "evaluation" parts of the code signature have to be placed in a certain order, such that after the variables each mechanism requires are defined and evaluated before the mechanism's evaluation code.
- The "update" parts of the code signature can be appended anywhere after the mechanism's corresponding evaluation code.
- The data signatures of the mechanisms are simply concatenated to each other.

As signatures are generated for each higher or lower level mechanism, an auxiliary data structure that has the same hierarchical structure as the original object-oriented model is also formed. This is called the *implementation* of the signature, and it keeps track of how the conversion to signatures was performed, for each mechanism. Relevant information includes the specific decisions made for the generated code (like selection of ODE integrator for the particular mechanism), and the mapping of abstract parameters and state variables (such as the gate variable of an ion channel, the fixed time constant of a synapse, the membrane capacitance of a compartment, etc.) to the specific variables allocated in the data signature. The information is useful for:

- referring to parts of the network symbolically (like when recording trajectories of state variables, and when communicating data-dependencies between machines in multi-node setups),
- initializing the data structures through the symbolic specifications in use (such as weights of specified synapses),
- properly combining signatures, according to implementation decisions (e.g., adjusting the update code to the integrator in use).

2.4.3. Data Tables and Table-Offset Referencing

To achieve high performance during simulation, EDEN uses a simplified data structure for the model. The model's data are structured in a set of one-dimensional arrays of numbers. These arrays (called *tables* henceforth) are grouped by numerical type (such as integral or floating-point), and mutability (whether their values remain fixed along the simulation, or they evolve through time). This means that each value in the model being simulated is identified by the table it belongs to, its position in the table, and the value's numerical type and mutability.

The value's location can then be encoded into an integer, from the table's serial number and the offset on the table. The code generated by EDEN can use such references to values at run time, to access data associated with other work items. This relieves EDEN's simulation engine from the need to manage communication between parts of the model with a fixed implementation. Instead, control is given to the work items' generated code on how to manage this communication effectively.

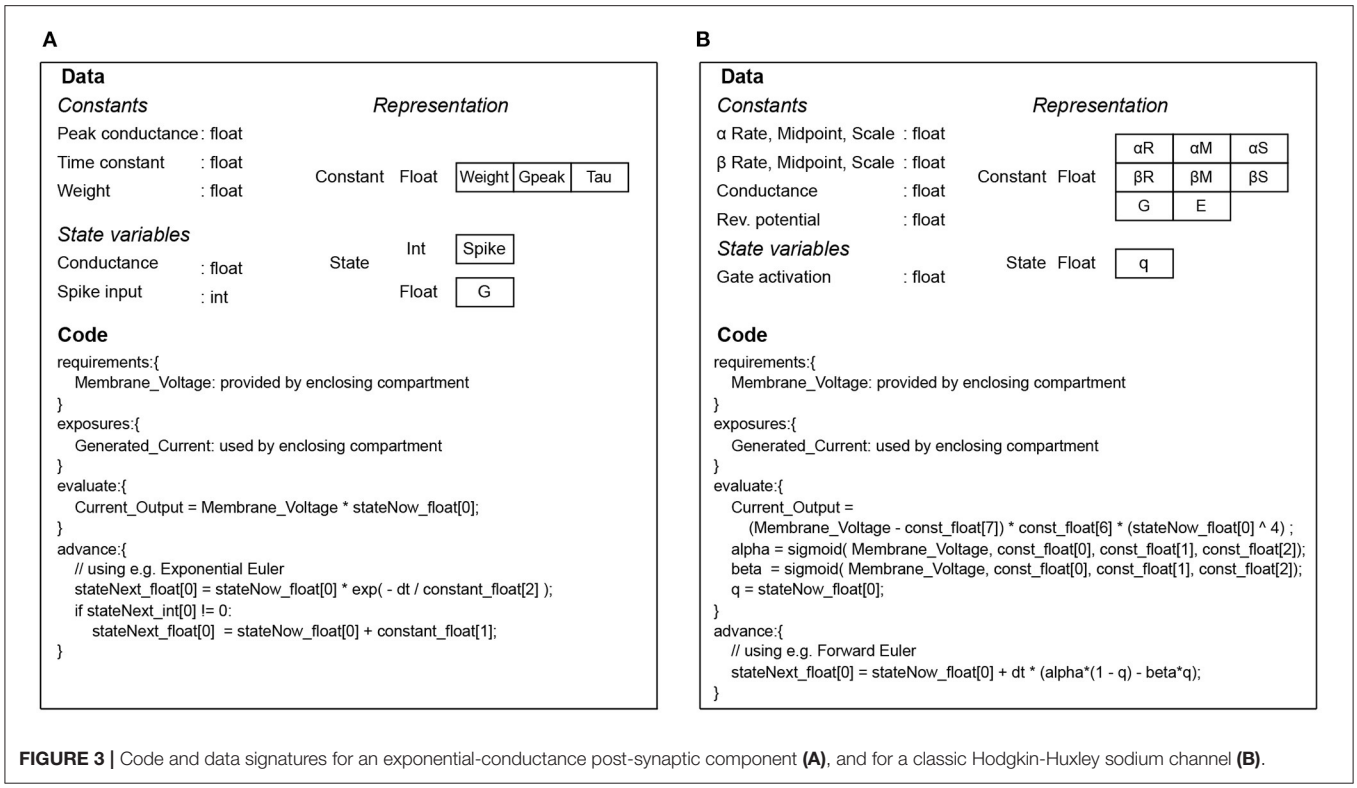


FIGURE 3 | Code and data signatures for an exponential-conductance post-synaptic component **(A)**, and for a classic Hodgkin-Huxley sodium channel **(B)**.

Another benefit of the table-offset referencing scheme is that the references can be redirected to any location in the model's data, if need arises. This is used in particular when a model is run on a computer cluster, where parts of the network are split between computers. In this case, only a fraction of the model is realized on each machine, and the data read by or written to remote parts of the network are redirected to local mirror buffers instead. The change is automatically applied by editing the references in the instantiated data, hence there is no need to change the generated code for the work items.

2.5. Implementation

The present implementation of EDEN takes as input NeuroML and supports all neural models in the NeuroML v2 specification. This implementation, and the code kernels it generates, can be used as a fall-back alternative to further extensions: the extensions can provide specialized implementations for specific parts of the neural network, while the rest of the network is still covered by the fully general, original implementation.

2.5.1. Structure of the Program

To begin analysis and simulation of a neural network, its NeuroML representation, along with additional LEMS components describing the custom neuron mechanisms present, is loaded into an object-oriented representation.

The main steps of the process are:

1. Model analysis
2. Work-item generation through code and data signatures
3. Model simulation in the EDEN simulation engine

These steps are further described in the following sections.

2.5.2. Model Analysis and Code Generation

The first part in model analysis is to associate the types of synapses in the network with the neuron types they are present in. This resolves which types of neurons contain which kinds of synaptic components, and where each kind of synapse is located on the neuron. The same assessment is also made for the input probes connected to each neuron, since probes are also part of the neurons' models.

Then, each neuron type is analyzed, to create a signature for each kind of neuron. First, the structure of the neuron is split into compartments, and the biophysical mechanisms applied over abstract groups of neurite segments are made explicit against the set of compartments. Thus, for each compartment, we get the entire list of biophysical mechanisms existing on it. Using these lists, the corresponding code and data signature is formed for each individual compartment.

If the number of compartments is small, these signatures are concatenated for all compartments present on the neuron, into a neuron-wide signature. This way, a compact code block is generated, with a form similar to how hand-made codes are written for reduced compartmental neuron models. The process is illustrated in **Figure 4**.

2.5.2.1. Signature Deduplication for Identical Compartments

If the number of compartments is large, it is not practical to generate a flat sequence of code instructions for each individual compartment. However, in practice, neuron models have less than a few tens of distinct compartment types with different

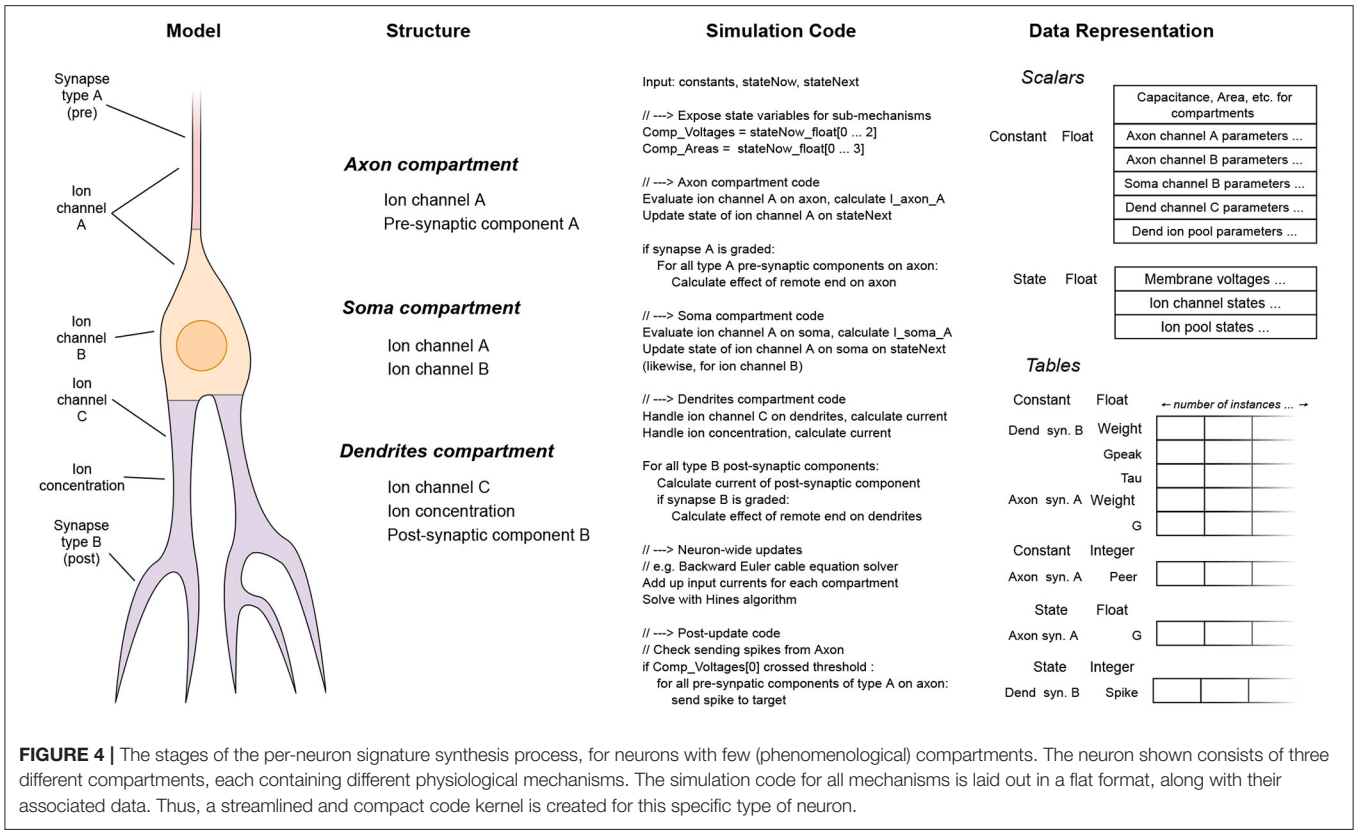


FIGURE 4 | The stages of the per-neuron signature synthesis process, for neurons with few (phenomenological) compartments. The neuron shown consists of three different compartments, each containing different physiological mechanisms. The simulation code for all mechanisms is laid out in a flat format, along with their associated data. Thus, a streamlined and compact code kernel is created for this specific type of neuron.

mathematical structure, in the most complicated models. Thus, a different approach called signature de-duplication is employed, as follows. In this approach, the compartments are grouped for processing, according to their structural similarity (equivalently, similarity of signature representation). The process is illustrated in **Figure 5**.

Using the per-compartment list of mechanisms, we can immediately deduce which compartments have the exact same structure; which is the case when the set of mechanisms, and thus the code and data signature representation, is the same. The code signature for the whole neuron now has a set of loops, one for each type of compartment. Inside the loop, the code signature to simulate a single compartment is expanded. Each iteration of the loop performs the work for a different compartment with the same structure. Thus, the data signatures are concatenated together for each group of compartments, and the appropriate offsets are shifted in each iteration of the loop, so that they point to the specific instance of the per-compartment data signature to be used each time. By generating a specific code block for each sort of compartment, we eliminate the computational overhead of traversing the individual mechanisms present on each simulation step, that affects previous general-purpose neural simulators. Finally, after the code signatures for the work items are determined, they are compiled to machine code, and loaded dynamically on the running process.

2.5.3. Model Instantiation

After the model is analyzed to determine the structure of the work items it is converted to, it is time for the work items and their associated data to be realized in memory. The process that we describe in the following is also illustrated on **Figure 6**. As mentioned previously, in this version of EDEN, each neuron in the network, along with the synaptic components and input probes attached to it, is assigned to an individual work item. The mapping of parts of the network to work items, is thus fixed.

The data signatures of the work items specify the number of scalar variables and tables each work item uses. Thus, to instantiate each work item, we just have to allocate the same number of scalars and tables, and keep track of the work item for which these blocks of memory were allocated. After the variables are allocated for each work item instance, they are filled in, according to the model definition. This is made possible by the implementations of the work items, that keep track of how model-specific references to values map to concrete data values for each work item. Thus, the changes between different instances in the specified model are mapped into changes in the low-level data representation.

The scalar values for each instantiated work item are located in contiguous slices of certain tables, which are reserved for each type of scalar values. Other parts of a work item may not have a fixed size every time. This is, for example, the case for synaptic components of a given type; they may exist in multitudes on a compartment of a neuron, and their number varies across

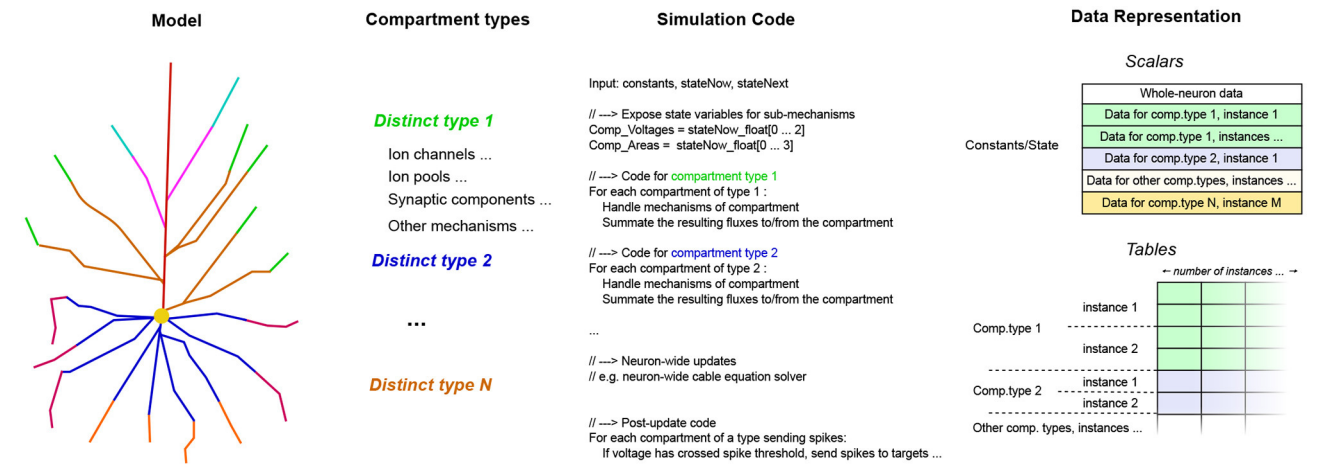


FIGURE 5 | The stages of the per-compartment signature de-duplication process, from the abstract model to the concrete implementation. On the schematic of the detailed neuron model, distinct compartment types are shown in different colors. The components sharing the same type are then grouped together, in terms of simulation code and data representation. The specific mechanisms comprising the compartments and the data cells they contain are not shown here, for brevity.

instances of the neuron or compartment type. The data for these variable-sized populations is stored in tables; one set of tables per kind of mechanism on the same compartment. That way, although the sizes of each set of tables may vary eventually, the number of scalars and individual tables required for a work item remains fixed, for all of its instances.

After allocating the scalars and the tables for the model, what remains is to replace default scalar values with per-instance overrides specified by the model where they exist, and to fill in the allocated tables with their variable-sized contents. The customized scalar values and tables pertaining to the inner models of neurons (where the "inner model" excludes the synapses and input probes attached to the neuron) are filled in while running through the list of neurons specified in the model. The synaptic connections in the network model are also run through, and the corresponding pre- and post-synaptic components are instantiated on the connected cells. More specifically, on each cell, the tables representing the specified synaptic component are extended by one entry each, with the new entries having the values of the scalar properties of the mechanism. The default values for these properties are provided by the data signature of the mechanism, and customized values (such as weight and delay of the synapse) are filled in using the connection list in the model description.

2.5.4. Simulation Loop

After model instantiation is done, the code blocks and data structures for the model are set up in system memory and ready to run. Communication throughout the network is internally managed by the code blocks, *via* a shared-memory model. Double buffering is employed to allow parallel updates of the state variables within a time-step, thus all state-variable tables are duplicated to hold the state of the both the old and new time-step as the latter is being calculated.

All that remains to run the simulation, is to repeat the following steps for each simulation time-step:

- set the global "current time" variable to reflect the new step;
- execute the code for each work item in parallel, on the CPU;
- output the state variables to be recorded in the network, for the new time-step;
- alternate which set of state variable buffers is read from and written to, as per the common double-buffering scheme.

Parallel execution of the code kernels is managed by the OpenMP multi-threading library. The "dynamic" load-balancing strategy is followed by default, so whenever a CPU thread finishes executing a work item, it picks the next pending one. The synchronization overhead of this load-balancing strategy is mitigated by the relatively large computational effort to simulate physiological models of neurons, as will be shown in Section 3.

2.5.5. Numerical Methods

The numerical integration methods that EDEN employs in this version are simple but they are sufficient to provide accurate simulation, as we will demonstrate in Section 3. All calculations are done with single-precision arithmetic except for expressions involving the amount of simulated time, which is represented with double precision since it changes by microseconds throughout up to hours. The state of synapses and of most membrane mechanisms is advanced using the Forward Euler integrator. An exception is made for the gate variables of Hodgkin-Huxley ion channels with alpha-beta rate (or, equivalently, tau-steady state) dynamics, where NEURON's `cnexp` integrator (i.e., Exponential Euler under the assumption that the transition rates are fixed throughout the timestep) is employed. To simulate the diffusion of electrical charge within each cell, we use a linear-time, Gaussian-elimination method that is equivalent to the Hines algorithm (Hines, 1984) used in NEURON.

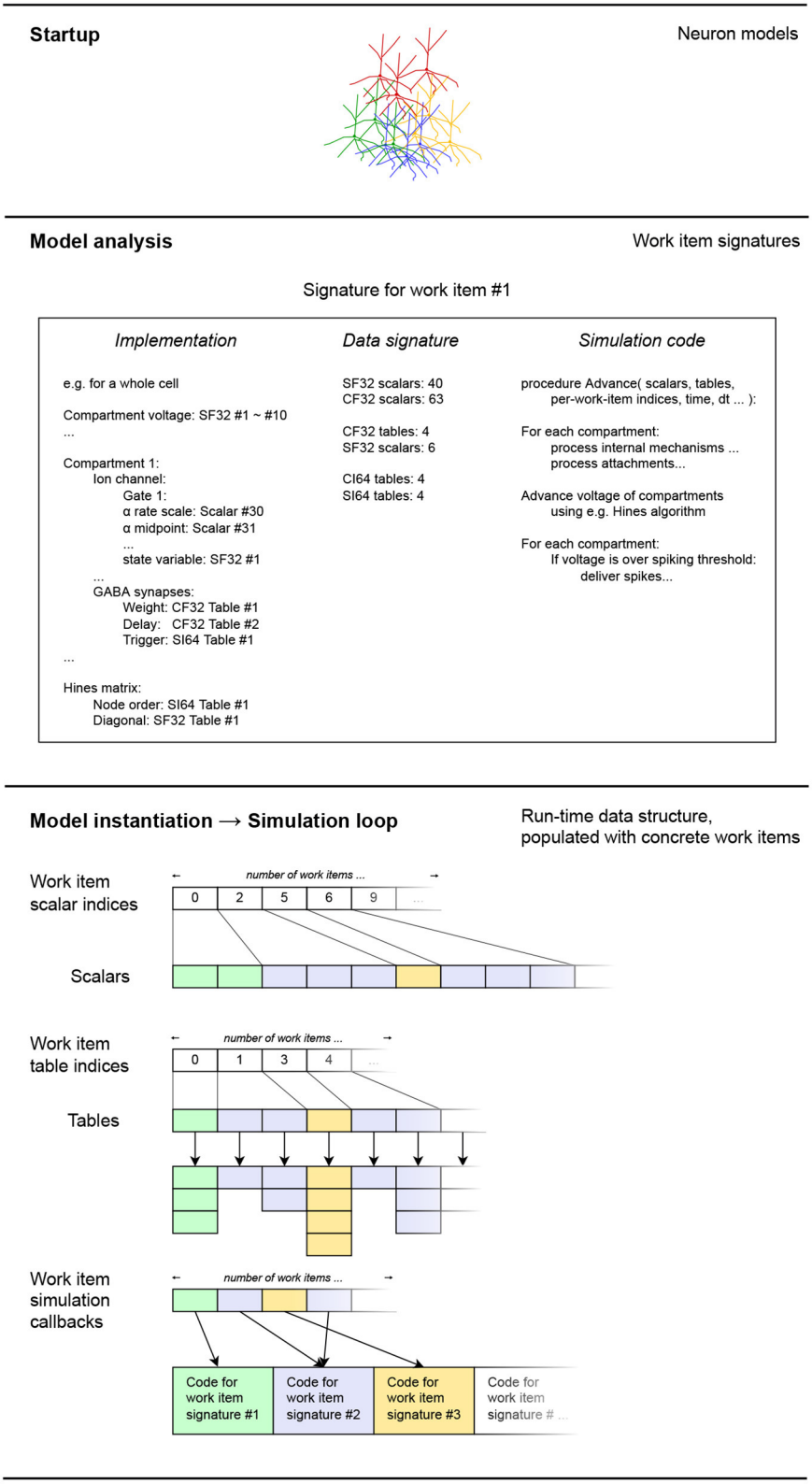


FIGURE 6 | A schematic representation of how the extracted work item signatures are converted to low-level data structures for efficient processing. For each work item, the set of scalars and work tables of each is appended into flat node-wide arrays, for each data type. Data types shown: CF32 = 32-bit floating-point constants, CF64 = 64-bit integer constants, SI64 = 64-bit integer variables. The different data types for scalars and tables have been omitted for clarity in the diagram, without loss of generality. Colors indicate different code-data signatures among work items.

2.5.6. Running on Multi-Node Clusters

Apart from the high-performance properties implemented in EDEN for fast simulation on a single computer, EDEN also supports MPI-based execution on a compute cluster, so as to further handle the large computational and memory needs of large simulations. To distribute the simulation over multiple co-operating computational nodes, some modifications are made to the process described above. In the following, each co-operating instance of EDEN is called a "node."

At the model-instantiation stage, the nodes determine which one will be responsible for simulating each part of the neural network. The neurons in the network are enumerated, and distributed evenly among nodes. To keep a small and scalable memory footprint, in this version of EDEN, each node is responsible for a contiguous range of the enumerated sequence of neurons. Then, each node instantiates only the neurons it is responsible to simulate, allocating the corresponding scalar values and tables. The parts that pertain only to individual neurons are also instantiated and filled in. But special care has to be taken when instantiating synapses, since they are the way neurons communicate with each other—and the neurons a node is managing may communicate with other neurons, that are managed by a different node. Thus, the instantiation of synapses is performed in three stages:

1. an initial scan of the list of synapses, to determine which information is needed by each node from each node during the simulation;
2. exchange of requirement lists among nodes, so they all are aware of which pieces of information they must send to other nodes, during the simulation;
3. establishment of cross-node mirror buffers, and remapping cross-node synapses so that they use these buffers, to access the non-local neurons they involve.

To support these stages, a new auxiliary data structure is created on each node. It is an associative array, mapping the identifiers of peer nodes to the set of information that needs to be provided by that node to run the local part of the simulation (called *send list* from now on). A send list consists of the spike event sources and state variables on specific locations on neurons, that the node needs to be informed about to run its part of the simulation. The kinds and locations for these state variables and spikes, are stored and transmitted using a symbolic representation, that is based on the original model description. For example, a location on a neuron is represented by the neuron's population and instance identifiers, the neurite's segment identifier, and the distance along that segment from the proximal to the distal part. Using symbolic representations for send lists allows each node to use the most efficient internal data representation for its part of the model, without requiring peer nodes to be aware of the specific data representation being used on each node. The three stages to set up multi-node coordination are further described in the following:

2.5.6.1. Synapse-Instantiation Stage

First, the list of synaptic connections is scanned, and synapses connecting pairs of neurons are handled by each node according to four different cases:

1. If a synapse connects two neurons managed on this node, it is instantiated, and the tables are filled in just as described above, for the single-node case.
2. If neither neuron connected by the synapse is managed by this node, the synaptic connection is skipped.
3. If the local neuron needs to receive information from the remote neuron (as is the case with post-synaptic neurons and those with bi-directional synapses), then the location on the remote neuron and type of data (e.g., spike event or membrane voltage), is added to the send list for the remote node. The local neuron's synaptic mechanism is also instantiated using its data signature, however:
   - If the synaptic mechanism is continuously tracking a remote state variable (as is the case with graded synapses), the table-offset reference to that variable is set with a temporary dummy value. This entry is also tracked, to be resolved in the final synapse fix-up stage.
   - If the mechanism receives a spiking event from a remote source (as is the case with post-synaptic mechanisms), the mechanism receives the spike event in one of its own state variables, instead of tracking a remote variable. (This is the same way event-driven synapses are implemented in the single-node case.) The state variable is used as a flag, so custom event-based dynamics are handled internally. Thus, this entry has to be tracked, so that its flag can be set whenever the remote spike source sends a spiking event, at runtime.
4. If the locally managed neuron does not need to receive information, then is it skipped. The need for this node to send information to other peers will be resolved in the following send-list exchange stage.

2.5.6.2. Send-List Exchange Stage

At this point, the send lists have been determined, according to the information each node needs from the other nodes. These send lists are then sent to the nodes the data is needed from; sending nodes do not have to know what they are required to send *a priori*. Therefore, the algorithm described in the following also applies to the more general problem of distributed sparse multigraph transposition (Magalhães and Schürmann, 2020).

In the beginning of this stage, each node sends requests to the nodes it needs data from; each request contains the corresponding send list it has gathered. Then, from each node it sent a request to, it awaits an acknowledgement. While nodes are exchanging send lists, they also participate asynchronously in a poll of whether they have received acknowledgements for all the requests they sent. When all nodes have received all acknowledgements, this means all send lists have been exchanged, and the nodes can proceed to the next stage.

By using this scheme, information is transmitted efficiently in large clusters: no information has to be exchanged between nodes that do not communicate with each other. This is a scalability improvement over existing methods, where the full matrix of connectivity degrees among nodes is gathered on all nodes (Vlag et al., 2019; Magalhães and Schürmann, 2020).

2.5.6.3. Synapse Fix-Up Stage

After all data dependencies between nodes are accounted for, each node allocates communication buffers to send and receive spike and state-variable information. The buffers to receive the required information are allocated as additional tables in the data structures of the model. They are "mirror buffers" that allow each node to peek into the remote parts of the network they need to.

The table-offset references that were left unresolved in the synapse-instantiation stage because the required information was remote, are now updated with references to the mirror buffers for the corresponding remote nodes. This way, the components of cross-node synapses that—were the simulation run on a single node—would directly access the state of adjacent neurons, now access these mirror buffers instead. The mirror buffers are, in turn, updated on every simulation step as described in the next section, maintaining model integrity across the node cluster.

2.5.6.4. Communication at Run-Time

After the additional steps to instantiate the network on a multi-node setup, the nodes also have to communicate continuously during the simulation. Each node has to have an up-to-date picture of the rest of the network its neurons are attached to, to properly advance its own part of the simulation. Thus, the simulation loop is extended with two additional steps: to send local data to other nodes that need them, and to receive all information from other nodes it needs to proceed with the present time step.

The nodes follow a peer-to-peer communications protocol, which resembles the MUSIC specification (Ekeberg and Djurfeldt, 2008). The data sent from each node to a peer per time-step form a single message, consisting of:

- A fixed-size part, containing the values of state variables the receiving node needs to observe.
- A variable-size part, containing the spike events that occurred within this communication period. The contents are the indices of the events that just fired, out of the full list of events previously declared in the send list.

During transmission, each data message is preceded by a small header message containing the size of the arriving message; this is done so that the receiving node can adjust its message buffer accordingly.

After receiving the data message, the fixed-size part is directly copied to the corresponding mirror buffer for state variables, while the firing events in the variable-sized list are broadcast to the table entries that receive them. Broadcasting of firing events is performed using the spike recipient data structure that was created in the synapse instantiation stage.

Inter-node communications are placed in the simulation loop, as follows:

- In the beginning of the time step, the information to be sent to other nodes is picked from this node's data structures, into a packed message for each receiver. Transmission of these packed messages begins;
- Meanwhile, the node starts receiving the messages sent by other nodes to this one. Whenever a message arrives, it is unpacked and the contents are sent to mirror buffers and spike recipients in the model's data.
- When messages from all peers for this node are received, the node can start running the simulation code for all work items, while its own messages are possibly still being sent;
- Before proceeding with the next simulation step, the node waits until all messages it started sending have been fully sent; so, then, the storage for these messages can be re-used to send the next batch of messages.

3. RESULTS

During development of the EDEN simulator, we ran functional and computational performance tests, using NeuroML models from the existing literature. The functional tests were used to ensure that the simulator properly supports the various model features specified by NeuroML, and that its numerical techniques are good enough, with regard to stability and numerical accuracy.

The NeuroML-based simulations used in the experiments here were sourced from the Open Source Brain model repository (OSB) (Gleeson et al., 2019), and from the NeuroML-DB (Birgiolas et al., 2021). They were selected to cover a wide range of models in common use (regarding both level of detail and model size), and because their results clearly show various features of neural activity, and how each simulator handles them.

The simulations for the functional tests included all neuron models available on the NeuroML-DB and also present in the more general ModelDB (McDougal et al., 2017), and the smaller version of each network used in the performance tests. A visual comparison of output trajectories for various other OSB models is included in the **Supplementary Material**, in order to illustrate some finer details of the differences between the simulators. The performance tests were done on large neural networks in order to evaluate EDEN's computational efficiency and scalability, in various realistic cases.

Both simulation accuracy and performance characteristics were compared to the standard NeuroML simulation stack for biophysical models: the NEURON simulator (version 7.7), with the NeuroML-to-NEURON exporter jNeuroML (version 0.10.0). Model-porting complications were thus avoided by using the same NeuroML model descriptions. NEURON is the most commonly used general-purpose neural simulator, its numerical algorithms have been proven through decades of use, and it also enjoys the most complete NeuroML support to date (through the jNeuroML tool).

3.1. Evaluation of Functional Correctness

3.1.1. Evaluation Through Single-Neuron Models

Each neuron model present in the NeuroML-DB and also present in the ModelDB was individually simulated, to compare EDEN's results with NEURON's in each case. The protocol used was to stimulate each neuron with a 2 nA DC current clamp on its soma, from 10 to 90 ms of simulated time, with total simulation time being 100 ms. A fixed timestep of 0.025 ms was used for all simulations. The one recorded waveform for each neuron was membrane voltage on the soma. This is one type in the array

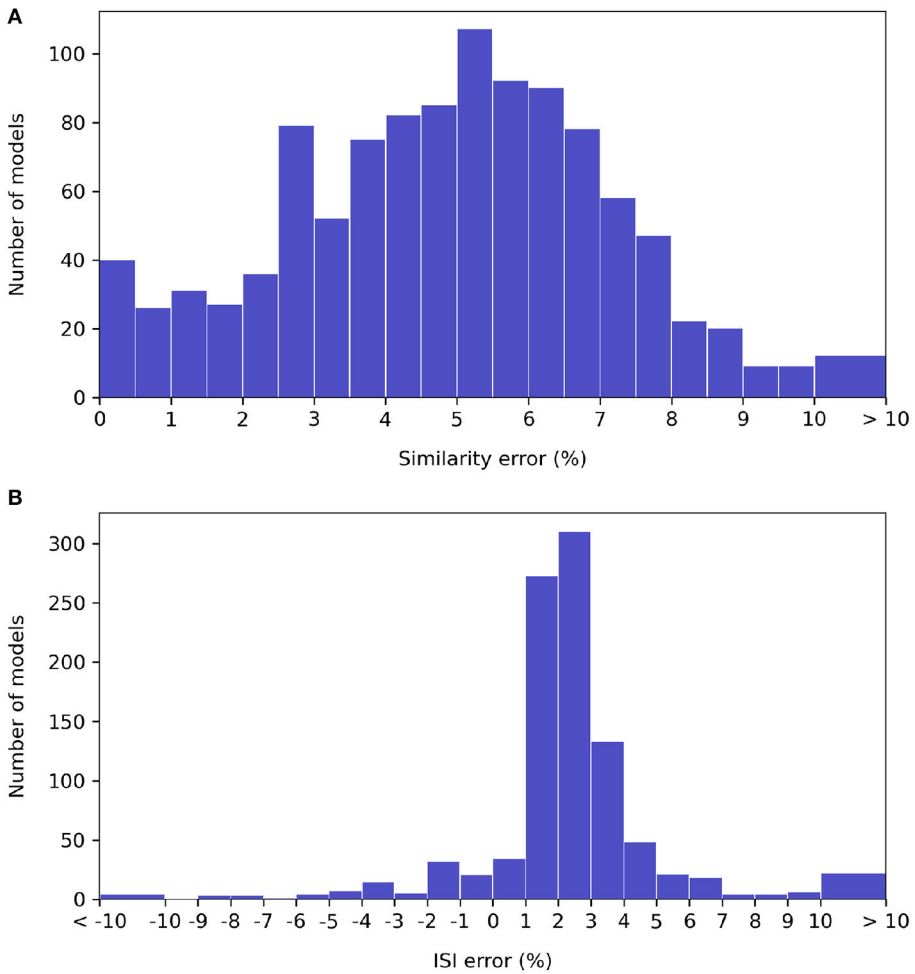


FIGURE 7 | Histograms of relative error under the NeuroML-DB similarity **(A)** and inter-spike interval **(B)** accuracy metrics for each NeuroML-DB neuron model. The bins around the "<-10" and ">10" limits include all models with more than 10% of discrepancy.

of tests already used in the NeuroML-DB, to characterize the electrophysiology of each neuron model.

The similarity metrics being assessed for EDEN's resulting waveforms, using NEURON's waveforms as reference, are:

- per-cent difference in inter-spike interval (ISI), assuming a spike threshold of $-20mV$,
- the NeuroML-DB similarity metric $1 - \frac{mean(|x-\hat{x}|)}{max(x)-min(x)}$, where x, $\hat{x}$ are the reference and tested waveforms. This one is used throughout NeuroML-DB to measure the discrepancy of NEURON's results under different (fixed) simulation step sizes, to determine an optimal step size that balances error with simulation time.

In total, EDEN failed to run seven models, whereas jNeuroML failed to run 24 models, out of 1,105 neuron models in total. EDEN could not run said seven models because they contain minor LEMS features it does not support at the time—though all these models can still work with a minimal, equivalent change to their description. We speculate that jNeuroML could not run said 24 models because of a defect in its support for certain types of artificial cells.

A histogram of waveform accuracy under each metric for the specified timestep, over all neuron types, is shown on **Figure 7**. (The models that either EDEN or jNeuroML could not simulate are excluded).

We observe that the EDEN's discrepancy against NEURON under the NeuroML-DB similarity metric is centered around 5%, and 99% of the models run under EDEN at <10% of waveform error. Under the inter-spike interval metric, EDEN's difference with NEURON is centered around +2.5%, with 90% of models having <±5% difference and with 98% of models having <±10% difference in mean inter-spike interval compared to NEURON.

Regarding error in the NML-DB metric, this is typically high for certain models with a high firing rate; as the waveforms get out of phase this metric drops sharply, even though the waveform of a single firing period has the same overall shape[5]. Regarding the discrepancies in firing period: We compared the mechanisms present on each neuron model with a high ISI

[5]This effect had been discounted in the evaluation method used in the Rallpacks (Bhalla et al., 1992), by linearly distorting the waveforms of inter-spike intervals to have the same nominal duration for both simulators, before comparing.

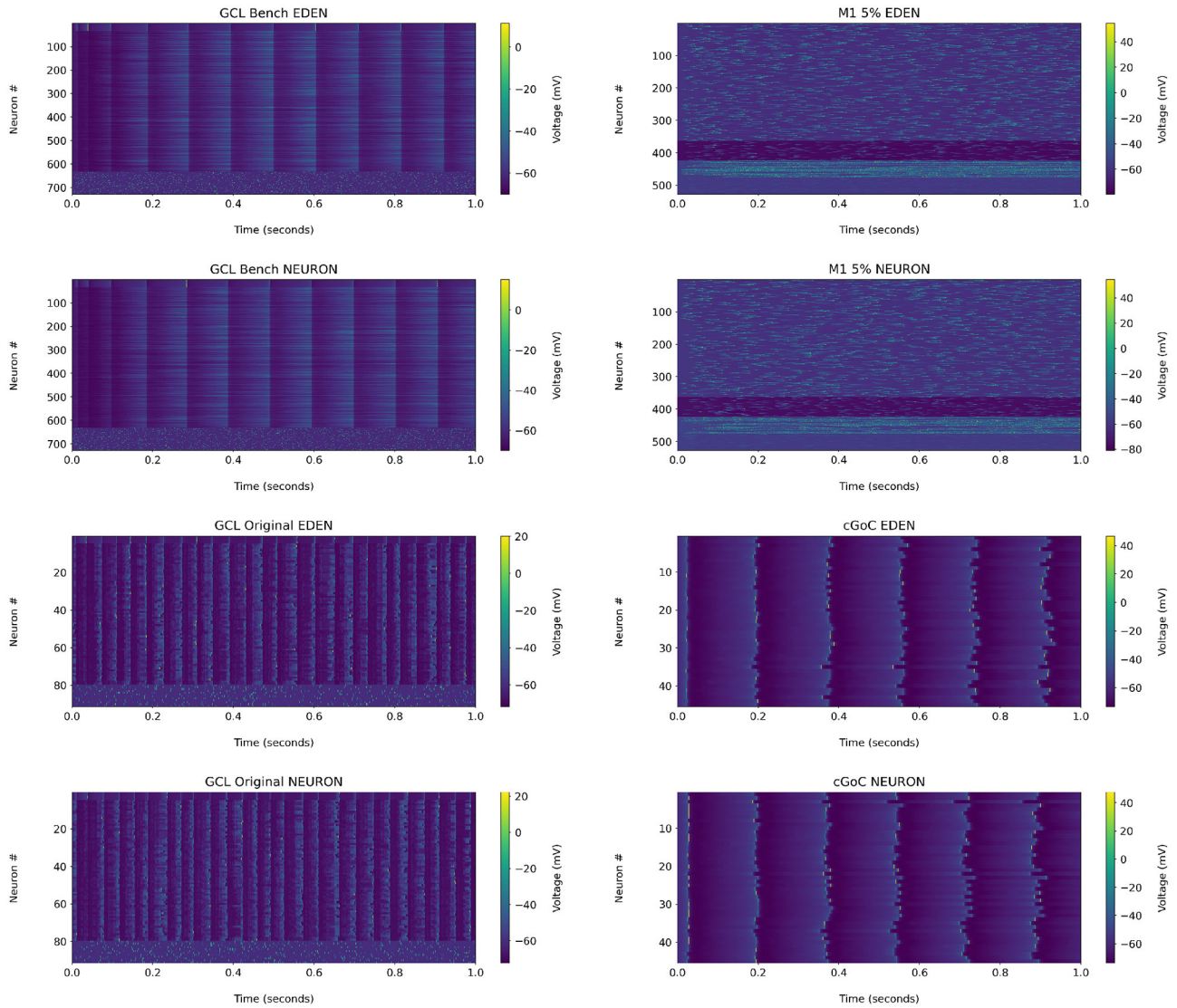


FIGURE 8 | Raster plots for each network used in the performance benchmarks, when run on NEURON and EDEN. Note that the input stimuli are pseudo-randomly generated.

difference between EDEN and NEURON. These neuron models do not share a distinct mechanism type, or other distinguishing commonality that could explain this; nonetheless, we note all these models originate from the Blue Brain Project collection and they showed a low firing rate under the protocol's clamp current. Since these neuron models emitted few spikes, the difference might be specific to the starting phase of regular firing, when induced by DC current.

The full set of results with accuracy metrics and waveform plots for each simulated model is available on Zenodo: https://zenodo.org/record/5526323.

3.1.2. Evaluation on Neural Network Models

To assess EDEN's functionality when simulating networks, the result data from simulating the smaller versions of each network used in the performance benchmark on Section 3.2 were analyzed. Note that the enlarged versions of the GCL and cGOC models should not be used for functional analysis, because they have not been validated by the creators of the original models and they serve only as a computational benchmark.

For reference, the raster plots for the networks analyzed in the following are shown in **Figure 8**. We observe that in the GCL model used for the benchmark, the granule cells did not generate action potentials under either NEURON or EDEN; a closer inspection of the voltage waveforms of these cells indicated that they are over-inhibited by the synapses. Therefore, we chose to apply the analysis on the original, smaller, single-compartment version of the network, which is also discussed in the **Supplementary Material**. The raster plot of this version of the network is also included there.

Since the networks are driven by random stimuli, the results cannot be compared directly as waveforms, but through network activity statistics. We employed the analysis methods proposed by Gutzen et al. (2018), who used them to compare the results generated by the SpiNNaker system and the original floating-point arithmetic based C code, for an Izhikevich cell network.

The measured metrics are: average firing rate (simply number of spikes divided by simulation time), local variation, mean inter-spike interval, correlation values of the binned spike trains with small and large temporal bins (metrics CC and RC respectively), and eigenvalues of the correlation matrix (computed by correlating the exact waveforms, in this work). For each of the metrics except for eigenvalues, the effect size between NEURON and EDEN's results is computed as Cohen's *d*, that is the difference of mean values of the distributions, divided by the pooled standard deviation of the two distributions. The 95% confidence interval for each effect size is calculated using the formula: $\pm 1.96\sqrt{\frac{N1+N2}{N1N2} + \frac{(\text{effectSize})^2}{2(N1+N2-2)}}$

The resulting metrics for each network are shown on **Figure 9**. Presentation is similar to **Figure 10** of the Gutzen et al. (2018) article.

For each of the networks, we observe the following:

- The results for the GCL network, we observe a slightly wider distribution of average firing rates in EDEN's results than in NEURON's, which is however not reflected in the inter-spike interval metric. In contrast with the other two networks, this one exhibits a wide range of neuron-pair correlation coefficients, both in fine time resolution and in rate correlation.
- The results for the M1 network are largely the same. This was expected, since NEURON and EDEN produce very similar results when simulating artificial cells (see also results from various OSB models in the **Supplementary Material**).
- The neurons of the cGOC network exhibit periodic, synchronized spiking. Thus all neurons exhibit the same estimated firing rate and are concentrated around specific values in fine temporal and rate-based correlation. Furthermore, the short-term and rate-based correlation indices are tightly clustered around specific values. There is a slight but clear difference between the simulators on the means of local variation and inter-spike interval; the effect size is very high because the variation in these metrics is very small across the neuron population (see the range of the in the LV and ISI plots).

In many cases, as the effect size is estimated to be low, the confidence interval for the monovariate metric s(firing rate, local variation, inter-spike interval) is determined by the small number of data points in the sample (i.e., neurons). Overall, the quantitative analysis indicates that our simulator succeeds in capturing the characteristics of simulated networks, much like NEURON does.

3.2. Computational Performance Analysis

3.2.1. Overview

Beside flexibility in supported models, another distinguishing characteristic of neural simulators is speed. To evaluate the simulation speed EDEN offers we ran simulations of neural networks available in NeuroML literature, on a recent cost-effective desktop computer. We chose to run published neural networks over synthetic benchmarks, because:

- they have been used in practice, so they are concrete examples of what end users need; and
- existing models are usually the base for newer models, so the insights about the former do remain relevant.

Since the original neural networks were developed with the computing limitations of earlier years, these days they run comfortably in a desktop computer, using a minor fraction of system memory and within just a few minutes per run. (Unfortunately, new network models that do push the limits of present hardware, are still only available in heavily custom setups, that cannot be easily ported to another data format, simulator, or HPC cluster). To evaluate simulation performance for longer simulation run times, and more challenging neural network sizes, we also used enlarged versions of the original neural networks. This was possible, because the original networks were themselves procedurally generated, with parametric distributions of networks and synapses.

The neural networks that were run for performance evaluation are listed in **Table 2**, along with quantitative metrics for each case. Beside these quantitative metrics, there also are substantial qualitative differences between the models. These differences determine both the neural functions of each network, as well as the required computational effort to simulate each one.

3.2.2. Simulated Networks

The neural network models used were the following, sourced from NeuroML-DB:

1. A multi-compartmental extension of the (Maex and Schutter, 1998) Cerebellar Granule Cell Layer (**GCL**) model (NeuroML-DB ID: NMLNT000001).
2. An Izhikevich cell-based, multiscale model of the mouse primary motor cortex (**M1**) (Dura-Bernal et al., 2017) (NeuroML-DB ID: NMLNT001656).
3. A model of the Golgi cell network in the input layer of the cerebellar cortex(**CGoC**), electrically coupled with gap junction (Vervaeke et al., 2010) (NeuroML-DB ID: NMLNT000070).

3.2.2.1. The GCL Network

The GCL network is based on the Maex and Schutter (1998) model for the cerebellar granule cell layer, which includes granule cells, Golgi cells, and mossy-fiber cells. The designers of the GCL network benchmark extended the original GCL model to have multi-compartmental cells; in particular, the axons and parallel fibers of the granule cells are spatially detailed with four compartments per cell, and Golgi cells follow the ball-and-stick model, with 4 compartments per cell. The mossy-fiber cells are stimulated by Poisson randomly firing synapses and stimulate the granule cell population through AMPA and blocking NMDA synapses. The granule cells excite the Golgi cells through AMPA synapses, and the Golgi cells inhibit the granule-cell population through $GABA_A$ synapses. We enlarged

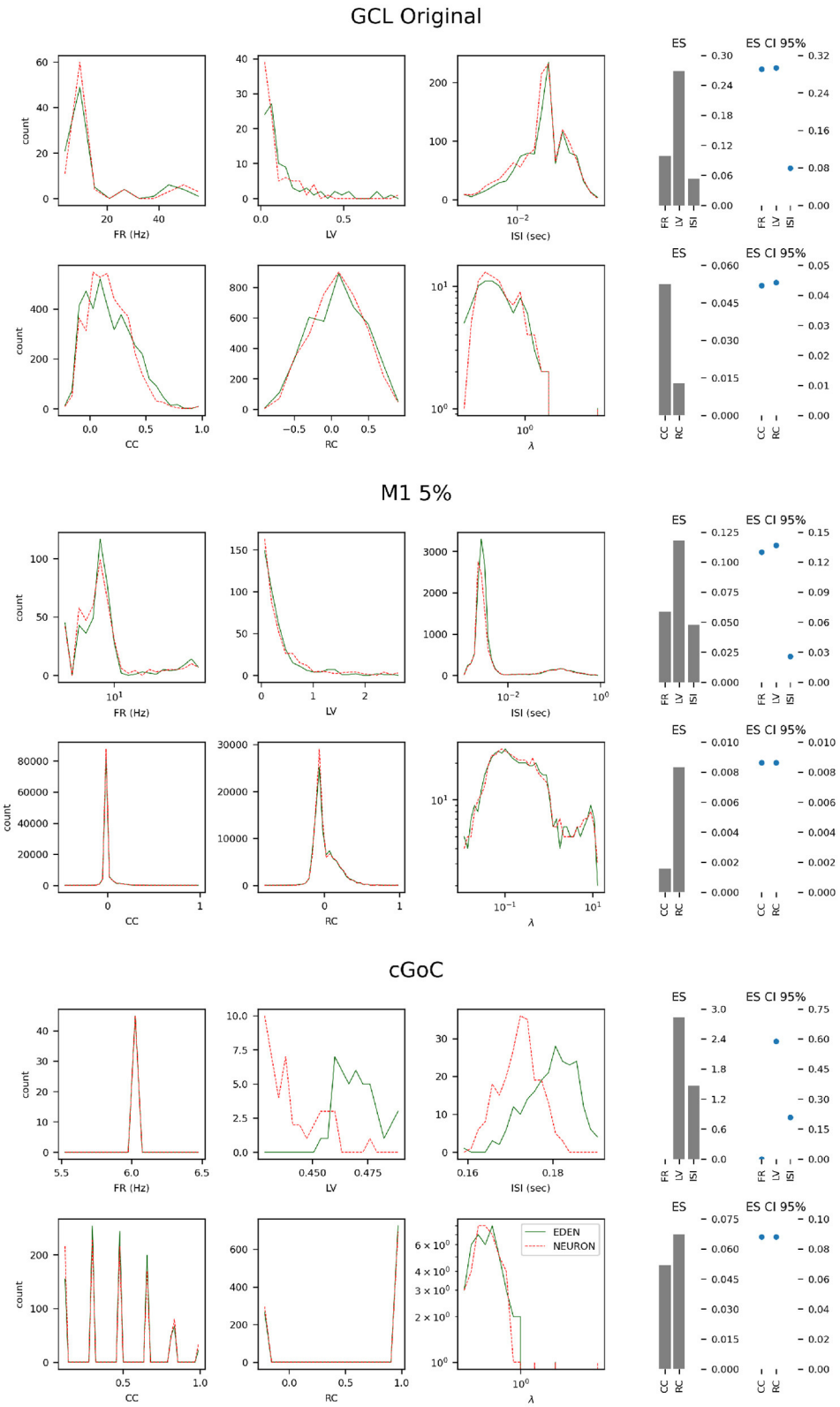


FIGURE 9 | Histograms of neural activity metrics for each network used in the performance benchmarks. FR, firing rate; LV, local variation; ISI, inter-spike interval; CC, short-time firing correlation; RC, rate correlation; (λ), correlation eigenvalue. For each simulated network, the solid green line outlines the distribution of metric values when using EDEN, and the overlaid dashed red line outlines the distribution when using NEURON. On the right, the effect size (ES) and its confidence interval is shown for each applicable metric.

TABLE 2 | The simulated networks used for performance benchmarking.

Simulation	Simulated time (s)	Steps	Compartments per neuron	Neurons	Total compartments	Total synapses
GCL	1	100,000	1 ∼ 4	728	2,624	7,958
GCL x10				7,280	26,240	79,825
M1 5%	1	20,000	1	527	527	15,469
M1 10%				1,065	1,065	61,538
M1 100%				10,734	10,734	5,032,223
CGoC	0.1	40,000	319	45	14,355	472
CGoC x10				450	143,550	5,410

the original GCL network, by multiplying the population size by a factor of 10, and keeping the same per-neuron synapse density for the various projections. Thus, the total number of synapses was also 10 times the original.

3.2.2.2. The M1 Network

The M1 network is an Izhikevich cell-based model of the mouse primary motor cortex, with various groups of cells intertwined across cortical depth. There are 13 groups of cells and four different sets of dynamics parameters among the cells. Each cell is stimulated by an external randomly firing synapse stimulus, and cells interact with each other through excitatory AMPA and NMDA synapses, and inhibitory GABA synapses. All synapses follow the stateless, double-exponential conductance model. This model is rather recent, so in its full size, it is computationally challenging enough to simulate, without enlarging it.

To better investigate performance characteristics, and evaluate performance at a model scale similar to the original GCL and CGoC networks, we generated two smaller versions of the M1 network, at "scale" values of 10 and 5%. Note that the model uses fixed probability connectivity for the various projections between populations, thus the number of synapses grows quadratically with the population size.

3.2.2.3. The CGoC Network

The CGoC network models a small part (0.1 mm^3) of the Golgi cell network, in the input layer of the mouse's cerebellar cortex. It was used in Vervaeke et al. (2010) to investigate the network behavior of Golgi cells, using experimental data. In this network, the neurons communicate with each other solely through gap junctions. Each cell also has 100 excitatory inputs in the form of randomly firing synapses, randomly distributed among apical dendrites.

Gap junctions have rarely been introduced in large network models in the past. This is not because they are absent from tissue, nor because their effects are negligible, but primarily because of their intense computational requirements. The continuous-time interaction between neurons that gap junctions effect requires a large amount of state data to be transferred to simulate each neuron in every step, while spike-based synapses need to only transfer the firing events between neurons, whenever they occur (rarely, compared to the number of simulation steps). As with the GCL network, we also enlarged this network, by making neuron count, synapse count and network volume 10 times the original.

3.2.3. A Note on Numerical Methods & Performance

In the following, computational efficiency is compared between EDEN and NEURON (run under jNeuroML), for the same models. We notice a disparity in per-thread efficiency between EDEN and NEURON, and an immediate question is whether the difference is caused by differences in the simulators' numerical methods.

We noticed that, because of the MOD files, jNeuroML generates in the present version, most membrane mechanisms are simulated with the Forward-Euler integrator, as they are under EDEN as well[6]. The methods EDEN uses in these benchmark are explained in Section 2.5.5. The only clear difference in numerical methods between NEURON and EDEN is that EDEN uses single-precision arithmetic whereas NEURON uses double-precision arithmetic; but this is not enough to explain the observed disparity in simulation speed. Thus, we expect that our simulator's improved performance comes mostly from a more compact control flow, improved data locality, and reduced memory usage (since memory transfers also are a factor) than from pure numerical efficiency.

3.2.4. Benchmark Results

The three neural networks described above—with network sizes, simulation time and time-steps as shown on **Table 2**—were run on a recent desktop PC, and simulation run time was measured in each case. The NEURON simulator was chosen as a baseline to compare simulation speed, because it is the predominant simulator for biophysically detailed, multi-compartmental neuron models. The models were run on NEURON, using both a direct-to-NEURON export of the networks as well as the HOC/MOD code that jNeuroML generates automatically. Although NeuroML2 models can be run on NEURON directly through the jNeuroML tool, there is no published information on the computational efficiency of simulations run through jNeuroML, compared to running hand-written NEURON code for the same model.

[6]With the exception of Markov gates for ion channels. In that specific case, the "sparse" method is employed by jNeuroML, but the kinetic scheme that is present in cGoC cells has too few state variables to affect overall run time compared to using a simpler method.

By running both versions of the model, we compare EDEN's computational efficiency directly against NEURON, and also evaluate experimentally and publish the first data points on the computational efficiency of running neural networks on NEURON through jNeuroML.

All three networks used in the benchmarks were originally generated with a high-level model generation tool; this was neuroConstruct for the GCL and cGoC models, and NetPyNE for the M1 model. This fact also serves as an indication that model creators prefer to focus on the pure aspects of their models, than spend effort on the simulator-specific programming. We checked the implementations of the networks that these tools export for NEURON, and found that the implementations are as efficient as a modeler would reasonably write them to be.

- For the GCL and cGoC models, the HOC and MOD code was generated by neuroConstruct; however, we inspected the generated code and found that it is similar to how the HOC and MOD files are typically written in practice. The only difference is that the HOC statements to create the network are laid out as explicit lines, whereas on manual code loops and procedural (or file-based) generation would have been used to populate the network. However, once NEURON's `run()` command is run, the simulation is controlled by the hard-coded NEURON engine, save for the NMODL mechanisms; whose code, although machine generated, is straightforward and efficient. As explained below, the time to initialize each model is excluded from the measurements.
- The original M1 network was generated at runtime and loaded into NEURON through NetPyNE, which uses the simulator's Python API. The MOD file for the Izhikevich cell model was hand-written and supplied by the model creators, and the Exp2Syn mechanism for synapses is one of NEURON's built-in mechanisms. This use case is thus considered to be how the model is run directly on NEURON.

Although NEURON can employ multi-process parallelism on a network simulation, setting up a simulation for this requires non-trivial, simulator-specific programming code that is difficult to keep free of errors, and possibly changes in the model's MOD files to allow parallel processing. NetPyNE aims to remedy this but parallelizing a NEURON simulation still relies on non-trivial custom programming and care by the model creator. We explored ways to run NEURON in parallel using the existing NeuroML tooling, but none worked correctly for our models. Thus, NEURON was run on a single processing thread for all simulations; this represents the use case of running a NeuroML model on NEURON directly, without extensive NEURON-specific modifications. EDEN was also run on a single thread, allowing a direct comparison of intrinsic computational efficiency of the two simulators (that is, excluding EDEN's performance boost from automatic parallelization).

In this work, in order to focus on the pure computational efficiency of simulating the networks, we excluded the time needed to set up the model and to write result data from our measurements; we only measured the wall-clock time to run the model over the simulated time. EDEN's run time was measured both when using all CPU cores of the PC and when running on a single CPU thread. These two cases reflect different usage scenarios of the simulation workload: the first one occurs when an individual simulation has to be run and the second one when a large batch of simulations has to be run, as a group. The uses and the technical considerations for each case are explained below.

The first case is relevant when a neural network is simulated once, and its behavior is empirically assessed by the experimenter, who adjusts parameters interactively. This takes place in the first exploratory steps of development, when the experimenter is still deciding on the form and type of dynamics of the network. In this case, a single simulation has to be run as quickly as possible, using all available computational resources. Thus, EDEN uses all CPU cores simultaneously to run this simulation.

The second case is relevant after the network's form and model are determined (or candidates for a more extensive evaluation). In this case, the model's properties are explored, by varying its structural and physiological parameters across simulations, and measuring high-level metrics for the behavior of the network (such as type of firing activity and emergent correlations). To that end, a batch of up to thousands of simulations has to be run, in order to explore each individual point of the parameter space. Simulations can then run on a separate CPU thread each, in parallel. Some technical effort is required to distribute the runs of the batch among CPU cores but this technique also allows non-parallel simulators to use multi-core computers effectively. Even so, this kind of parallelism has its limitations. If parameter exploration is performed on a relatively large network, the high memory requirements may prevent launching as many model instances as CPU cores. In that case, it would be better to assign multiple CPU cores per simulation. Likewise, a non-uniform memory hierarchy (common in high-end compute nodes) could even make it more efficient to run fewer models simultaneously, with more cores assigned to each.

For all performance benchmarks, the machine used was a desktop PC, with an Intel i7-8700 3.2 Ghz CPU and 32 GB of 2133 MT/S DDR4 RAM. The CPU has six physical cores and can run up to 12 (hyper)threads simultaneously. The particular CPU was selected to reflect the typical, current-day system available on a researcher's desk–rather than what is available on a supercomputer setting, which requires substantial technical effort to use and is often not available for day-to-day experimentation. The OS used was Ubuntu Linux 18.04. NEURON, EDEN and the code generated by both at runtime were all compiled using the GNU C compiler, version 7.4.

The results for the performance benchmarks are shown in **Table 3**. For each simulation in **Table 2**, the time to run it is shown when running NEURON directly, NEURON through jNeuroML, EDEN on one CPU thread, and EDEN on the whole CPU. The corresponding speedup ratios for EDEN on a single thread and on all threads over NEURON are also shown on the table. **Figure 10** visualizes the relative time to run each simulation with EDEN, using one CPU thread or the whole CPU, against the time to run the same simulation with NEURON.

We observe that EDEN largely outperforms NEURON while running on a single CPU thread, and even more so when the network is simulated across all threads of the CPU. This

TABLE 3 | Measured run time for benchmarks for NEURON on 1 thread, jNeuroML/NEURON, EDEN on 1 thread, and EDEN using all CPU threads, and respective speedup ratios.

Benchmark	NEURON run time (s)	jNML run time (s)	jNML speed ratio	EDEN run time (s)		EDEN speed ratio	
				1 thread	Full node	1 thread	Full node
GCL	145.71	153.03	×0.95	46.55	8.07	×3.13	×18.05
GCL x10	1,828.18	2,758.91	×0.66	756.20	179.54	×2.42	×10.18
M1 5%	13.28	9.20	×1.44	8.00	1.41	×1.66	×9.46
M1 10%	52.99	38.93	×1.36	25.12	3.98	×2.11	×13.32
M1 100%	5,378.23	3,581.39	×1.50	914.17	378.74	×5.88	×14.20
CGoC	152.69	170.36	×0.90	14.64	2.33	×10.43	×65.45
CGoC x10	6,227.36	6,269.13	×0.99	268.75	85.22	×23.17	×73.07

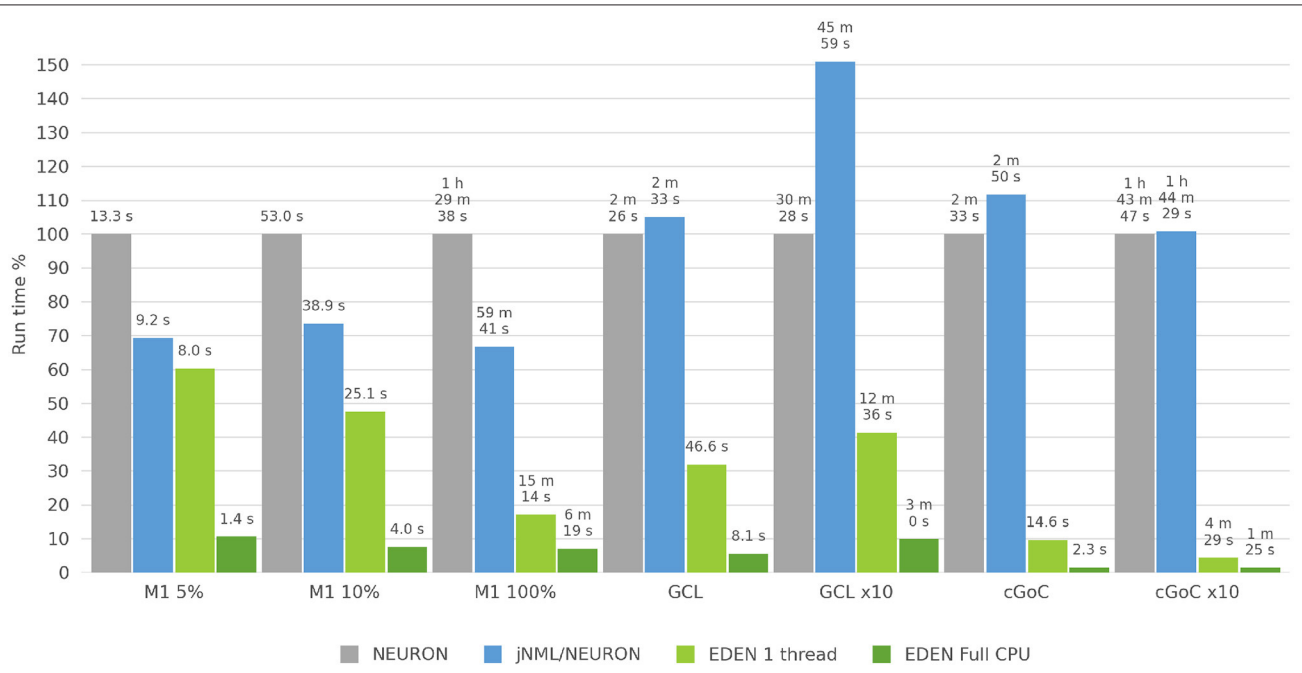


FIGURE 10 | Run time for each neural network considered, for NEURON versus jNeuroML/NEURON and versus EDEN on one CPU thread and on all CPU threads. For each neural network, benchmark the bar height in the chart is normalized against NEURON's run time for that benchmark.

is because EDEN was designed from the start to achieve high computational performance, especially when running complex, biophysically detailed neurons. In the following, we will comment on the performance characteristics demonstrated when running each specific neural network, and reiterate the network's properties that affect computational performance.

The GCL network comprises biophysically detailed cells, with a very small number of compartments per cell. In this case, EDEN generates fully simplified code kernels for each neuron type; the code to simulate each individual compartment is laid out as an explicit, flat sequence of arithmetic operations.

When running the original GCL network, EDEN runs at 3.1 × the speed of NEURON, using one CPU thread. This level of speedup over NEURON applies when running a batch of many small simulations; in which case, each simulation is run on a single CPU thread for best results. By utilizing all six cores of the CPU, simulation speed further improves around six times, for a total of 18 × the speed of NEURON. This shows that when a single simulation has to be run at maximum speed, EDEN can automatically, efficiently parallelize the computational work across multiple processor cores to run faster. For the enlarged version of the network, single-thread speedup using EDEN is less, to 2.4 × the simulation speed of NEURON. Speed improves by using all threads, but the total improvement in speed vs. running NEURON is not as great as when running the smaller, original-size model (×10.18 total, compared to ×27.1 previously). It could be that the processor's data transfer speed decreases with model size, and limits computational throughput; however, the fact that jNeuroML runs significantly slower than NEURON for this network could indicate that there is an inefficiency involved in interpreting the NeuroML2 version of the model. At any rate, this

relative slowdown of the NeuroML-based simulators warrants further investigation.

The M1 network comprises Izhikevich-type artificial cells, with dense synaptic connectivity between the neurons. In this case, each neuron's internal model is one Izhikevich-cell mechanism; EDEN generates a simplified code kernel, that simulates the neuron's internal dynamics and synaptic interaction. When running the 5% version of the network, EDEN on a single CPU thread runs at 1.7 times the speed of NEURON, and using all cores it runs at 9.5 times the speed of NEURON. Running the larger 10% network, these performance ratios increase to 2.1 × and 13.3 ×, respectively. Finally, when running the full-sized version of the network, EDEN on one CPU thread runs at 5.9x the speed of NEURON, and using all cores it runs at 14.2 × the speed of NEURON. For the reduced-size versions of the network, EDEN still runs faster than NEURON, but not by as much as when running the full-sized version. This might be because the amount of computations and data involving these simplified neurons is smaller (also due to the smaller number of synapses per cell, in the M1 network), which increases the effect of parallelization overhead for EDEN. For the full-sized network, EDEN's relative performance improves steadily. Another interesting observation is that all sizes of the M1 network run significantly faster as NeuroML models under jNeuroML/NEURON than as the original NetPyNE/NEURON model. We speculate that this is because of the MOD file describing the neurons; the original hand-written one contains many additional features, calculations and WATCH statements which are not used in this model. Compared to the original MOD file, the one that jNeuroML generates automatically is quite simpler.

Networks solely consisting of point neurons can already be run with high computational performance, on specialized simulators like NEST. However, there is the important class of hybrid SNNs (Lytton and Hines, 2004) that mixes physiologically-detailed and artificial cells according to the focus of each model. Such networks have to be run with general-purpose neural simulators, that support both types of neuron, which then need to run in tandem. By demonstrating a consistent high speedup factor even for artificial-cell networks that are not its main target, EDEN shows that it can run hybrid neural networks at a greatly increased speed, without running into performance problems. For pure artificial-cell networks, EDEN is still relevant for modifications that depart from the commonly supported models, or take a lot of effort to set up on high-performance artificial-cell simulators (e.g., require modifying the simulator's source code to extend model support).

The CGoC network is made up of Golgi cells, which are modeled with hundreds of physiological compartments. Since these cells have too many compartments to apply a flat-code representation per cell type, as was done for the GCL network, EDEN works differently in this case. For each type of cell, the compartments comprising it are grouped according to the set of physiological mechanisms that they contain. This way, one code kernel is generated for simulating each different type of compartment. Then, all compartments of the same type are simulated as a group using a loop over the same code. After computing the internal dynamics for each compartment, the interactions between the compartments, such as the cable equations, are also computed to complete the time-step. We notice that, when running either the original or the 10x-enlarged version of the CGoC network, EDEN exhibits a spectacular increase in simulation speed compared to NEURON. When running the original-sized network, the relative simulation speed over NEURON is 10.4 × using one thread, and 65.5 × using all threads. In wall-clock terms, this means that a simulation that used to take two and a half minutes to run with NEURON, takes 14.6 s with EDEN in batch mode, and 2.3 s with EDEN in single-simulation mode. When running the 10x-enlarged version of the network, the relative simulation speed using NEURON is 23.2 × when using one thread, and 73.1 × when using all threads. In this case, wall-clock run times are 1 h 44 min to run with NEURON, vs. 4 min 29 s with EDEN in batch mode and 1 min 25 s with EDEN in single-simulation mode. The significant improvement in speedup that EDEN exhibits when running the cGoC network vs. the other two networks indicates that the current implementation is best suited for large populations of highly detailed neurons.

4. DISCUSSION

4.1. Current Neural-Simulator Challenges

Through the process of developing EDEN and our involvement with the existing neural-modeling literature, tools and practices, we realized the urgent need for *standards* in brain modeling and *reproducibility* between simulators.

From the perspective of a computer engineer, there is an enormous learning curve in designing simulators for biophysically-detailed neural networks. The technical know-how on handling the differential equations of neural physiology is scattered across past publications and program source code and, even then, is rarely mentioned by name. A modeling standard could help form a compendium of all the mathematical concerns that affect simulator design, and would allow neuroscientists and engineers to co-operate efficiently.

As mentioned in Section 1, when working with highly detailed neural networks, swapping among different simulators during experimentation would take an impractical amount of effort. This is one of the reasons why there are so few inter-simulator comparisons of the same model in *in silico* neuroscience literature and why they usually only concern porting a custom simulation code to, or from, a general-purpose simulator. A standardized, interoperable description for models would remove this major obstacle and enable cross-simulator evaluation. There do exist software that can algorithmically generate neural networks and run them on different simulators [examples are PyNN (Davison et al., 2009), and also neuroConstruct *via* NeuroML export and jNeuroML]. The problem with them is that the model-building "recipes" they support are few and basic. However, model creators often use highly custom methods to construct their networks, which prevents them from using the multi-simulator capability of tools

to save programming effort. A solution to this conundrum may be to use an unambiguous description for generated neural networks, such as NeuroMLv2; then, model creators still have to convert the networks that their custom methods generate to the common description, but multi-simulator capability is much easier to implement since the network to simulate is described explicitly. Still, this approach allows combining all types of network-building recipes with all simulation platforms, without extra programming effort for each combination. The related field of systems biology reveals a success story in the CellML (Cuellar et al., 2003) and SBML (Hucka et al., 2003) standards; however, those standards are still not sufficient for capturing modern networks of multi-compartment neurons.

Another important aspect of upcoming neuroscience projects is *multiscale modeling*; that is, studying a neural structure across multiple levels of modeling detail. Since this often involves many different simulators of different model types, it only becomes practical by adopting extensive standards that capture not only the different models but also the results of the simulation at each level. This is necessary in order to reconcile and investigate the different scales of modeling without writing fully custom, one-off code for each case.

The integrated TVB modeling platform (Sanz Leon et al., 2013) is currently the leading tool for multiscale brain modeling and features a complete methodology for integration of macroscopic neuroimaging data into models. However, this methodology is mostly designed around the specific TVB platform; there is effort to co-simulate with the NEST simulator specifically, but it is still at an ad-hoc, proof-of-concept stage.

Besides standards, we also advocate for a more rigorous *integration* of the various simulators with neuroscientific as well as general (e.g., Python/Jupyter) workflows, which will speed up experimental setup and enable seamless transfer of simulation results across different platforms. This may sound obvious but it is in fact a crucial element for real-world quick adoption and utilization of this ensemble of platforms. NEURON, BRIAN2, GeNN, and Arbor have already caught on to this need; that is why they all natively support a Python interface, alternative to their own custom languages (BRIAN2 itself is Python-native). EDEN already offers such integration through its pyNeuroML-compatible Python bindings[7] and interoperability with the existing NeuroML tooling infrastructure before and after the simulation stage.

Regarding the NeuroML community, it is important to stress the usefulness of providing simulation files along with the published model descriptions. This is important not only to fully record the published experiments but also to be able to reproduce the experiments and cross-validate the results on multiple simulators. To illustrate, we tried to evaluate EDEN on as many NeuroML networks as possible but were only able to find five individual, non-trivial network models in the entire NeuroML-DB—and important simulation parameters such as duration, time-step size, and recording probes were only available in the original code repositories outside NeuroML-DB.

[7]https://pypi.org/project/eden-simulator/

Finally, from an HPC perspective, the large-network simulation needs of modern researchers call for the use of computer clusters. However existing simulators either offer partial support for clusters or require advanced programming from the end user to work. Automatic, complete support for clusters must therefore be a development priority, which the simulator designers are best suited to address. EDEN offers such built-in automation and will continue improving on its performance.

4.2. The EDEN Potential and Next Steps

The evaluation presented makes it abundantly clear that EDEN delivers on its triple mission toward high performance, high model generality and high usability. This first version of EDEN was focused on ensuring that all kinds of NeuroML models are supported, rather than optimizing the performance of a limited subset thereof. Thus, the performance results seen in this work form a minimum guaranteed baseline of performance, on top of which future improvements can boost performance even further.

Even so, we showed that this performance baseline provides, for real-world neural networks drawn from NeuroML-DB, a speedup ratio over NEURON of 2∼23× per CPU thread and 9∼73× in total, on an ordinary desktop PC. We also demonstrated that no technical expertise is required for deploying and parallelizing the simulations of small and large networks alike, which presents a great incentive for the quick adoption of EDEN by the neuroscientific community.

All its achievements notwithstanding, EDEN is far from a concluded simulator. Our future plans involve work in various directions. Below, we enumerate a few crucial ones:

1. Validate further the EDEN architecture through integrating existing, best-in-class code kernels from the community for special cases (Kasap and van Opstal, 2018; Miedema et al., 2020). Characterize performance etc. on various types of neural networks so as to determine further performance margins.
2. Boost the EDEN general-purpose backend by porting it to accelerator hardware, e.g., on GPUs and graph processors. Employ graph-theory methods for problem mapping in order to deploy EDEN on heterogeneous (e.g., CPU-GPU) platforms and reduce communication overheads.
3. Study the structure and communication patterns of spiking neural network models used in practice, and develop sophisticated strategies to map large simulated networks to computer clusters most efficiently.
4. Add further extensions to EDEN for high-end HPC application, such as support for the SONATA data format and for simulation checkpointing.
5. Research and refine innovative numerical integrators, to improve computational parallelism and maintain numerical accuracy on challenges like cable equations and kinetic schemes.
6. Evaluate and propose extensions to EDEN and NeuroML that enable direct interfacing with arbitrary data sources such as video stimuli, simulated environments to

allow training experiments, and dynamic clamps for hybrid experimentation.

5. CONCLUSION

The large scale, fast pace and ample diversity of *in silico* neuroscience necessitates simulation platforms that offer high computational performance alloyed with reproducibility, low complexity in model description and a wide range of supported mechanisms. To those ends, we have developed EDEN, a novel neural simulator that natively supports the entire NeuroML v2 standard, manages the simulation's technical details as well as multi-node and multi-core cluster resources automatically, and offers computational performance without precedent in the scope of general-purpose neural simulators.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The source code of the EDEN simulator is available on GitLab, through the URL: https://gitlab.com/neurocomputing-lab/Inferior_OliveEMC/eden. The benchmark files and scripts to reproduce the figures of the paper are available on Zenodo, accession number 5526323: https://zenodo.org/record/5526323.

AUTHOR CONTRIBUTIONS

SP designed and developed the EDEN simulator and conducted the experiments. HS was the technical manager of the simulator project and designed the experiments. MN provided guidelines for usability and for numerical issues of neural simulation. MN, CS, and DS conceived and supervised the simulator project. All authors contributed to the article, edited and wrote the manuscript, and approved the submitted version.

FUNDING

This research was supported by the European Commission Horizon2020 Framework Programme Projects EXA2PRO (Grant Agreement No. 801015) and EuroEXA (Grant Agreement No. 754337).

ACKNOWLEDGMENTS

The authors would like to thank their colleagues for their insights on GPU acceleration: Max Engelen, Lennart Landsmeer, and furthermore, to thank their students who were the first to use EDEN in the early development stage and provided valuable feedback and use cases in alphabetical order: Naomi Hulst, Hugo Nusselder, and Rocher Smol. This paper was first published as a preprint on arXiv, with the Panagiotou et al. (2021).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.724336/full#supplementary-material

REFERENCES

Akar, N. A., Cumming, B., Karakasis, V., Küsters, A., Klijn, W., Peyser, A., et al. (2019). "Arbor – A morphologically-detailed neural network simulation library for contemporary high-performance computing architectures," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (Pavia), 274–282. doi: 10.1109/EMPDP.2019.8671560

Bhalla, U. S., Bilitch, D. H., and Bower, J. M. (1992). Rallpacks: a set of benchmarks for neuronal simulators. *Trends Neurosci*. 15, 453–458. doi: 10.1016/0166-2236(92)90009-W

Birgiolas, J., Haynes, V., Gleeson, P., Gerkin, R. C., Dietrich, S. W., and Crook, S. M. (2021). Neuroml-db: Sharing and characterizing data-driven neuroscience models described in neuroml. *bioRxiv [Preprint]*. doi: 10.1101/2021.09.11.459920

Blundell, I., Brette, R., Cleland, T. A., Close, T. G., Coca, D., Davison, A. P., et al. (2018). Code generation in computational neuroscience: a review of tools and techniques. *Front. Neuroinformatics* 12:68. doi: 10.3389/fninf.2018.00068

Cannon, R. C., Gleeson, P., Crook, S., Ganapathy, G., Marin, B., Piasini, E., et al. (2014). LEMS: a language for expressing complex biological models in concise and hierarchical form and its use in underpinning NeuroML 2. *Front. Neuroinformatics* 8:79. doi: 10.3389/fninf.2014.00079

Cuellar, A. A., Lloyd, C. M., Nielsen, P. F., Bullivant, D. P., Nickerson, D. P., and Hunter, P. J. (2003). An overview of cellml 1.1, a biological model description language. *Simulation* 79, 740–747. doi: 10.1177/0037549703040939

Davison, A., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). PyNN: a common interface for neuronal network simulators. *Front. Neuroinformatics* 2:11. doi: 10.3389/neuro.11.011.2008

Dura-Bernal, S., Neymotin, S. A., Kerr, C. C., Sivagnanam, S., Majumdar, A., Francis, J. T., et al. (2017). Evolutionary algorithm optimization of biological learning parameters in a biomimetic neuroprosthesis. *IBM J. Res. Dev.* 61, 1–6.14. doi: 10.1147/JRD.2017.2656758

Dura-Bernal, S., Suter, B., Gleeson, P., Cantarelli, M., Quintana, A., Rodriguez, F., et al. (2019). NetPyNE, a tool for data-driven multiscale modeling of brain circuits. *eLife* 8:16. doi: 10.7554/eLife.44494.016

Ekeberg, Ö., and Djurfeldt, M. (2008). MUSIC - multisimulation coordinator: request for comments. *Nat. Prec*. doi: 10.1038/npre.2008.1830.1

Gewaltig, M., and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Gleeson, P., Cantarelli, M., Marin, B., Quintana, A., Earnshaw, M., Sadeh, S., et al. (2019). Open source brain: a collaborative resource for visualizing, analyzing, simulating, and developing standardized models of neurons and circuits. *Neuron* 103, 395–411.e5. doi: 10.1016/j.neuron.2019.05.019

Gleeson, P., Steuber, V., and Silver, R. A. (2007). neuroConstruct: a tool for modeling networks of neurons in 3D space. *Neuron* 54, 219–235. doi: 10.1016/j.neuron.2007.03.025

Gutzen, R., von Papen, M., Trensch, G., Quaglio, P., Grün, S., and Denker, M. (2018). Reproducible neural network simulations: Statistical methods for model validation on the level of network activity data. *Front. Neuroinformatics* 12:90. doi: 10.3389/fninf.2018.00090

Hines, M. (1984). Efficient computation of branched nerve equations. *Int. J. Bio-Med. Comput*. 15, 69–76. doi: 10.1016/0020-7101(84)90008-4

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., et al. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 524–531. doi: 10.1093/bioinformatics/btg015

Kasap, B., and van Opstal, A. J. (2018). Dynamic parallelism for synaptic updating in gpu-accelerated spiking neural network simulations. *Neurocomputing* 302, 55–65. doi: 10.1016/j.neucom.2018.04.007

Kumbhar, P., Hines, M., Fouriaux, J., Ovcharenko, A., King, J., Delalondre, F., et al. (2019). CoreNEURON: an optimized compute engine for the NEURON simulator. *Front. Neuroinformatics* 13:63. doi: 10.3389/fninf.2019.00063

Lytton, W. W. (1996). Optimizing synaptic conductance calculation for network simulations. *Neural Comput*. 8, 501–509. doi: 10.1162/neco.1996.8.3.501

Lytton, W. W., and Hines, M. (2004). "Hybrid neural networks - combining abstract and realistic neural units," in *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol. 2* (San Francisco, CA), 3996–3998. doi: 10.1109/IEMBS.2004.1404116

Maex, R., and Schutter, E. D. (1998). Synchronization of golgi and granule cell firing in a detailed network model of the cerebellar granule cell layer. *J. Neurophysiol*. 80, 2521–2537. doi: 10.1152/jn.1998.80.5.2521

Magalhães, B., and Schürmann, F. (2020). Efficient distributed transposition of large-scale multigraphs and high-cardinality sparse matrices. *arXiv preprint arXiv:2012.06012*. doi: 10.48550/arXiv.2012.06012

McDougal, R. A., Morse, T. M., Carnevele, T., Marenco, L., Wang, R., Migliore, M., et al. (2017). Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience. *J. Comput. Neurosci.* 42, 1–10. doi: 10.1007/s10827-016-0623-7

Miedema, R., Smaragdos, G., Negrello, M., Al-Ars, Z., Müller, M., and Strydis, C. (2020). flexhh: A flexible hardware library for hodgkin-huxley-based neural simulations. *IEEE Access* 8, 121905–121919. doi: 10.1109/ACCESS.2020.3007019

Panagiotou, S., Sidiropoulos, H., Negrello, M., Soudris, D., and Strydis, C. (2021). EDEN: A high-performance, general-purpose, NeuroML-based neural simulator. *arXiv* doi: 10.48550/ARXIV.2106.06752

Raikov, I., Cannon, R., Clewley, R., Cornelis, H., Davison, A., De Schutter, E., et al. (2011). Nineml: the network interchange for neuroscience modeling language. *BMC Neurosci*. 12:P330. doi: 10.1186/1471-2202-12-S1-P330

Sanz Leon, P., Knock, S., Woodman, M., Domide, L., Mersmann, J., McIntosh, A., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinformatics* 7:10. doi: 10.3389/fninf.2013.00010

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8:e47314. doi: 10.7554/eLife.47314

Vervaeke, K., Lőrincz, A., Gleeson, P., Farinella, M., Nusser, Z., and Silver, R. A. (2010). Rapid desynchronization of an electrically coupled interneuron network with sparse excitatory synaptic input. *Neuron* 67, 435–451. doi: 10.1016/j.neuron.2010.06.028

Vlag, M. A. v. d., Smaragdos, G., Al-Ars, Z., and Strydis, C. (2019). Exploring complex brain-simulation workloads on multi-GPU deployments. *ACM Trans. Archit. Code Optim*. 16, 1–25. doi: 10.1145/3371235

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. *Sci. Rep*. 6:18854. doi: 10.1038/srep18854

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 27 May 2022
doi: 10.3389/fncom.2022.885207

Exploring Parameter and Hyper-Parameter Spaces of Neuroscience Models on High Performance Computers With Learning to Learn

Alper Yegenoglu[1,2], Anand Subramoney[3], Thorsten Hater[1], Cristian Jimenez-Romero[1], Wouter Klijn[1], Aarón Pérez Martín[1], Michiel van der Vlag[1], Michael Herty[2], Abigail Morrison[1,4,5] and Sandra Diaz-Pier[1]*

[1] Simulation and Data Lab Neuroscience, Jülich Supercomputing Centre (JSC), Institute for Advanced Simulation, JARA, Forschungszentrum Jülich GmbH, Jülich, Germany, [2] Department of Mathematics, Institute of Geometry and Applied Mathematics, RWTH Aachen University, Aachen, Germany, [3] Institute of Neural Computation, Ruhr University Bochum, Bochum, Germany, [4] Institute of Neuroscience and Medicine (INM-6), Institute for Advanced Simulation (IAS-6), JARA BRAIN Institute I, Jülich Research Centre, Jülich, Germany, [5] Computer Science 3-Software Engineering, RWTH Aachen University, Aachen, Germany

OPEN ACCESS

Edited by:
Kelly Shen,
Simon Fraser University, Canada

Reviewed by:
Sora A. N.,
Ewha Womans University, South Korea
Mantas Mikaitis,
The University of Manchester, United Kingdom

***Correspondence:**
Alper Yegenoglu
a.yegenoglu@fz-juelich.de

Received: *27 February 2022*
Accepted: *13 April 2022*
Published: *27 May 2022*

Citation:
Yegenoglu A, Subramoney A, Hater T, Jimenez-Romero C, Klijn W, Pérez Martín A, van der Vlag M, Herty M, Morrison A and Diaz-Pier S (2022) Exploring Parameter and Hyper-Parameter Spaces of Neuroscience Models on High Performance Computers With Learning to Learn. Front. Comput. Neurosci. 16:885207. doi: 10.3389/fncom.2022.885207

Neuroscience models commonly have a high number of degrees of freedom and only specific regions within the parameter space are able to produce dynamics of interest. This makes the development of tools and strategies to efficiently find these regions of high importance to advance brain research. Exploring the high dimensional parameter space using numerical simulations has been a frequently used technique in the last years in many areas of computational neuroscience. Today, high performance computing (HPC) can provide a powerful infrastructure to speed up explorations and increase our general understanding of the behavior of the model in reasonable times. Learning to learn (L2L) is a well-known concept in machine learning (ML) and a specific method for acquiring constraints to improve learning performance. This concept can be decomposed into a two loop optimization process where the target of optimization can consist of any program such as an artificial neural network, a spiking network, a single cell model, or a whole brain simulation. In this work, we present L2L as an easy to use and flexible framework to perform parameter and hyper-parameter space exploration of neuroscience models on HPC infrastructure. Learning to learn is an implementation of the L2L concept written in Python. This open-source software allows several instances of an optimization target to be executed with different parameters in an embarrassingly parallel fashion on HPC. L2L provides a set of built-in optimizer algorithms, which make adaptive and efficient exploration of parameter spaces possible. Different from other optimization toolboxes, L2L provides maximum flexibility for the way the optimization target can be executed. In this paper, we show a variety of examples of neuroscience models being optimized within the L2L framework to execute different types of tasks. The tasks used to illustrate the concept go from reproducing empirical data to learning how to solve a

problem in a dynamic environment. We particularly focus on simulations with models ranging from the single cell to the whole brain and using a variety of simulation engines like NEST, Arbor, TVB, OpenAIGym, and NetLogo.

Keywords: simulation, meta learning, hyper-parameter optimization, high performance computing, connectivity generation, parameter exploration

1. INTRODUCTION

An essential common tool to most efforts around brain research is the use of algorithms for analysis and simulation. Specialists have developed a large variety of tools that typically rely on many parameters in order to produce the desired results. Finding an appropriate configuration of parameters is a highly non-trivial task that usually requires both experience and the patience to comprehensively explore the complex relationships between inputs and outputs. This problem is common to all input and output formats, as they differ in their type such as images, continuous or discrete signals, experimental data, spiking activity, functional connectivity, etc. In this article, we focus on parameter specification for simulation.

In order to address this problem, we present a flexible tool for parameter optimization: L2L. Initially inspired by the learning to learn (L2L) concept in the machine learning (ML) community, the L2L framework is an open-source Python tool[1] that can be used to optimize different workloads. The flexibility of the framework allows the user to set the target of optimization to be a model which can be executed either from Python or the command line. The optimization target can also be adaptive and capable of learning, providing a natural way to carry out hyper-parameter optimization. The L2L framework can be used in local computers as well as on clusters and high performance computing (HPC) infrastructure.

This manuscript is structured as follows. First, we provide a quick overview on the state of the art for optimization methods and highlight the main differences between those tools and the L2L framework. In Section 2, we provide an overview of the framework's architecture, its implementation, and the way it can be used and extended. We then demonstrate its effectiveness on a variety of use cases focused on neuroscience simulation at different scales (Section 3).

1.1. State of the Art

In the field of ML, the concept of L2L (c.f. Section 2.1) has been well studied. The L2L concept can be decomposed into two components: (a) an inner loop where a program to be optimized, here named the optimizee, executes specific tasks and returns a measure of how well it performs, called the fitness, and (b) an outer loop where an optimizer searches for generalized optimizee parameters (hyper-parameters) that improve the optimizee's performance over distinct tasks measured by the fitness function. The fitness function is different for each model and tightly linked to the expected transitions in its dynamics. The optimizee can consist of any program such as an artificial neural network, a spiking network, a single cell model, or a whole brain simulation using rate models. In a recent work, Andrychowicz et al. (2016) proposed using long short term memory network (LSTM) with access to the top-level gradients to produce the weight updates for the task LSTM. The main idea is to replace the gradient descent optimizer of the optimizee with an LSTM as an optimizer. In this case, the weights of the inner loop network are treated as the hyper-parameters and trained/learned in the outer loop, while being kept fixed in the inner loop. Based on the work of Andrychowicz et al. (2016) and Ravi and Larochelle (2017) modified the optimization scheme so that the test error can be incorporated into the optimization step. Thus, the optimization can be executed in fewer steps which leads to fewer unrollings of the LSTMs and a reduction of the computational burden. By representing the learning updates of the classifier within the hidden state of the outer-loop optimizer network, the authors acquire a good initialization for the parameters of the inner-loop learner and for further update steps.

For feed-forward networks, Model Agnostic Meta-Learning (MAML) was introduced by Finn et al. (2017). MAML can learn initial parameters for a base-model solving inner-loop-level task. After a few steps of optimization with gradient descent, the base-model can generalize well on the validation set, which is the related data seen for the first time from the same class as the training set. The method can be applied to a vast set of learning problems since the learning itself is agnostic to the inner-loop model. Finn and Levine (2017) showed that learning the initialization combined with gradient updates was as powerful as L2L using a recurrent network. Several extensions have been proposed to enhance the performance of the learning and computation time (Finn et al., 2018, 2019). For example, Li et al. (2017) introduce META-SGD, a stochastic gradient optimization method that not only learns the parameter initialization but also the gradient update of the base-model optimization. However, Antoniou et al. (2018) list several issues found with MAML, such as training instabilities, due to repeated application of backpropagation through the same network multiple times which leads to gradient issues. This leads to a performance drop in learning and computational overhead. A gradient-free version of MAML was proposed by Song et al. (2019) using evolution strategies to replace the second-order backpropagation used in MAML. A framework that is model agnostic but does not depend on calculating gradients or backpropagating through networks and is not limited to a single optimization algorithm would be highly desirable, especially to address the needs of highly interdisciplinary fields such as neuroscience.

Cao et al. (2019) utilize particle swarm optimization (Kennedy and Eberhart, 1995) to train a meta-optimizer that learns both point-based and population-based optimization algorithms in a continuous manner. The authors apply a set of LSTMs to train and learn the update formula for a

[1] https://github.com/Meta-optimization/L2L

population of samples. Their learning is based on two attention mechanisms, the feature-level ("intra-particle") and sample-level ("inter-particle") attentions. The intra-particle module reweights every feature based on the hidden state of the corresponding *i*-th LSTM, whereas the inter-particle attention module learns in the update step of the actual particle information from the previous already updated particles.

Similarly, Jaderberg et al. (2017) use a parallel population-based approach and random search to optimize the hyper-parameters of neural networks. They randomly sample the initialization of the network parameters and hyper-parameters and every training run is evaluated asynchronously. If a network is underperforming, it is replaced by a more successful network. Furthermore, by perturbing the hyper-parameters of the replacing network the search space is expanded. Neural architecture search (Zoph and Le, 2016) and related methods have been shown to be very useful in choosing network architectures for various tasks. A random search was shown to be surprisingly effective for hyper-parameter searches for a wide variety of tasks (Bergstra and Bengio, 2012). Many of the automated hyper-parameter searches also fall under the category of Automated Machine Learning or AutoML (Hutter et al., 2019; He et al., 2021).

In the area of computational neuroscience, BluePyOpt (Van Geit et al., 2016) has represented a robust solution to address optimization problems. Even if it was originally meant to support the optimization of single cell dynamics, BluePyOpt is also able to optimize models at other scales. It makes use of DEAP (Fortin et al., 2012) for the optimization algorithms and of SCOOP (Hold-Geoffroy et al., 2014) to provide parallelization. The target of optimization in BluePyOpt is also quite flexible, it can be any simulator that can be called from Python. This framework can also be used in different infrastructures, from laptops to clusters. However, the framework only allows the execution of optimization targets written in Python.

Deep Learning compatible spiking network libraries, such as NengoDL (Rasmussen, 2018) or Norse (Pehle and Pedersen, 2021), are getting more popular. They are based on modern tensor libraries and can be executed on GPUs which can speed up the simulations. Although these libraries do not focus on meta-learning they are interesting for solving ML tasks using spiking neural networks (SNN). They can be used to quickly learn the tasks while the hyper-parameters of the SNNs can be optimized in an outer loop.

The L2L framework offers a flexible way to optimize and explore hyper-parameter spaces. Due to its interface, the optimization targets are not restricted to executables with a Python interface offering the possibility to optimize models written in different programming languages. In our work, we focus on neuroscientific use cases. The framework, however, is available for a variety of simulations in different scientific domains. Furthermore, the framework is agnostic to the inner loop models and thus allows for different types of optimization techniques in the outer loop. Most of the optimizers adapt population-based computational algorithms, which enable parallel executions of optimizees (see Section 3). This helps to optimize for a vast range of parameter ranges. The error or rather fitness of the inner loop on the absolved tasks is included in the optimization step to update the parameters. Optimizers such as the genetic algorithm or ensemble Kalman filter (EnKF) use the fitness in order to rank the individuals and replace underperfoming individuals with more successful ones (e.g., see Section 3.1).

2. METHODS

2.1. Concept of L2L

Learning to learn or meta-learning is a technique to induce learning from experience (Thrun and Pratt, 2012). The L2L process consists of two loops, the inner and outer loop (**Figure 1**). In the inner loop, an algorithm with learning capabilities (e.g., an artificial or SNN, a single cell model or a whole brain simulation using rate models) is executed on a specific task T from a family $\mathcal{F}$ of tasks.

Tasks can range from classification (e.g., MNIST; LeCun et al., 2010, see Section 3.1), to identifying parameter regimes that result in specific network dynamics (Sections 3.2, 3.4) or training agents to autonomously solve optimization problems (Sections 3.3, 3.5).

The performance of the algorithm over tasks is evaluated with a specifically designed fitness function, which produces a fitness value f or a fitness vector $\mathbf{f}$. The function is, in general, different for every model but closely connected to the task itself. Parameters and hyper-parameters, together with the fitness value of the optimizee are sent to the outer loop. Different optimization techniques, such as evolutionary algorithms, filtering methods or gradient descent, can be utilized to optimize the hyper-parameters in order to improve the performance of the optimizee. Afterward, the hyper-parameters are fed back into the algorithm and a new iteration (i.e., a new generation) is invoked. It is important to note that from a technical point of view, the optimizee acts as an orchestrator of the inner loop. Each optimizee executes a simulation. Borrowing the terminology from evolutionary algorithms, the parameter set which is optimized is called an individual. The optimizee accepts (hyper-)parameters from the outer loop and starts the inner loop process to execute the simulation. Last, it calculates the fitness and transmits everything to the optimizer.

2.2. Parallel Executions in the L2L Framework

In L2L, the optimizers apply population based methods which enable simulations to be run in an embarrassingly parallel fashion. Each individual is initialized independently. They can be easily distributed on several computing nodes and thus can exploit HPC systems. The L2L framework supports the message passing interface (MPI) over several nodes and multi-threading per node. The number of nodes and cores can be set in the beginning of the run and the L2L framework will automatically take care of the distribution and collection of results. Section 2.3 explains in detail how to set up a simulation run in L2L.

2.3. Workflow Description

Listing 1 | Template script to start a L2L run. The optimizee, optimizer are defined. The experiment class is managing the run.

```
1  from l2l.utils.experiment import Experiment
2  from l2l.optimizees.optimizee import Optimizee, OptimizeeParameters
3  from l2l.optimizers.optimizer import Optimizer, OptimizerParameters
4
5  experiment = Experiment(root_dir_path='/home/user/L2L/results')
6   jube_params = {"exec": "srun -n 1 -c 8 --exclusive python"}
7   traj, all_jube_params = experiment.prepare_experiment(name='L2L-Run',
8                                                         log_stdout=True,
9                                                         jube_parameter=jube_params)
10
11 ## Inner loop simulator
12 # Optimizee class
13 optimizee = Optimizee(traj)
14 optimizee_parameters = OptimizeeParameters()
15
16 ## Outer loop optimizer initialization
17 optimizer_parameters = OptimizerParameters()
18 optimizer = Optimizer(traj,
19                       optimizee_prepare=optimizee.create_individual,
20                       fitness_weights=(1.0,),
21                       optimizee_bounding_func=optimizee.bounding_func,
22                       parameters=optimizer_parameters)
23
24 experiment.run_experiment(optimizee=optimizee,
25                           optimizee_parameters=optimizee_parameters,
26                           optimizer=optimizer,
27                           optimizer_parameters=optimizer_parameters)
28 experiment.end_experiment(optimizer)
```

In L2L, the user has to work on two main files. The first file is the **run script**, which invokes the whole L2L two loop run. The second file is the **optimizee**, which operates the simulation in the inner loop.

In the run script, the user configures hardware-related settings, e.g., if the run is executed on a local computer or on an HPC. Furthermore, the optimizee and optimizer and their parameter options have to be set. An example code template to start the whole L2L run is shown in **Listing 1**. Lines 1-3 import the necessary modules, i.e., the experiment, optimizee, and the optimizer. Of course, in the real run, the names of the modules and classes have to be adapted to their respective class names, for simplicity, we call them here optimizee and optimizer. The **experiment** class manages the run. In line 5, the results path is set in the constructor of the class. The experiment method `prepare_experiment` in line 7 prepares the run. It accepts the name of the run, whether logging should be enabled, and the Juelich Benchmarking Environment (JUBE; Speck et al., 2021) parameters. In L2L, JUBE's functionality was stripped down to submit and manage parallel jobs on HPCs and interact with the job management system SLURM (Yoo et al., 2003). The execution directives for the HPC jobs can be seen in line 6. Here, `exec` is the indicator command to invoke a run on a supercomputer, followed by a `srun` directive for SLURM. In the example, one task (`-n 1`) should be run on 8 cores (`-c 8`). Optimizees and optimizers run as Python executables, which is why the `python` command is needed here. If a local run is desired, just the Python command is sufficient, i.e., `"exec":"python."` Internally, JUBE creates a job script and passes it to SLURM, which then executes the parallel optimizees and the optimizer. JUBE accepts many more commands for SLURM, but elaborating on all options would go beyond the scope of this work; see the SLURM documentation[2] for a list of executives. The run script can be executed either as a batch script or as an interactive job on an HPC.

The **optimizee** is defined in line 13 and requires only the trajectory `traj`. The trajectory, modeled after PyPet's trajectory[3], is a class that holds the history of the parameter space exploration and the results from each execution and the parameters to be explored. `OptimizeeParameters` is a Python `namedtuple` object, which accepts the parameters of the optimizee. For the optimizee, the `namedtuple` appears as a parameter object and can be accessed as a class variable, i.e., as `parameters.name`. The optimizee has access to the trajectory and the parameters object.

[2]https://slurm.schedmd.com/
[3]https://github.com/SmokinCaterpillar/pypet

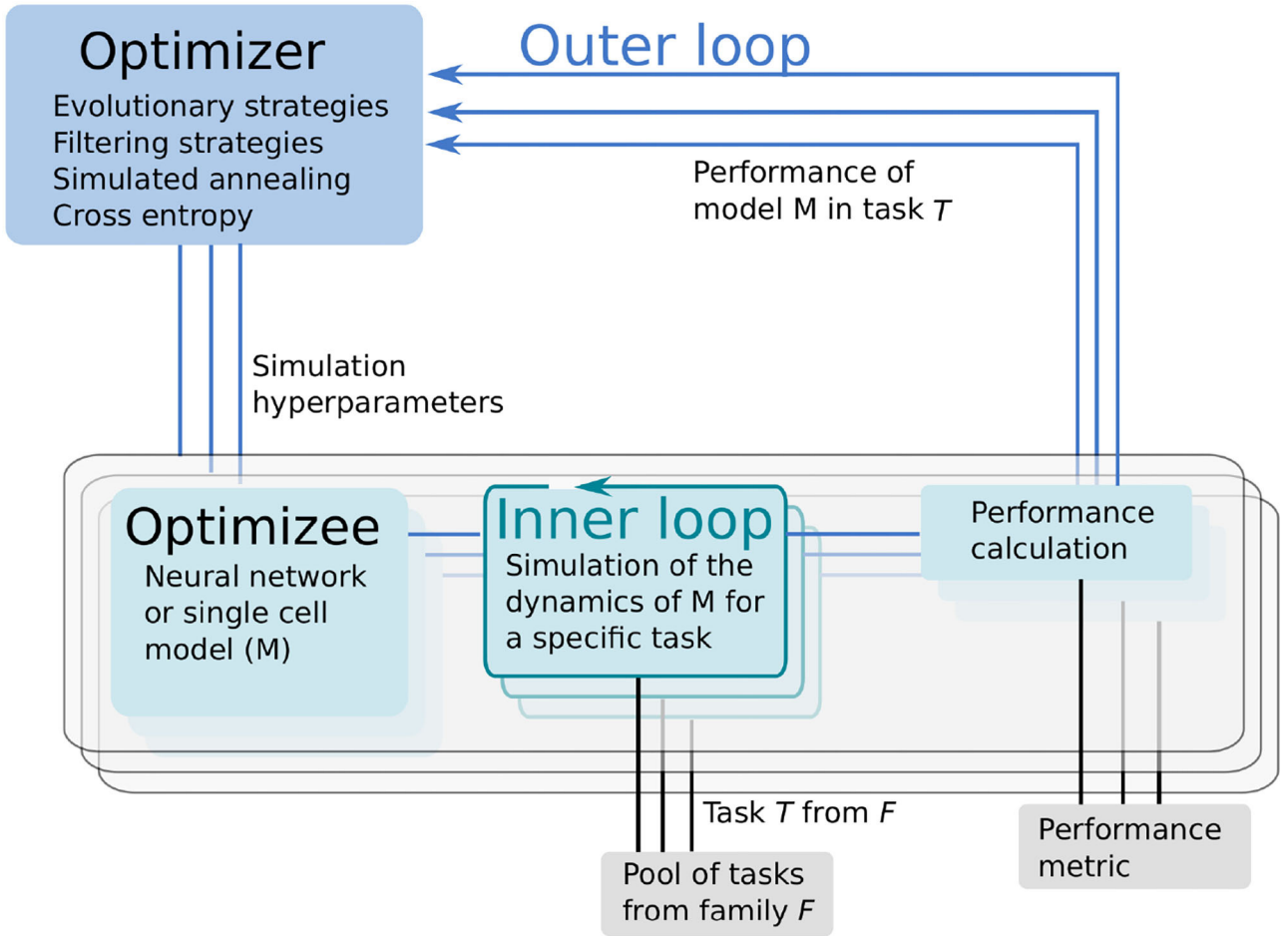


FIGURE 1 | Learning to learn (L2L) consists of two loops. In the inner loop, the optimizee, an algorithm with learning capabilities is trained on a family of tasks. A fitness function evaluates the performance of the algorithm. The (hyper-) parameters and the fitness value of the algorithm are sent to the optimizer in the outer loop. Several optimization methods are available to optimize the parameters, which are fed back to the optimizee and the algorithm.

In the optimizee, three main functions have to be implemented.

1. The function `create_individual()` defines the individual. Here, the parameters which are going to be optimized need to be initialized and returned as a Python dictionary.
2. `simulate()` is the main method to invoke the simulation. The L2L framework is quite flexible about the simulation in the inner loop. It is agnostic with regards to the application carrying out the simulation and only requires that a fitness value or fitness vector is returned.
3. `bounding_func()` is a function that clips parameters before and after the optimization to defined ranges. For example, in an SNN, it is necessary that delays are strictly positive and greater than zero. The function is applied only on parameters that are defined in `create_individual()`.

Similarly, the **optimizer** is created in line 18. It requires the optimizer parameters (line 17) and the method `optimizee.create_individual`, and if available, the bounding function `optimizee.bounding_func`. Additionally, a tuple of weights (`fitness_weights`, here $(1.0,)$) can be given, which weights the optimizee fitness by multiplying those values with the fitness itself. For example, in the case of a two-dimensional fitness vector, a tuple of $(1.0, 0.5)$ would weigh the first fitness fully and the second one only half. Most of the optimizers in the L2L framework perform fitness maximization, but if minimization is required, then it suffices to flip the sign of the fitness function that would be used for maximization. Several optimization techniques are available in the framework, such as cross-entropy, genetic algorithm (GA), evolutionary strategies (Salimans et al., 2017), gradient descent, grid-search, ensemble Kalman Filter (EnKF; Iglesias et al., 2013) natural evolution strategies (Wierstra et al., 2014), parallel tempering, and simulated annealing. The results of the optimizations are automatically saved in a user specified results folder as Python binary files; however, users can store result files from within the optimizee in any format they wish.

The method `run_experiment` (line 24) requires that the optimizee and the optimizer and their parameters have to be defined. Finally, the `end_experiment` method is needed to end the simulation and to stop any logging processes.

3. RESULTS

In this section, we present the results of using L2L to optimize the parameters for a variety of simulation use cases. Every task is executed with a different set of simulation tools, and the interfaces with the simulators also differ between use cases. We present here 5 use cases. Please see the **Supplementary Material** for an additional use case where hyper-parameters are also optimized. A GitHub repository with instructions to run the provided use cases can be found at https://github.com/Meta-optimization/L2L/tree/frontiers_submission.

3.1. Use Case 1: Digit Classification With NEST

The first use case describes digit classification with an SNN implemented in the NEST simulator (Gewaltig and Diesmann, 2007). The SNN is designed as a reservoir, i.e., a liquid state machine (LSM, Maass et al., 2002). The network consists of an input encoding layer, a recurrent reservoir, and an output layer as shown in **Figure 2**. The weights between the reservoir and the output layer are optimized to maximize the classification accuracy.

3.1.1. Description of the Simulation Tool

NEST is a simulator for SNN models. Its primary design focus is the efficiency and accurate simulation of point neuron models, in which the morphology of a neuron is abstracted into a single iso-potential compartment; axons and dendrites have no physical extent. Since NEST supports parallelization with MPI and multi-threading and exhibits excellent scalability, simulations can either be executed on local machines or efficiently scaled up to large scale runs on HPCs (Jordan et al., 2018). Our experiments were conducted on the HDF-ML cluster of the Jülich Supercomputing Center using NEST 3.1 (Deepu et al., 2021).

3.1.2. Optimizee: Spiking Reservoir Model

The network consists of three populations of leaky integrate-and-fire (LIF) neurons, the encoder, the reservoir, and the output; see **Figure 2**. The input to the network is the set of MNIST digits, encoded into firing rates; the firing rates are proportional to the intensity of the pixels from 0 to 255 mapped between $[1, 100]$ Hz. A total of 768 excitatory neurons receive input from a pixel of the image in a one-to-one connection. The reservoir has 1,600 excitatory and 400 inhibitory neurons, while the output has a population of 12 neurons (10 excitatory (red), 2 inhibitory (blue)) per digit. The connections in the reservoir are randomly connected but limited to a maximal outdegree of 6% and 8% for each excitatory and inhibitory neuron. In this setting, we focused explicitly on three digits of the dataset (0 to 2), thus having three output clusters. Each excitatory neuron receives a maximal indegree of 640 connections and each inhibitory neuron receives an indegree of maximal 460 connections from the reservoir. This results in 28,800 ($= 800 \times 12 \times 3$) connections in total. The neurons within an output are recurrently connected, while the output clusters do not have connections to each other. If an input is not presented, the network exhibits low spiking activity in all three parts. The whole network is constructed in the `create_individual` function. The connection weights are sampled from a normal distribution with $\mu = 70$ and $\sigma = 50$ for the excitatory neurons and $\mu = -90$ and $\sigma = 50$ for the inhibitory neurons.

In the simulation (`simulate` function), a small batch of 10 different numbers from the same digit is presented to the network for 500 ms per image as spike trains. Additionally, each neuron in the network receives background Poissonian noise with a mean firing rate of ≈ 5 spikes/s to always maintain a low activity within the reservoir.

Before any image is presented, there is a warming up simulation phase lasting for 100 ms in order to decay all neuron parameters to their resting values. Likewise, between every image, there is a cooling period of 200 ms where no input is shown. After the simulation is run, the output with the highest spike activity indicates the number of the presented digit.

3.1.3. Fitness Metric

In the output, we acquire the firing rates of all clusters and apply the softmax function

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_k e^{x_k}},$$

where $\sigma : \mathbb{R}^k \to [0, 1]^k$ and $\mathbf{x} = (x_0, x_1, \ldots x_k) \in \mathbb{R}^k, j = 1, \ldots, k$ is the vector of firing rates.

We take the highest value, which indicates the digit the network classified. Since every image in the dataset has a label, we can calculate the loss by applying the mean squared error function to the corresponding label:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{1}$$

with y_i the label and $\hat{y}_i$ the predicted output, encoded as one-hot vectors with a non-zero entry corresponding to the position of the label. As the optimizer used in the outer loop for this use case is the ensemble Kalman filter, which minimizes the distance between the model output and the training label, we define the fitness function as $f = 1 - \mathcal{L}$ and use it in order to rank individuals (see next Section 3.1.4). After each presentation of a digit, the fitness and the softmax model output are sent to the optimizer.

3.1.4. Optimizer: EnKF

The ensemble Kalman filter (Iglesias et al., 2013) is the optimization technique we use to update the weights between the reservoir and the output, as described in Yegenoglu et al. (2020). Before the optimization, they are normalized to be in the range of $[0, 1]$. The weights from the reservoir to the output are concatenated to construct one individual. In total, 98 individuals go into the optimization. Each individual has 28,800 weights. To specify in terms of the EnKF setting, the set of ensembles are the network weights, the observations are the softmax model outputs. In Yegenoglu et al. (2020), it was shown that around 100 ensembles are required to reach at least chance level on the MNIST dataset. However, the experiments

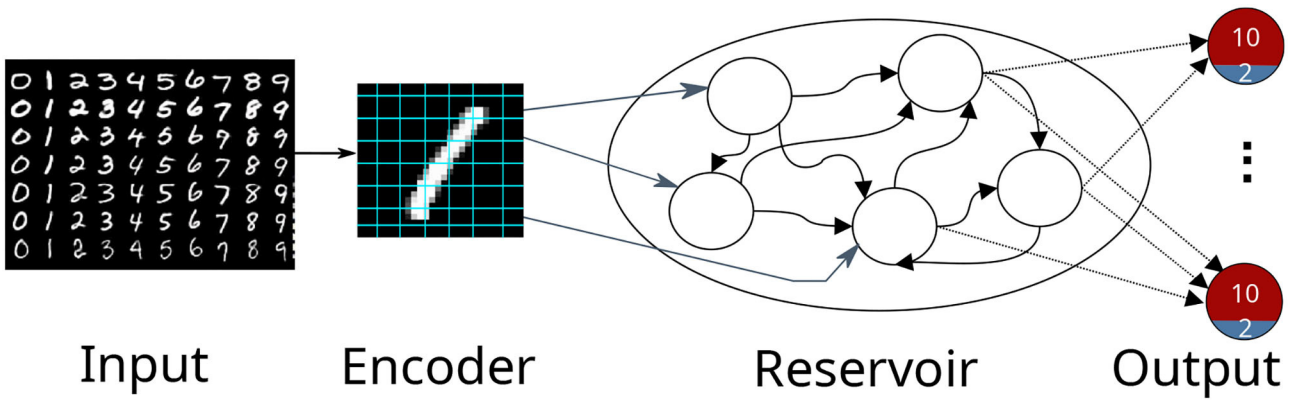


FIGURE 2 | A schematic view of a reservoir network classifying the MNIST dataset. The input image is encoded into firing rates and fed afterward into the reservoir. The output consists of 10 excitatory neuron depicted in red and 2 inhibitory neurons depicted in blue. The highest activity at the output indicates the presented digit.

were conducted using convolutional neural networks tested with harsh conditions such as poor weight initialization and different activation functions. Due to long simulation times, we limited the number of ensembles in this case. Future work will investigate a more variable ensemble size. We implemented a slight modification of the EnKF in which poorly performing individuals can be replaced by the best individuals. The fitness is used to rank the individuals and replace the worst n individuals with m best ones. Furthermore, we add random values drawn from a normal distribution to the replacing individuals in order to increase the search space for the parameters and to find different and possibly better solutions. We set n and m to be 10% of the corresponding individuals. One hyper-parameter of the EnKF is γ (set to $\gamma = 0.5$), it can be compared to the effect of the learning rate in stochastic gradient descent. A lower γ may lead to a faster convergence but also has the risk of overshooting minima. In contrast, a higher γ is slower to converge or can get trapped in minima. Since the simulations take a relatively long time to finish, we cannot train on the whole dataset (see next Section 3.1.5) In this setting, the EnKF with the implemented additions is a suitable optimization technique, because it is able to quickly converge to minima and provide satisfactory results.

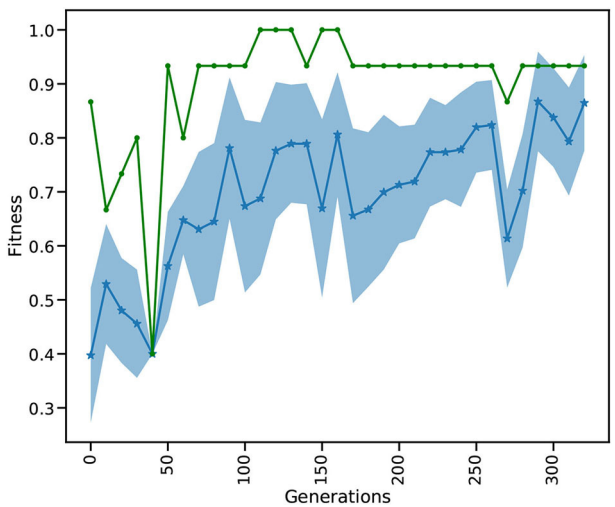


FIGURE 3 | Every tenth iteration the reservoir is tested on a small part of the MNIST test data. The blue dotted line shows the mean fitness and the shaded area is the standard deviation of all individuals. The green line depicts the best fitness in every generation.

3.1.5. Analysis

Figure 3 depicts the evolution of the fitness over 320 generations. The test is acquired over a subset of the MNIST test set in every tenth generation. The test set (10, 000 images) is separated from the training set (60, 000 images) and contains digits that were not presented during training.

While the mean fitness steadily increases over the generations, the best individual fitness exceeds 0.9 at generation 50 and improves to a fitness very close to 1.0 before decreasing again to around 0.9. Toward the end of training, we observe that the standard deviation of the individuals gets smaller and the mean increases. After a maximum standard deviation of 0.16 in generation 100, the spread of the ensemble contracts to a minimum standard deviation of 0.08 in generation 260, and remains low thereafter. It is important to note that the green curve indicates the performance of the highest performing individual in each generation, this is not necessarily the same individual. Currently we show 10 images for 500 ms on each generation in every training and testing phase which takes relatively long simulation times, thus hindering our ability to process the whole dataset and limits the total number of used images to 3, 200 (2, 880 training, 320 testing). Although the simulations take a relatively long time, using the HPC capabilities of L2L we are able to process an entire generation of 98 individuals including the optimization of a total of $98 \times 28,800$ weights in less than 3 min. In comparison a grid search on 28, 000 parameters exploring a range of 20 values for each weight would require the evaluation of $20^{28,000}$ combinations. Due to the fast convergence behavior of the EnKF it is possible to reach an

optimal solution in few generations. Our modifications to sample new individuals from well performing ones and perturbing them increases the possibility to find an overall better solution by exploring other parameter ranges. A future research direction we want to investigate is to move the optimization process of the weights into the inner loop and optimize the hyper-parameters of the optimizer. In this light, it would be interesting to use Nengo or Norse which are suitable for solving ML tasks with SNNs and optimizing the hyper-parameters of the optimizers provided by those libraries. Finally, we can compare the results by executing the same approach having NEST as the SNN back-end. Our setup for learning MNIST is different from other reported works in literature in terms of architecture, learning strategy, and even metrics to measure performance. This makes a direct comparison not straightforward. Previous studies have shown a high accuracy in the MNIST dataset by shaping the structure of the reservoir. For instance, Wijesinghe et al. (2019) divide the reservoir into clusters of locally connected neurons and change the connectivity in order to reach satisfactory results on different tasks. Zhou et al. (2020) apply neural search techniques and hyper-parameter optimization using a mix of covariance matrix adaptation evolution strategy and Bayesian optimization to modify the reservoir structure, reaching an accuracy of more than 90% on the MNIST dataset. They also report high accuracy on different spatio-temporal tasks.

3.2. Use Case 2: Fitting Electrophysiological Data With Arbor

This use case is concerned with optimizing the parameters of a biophysically realistic single cell model implemented in Arbor such that the response of the neuron to a specific input stimulus matches an experimental recording. Both passive parameters—morphology and resistivities—and active response to an external stimulus are commonly recorded in electrophysiological experiments. Similarly, the ion channels present are typically known. However, the internal parameters of the mechanisms—usually implemented as a set of coupled linear ODEs—are not known. To address this, we use L2L to fit the model parameters to the available data. This proof-of-concept aims at providing a robust way for model fitting for the Arbor simulator using HPC resources.

3.2.1. Description of the Simulation Tool

Arbor is a library for writing high-performance distributed simulations of networks of spiking neuron with detailed morphologies (Akar et al., 2019). Arbor implements a modification of the cable-equation model of neural dynamics which describes the evolution of the membrane potential over time, given the trans-membrane currents. In this model, neurons comprise a tree of *cables* (the morphology), a set of dynamics assigned to sub-sections of the morphology (called *ion-channels* or *mechanisms*), and a similar assignment of bio-physical parameters. The morphology describes the electric connectivity in the cell's dendrite and the mechanisms primarily produce the trans-membrane currents.

3.2.2. Optimizee: Morphologically-Detailed Single Cell

As outlined above, we expect models to be imported from laboratory data, that is a morphological description of the cell from microscopy, a template of ion channels with yet unknown parameter values, and some known data like the temperature of the sample. In addition, a series of stimulus and response measurements need to be provided, which will be the target of optimization. Our objective then is to assign values to the parameters to best approximate the measured response. For designing this use case, we focus on a single specimen from the Allen Cell Database with a known parametrization in addition to the input/response data (Lein et al., 2007).

We define the parameter sets **P** to be fit as a list of 4-tuples: a sub-section of the morphology, an ion-channel id, a parameter name, and the value to set the parameter to. Regions in the morphology are written as queries against Arbor's layout engine, e.g., selecting all parts of the dendrite where the cable radius is smaller than 1 µm becomes `(rad-lt (tag 2) 1)`, since `tag=2` has been set during morphology creation. Consequently, setting the parameter `tau` in the `expsyn` mechanism to 2 ms appears as

```
[.., ((rad-lt (tag 2) 1), expsyn,
 tau, 2), ..]
```

in the individual. Optimizee instances are constructed from are configuration file which lists the following items (example item)

- morphology file name (`cell.swc`)
- list of current clamps with expected response (`delay, duration, amplitude, ref.csv`)
- simulation parameters: length and time-step
- location where to record the response (`location 0 0.5`)
- fixed parameter assignments (`T=285 K`)
- list of ion channel assignments and optimizable parameters `[(tag 2), pas, e, -70, -30]`

Parameters to be optimized are given a bounding range used to automatically restrict the optimizer, here `e` may vary in the range of [-70 mV…-30 mV]. This data is sufficient—together with the statically known items—to construct a simulation in Arbor that can be run forward in time.

3.2.3. Fitness Metric

We implemented the naive approach of using the mean square loss as the measure of fitness. Given the experimentally obtained membrane potential $U_{\text{ref}}(t)$ we define the fitness as

$$\mathcal{L}(\mathbf{P}) = -\frac{1}{T^2}\sum_{t=0}^{T}\left[U_{\text{ref}}(t\cdot\tau) - U_{\text{sim}}(\mathbf{P}, t\cdot\tau)\right]^2 \qquad (2)$$

where $U_{\text{sim}}(\mathbf{P}, t)$ is the measurement produced by Arbor given the parameter set **P** and τ is the sampling interval of the voltage measurement. The optimizer attempts to maximize the given metric, which is why we defined the fitness as the negative of the L_2 norm here.

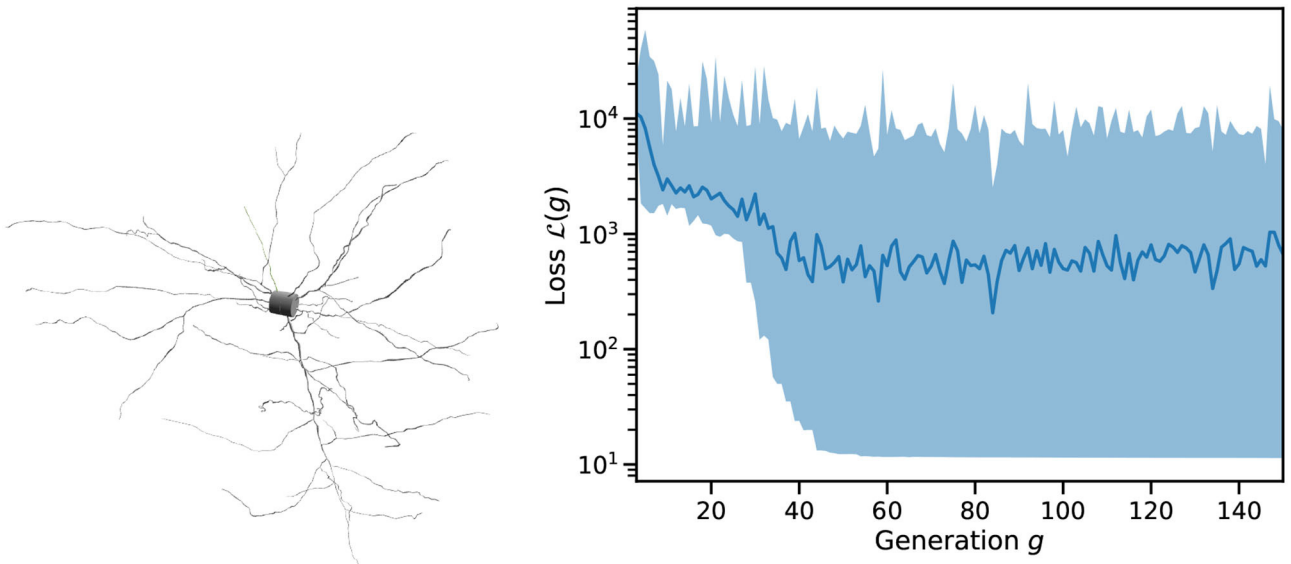


FIGURE 4 | Example input to Arbor and trace of a run of the optimizer. **(Left)** Cell morphology as consumed by Arbor and imported from the Allen DB, regions are marked as "soma," "dendrite," and "axon." **(Right)** Loss function over successive generations of the genetic optimizer for an example run of L2L on this cell starting from random parameters. Shown are the mean loss per generation (as a line) and the spread between minimum and maximum (shaded area).

Figure 4 shows an example of single cell morphology and the loss function across a single run of L2L. The scales and units of $\mathcal{L}$ are arbitrary. After roughly 50 generations, the best result has been identified and we found only minor improvements to the fitness after this. As can be seen in **Figure 5** (left), we quite easily reach a configuration that reproduces the *mean* membrane voltage but does not exhibit spiking behavior. From experience, we know that spikes are only produced for a narrow band of parameters in these complex configurations.

Thus, the fitness function will need to be extended to include the requirement for spiking. Furthermore, it seems prudent that the final result of the optimization process should include the responses to multiple separate stimulation protocols. Therefore, the overall fitness becomes a vector

$$\mathcal{F}(\mathbf{P}, \mathbf{I}) = \begin{pmatrix} \mathcal{L}(\mathbf{P}, I_0) \\ \mathcal{S}(\mathbf{P}, I_0) \\ \mathcal{L}(\mathbf{P}, I_1) \\ \vdots \end{pmatrix} \tag{3}$$

which—in conjunction with a vector of weights—is suited for use with L2L's multi-objective optimization. Here, $\mathbf{I}$ is the vector of stimuli and the function $\mathcal{S}$ collects the fitness with respect to the spiking behavior. Thus, the fitness function was changed to emphasize spiking behavior

$$\mathcal{L}(\mathbf{P}, I) = |\langle U_{\text{ref}} \rangle - \langle U_{\text{sim}} \rangle| \tag{4}$$

$$\mathcal{S}(\mathbf{P}, I) = \langle U_{\text{ref}} - U_{\text{sim}} \rangle \Big|_{U_{\text{ref}} > \sigma} \tag{5}$$

where $\mathcal{S}$ selects spikes by applying a threshold σ and then applies the temporal average $\langle \cdot \rangle$. As can be seen in **Figure 5** (right), we find spiking behavior with this fitness function, albeit still different from the expected outcome.

3.2.4. Optimizer: Evolutionary Algorithm

The fitness metric is used to drive the outer loop optimizer, an evolutionary algorithm searching for maximum fitness. This choice of the algorithm was motivated by prior studies showing it to be computationally efficient for this kind of fitting problem (Druckmann et al., 2007).

In the L2L framework, the genetic algorithm optimizer (GA) is a wrapper around the DEAP library (Fortin et al., 2012). This adapter takes care of handling the parameters received from the inner loop and prepares them for the optimization process. The DEAP library then facilitates the cross-over and mutation methods, applies them to the actual parameter set, and saves the best individuals into the Hall of Fame if they fare better than previous runs. Afterward, the optimized parameters are sent back to the optimizee, which then initializes the next generation of individuals.

Here, we use a population of 100 individuals and a total of 200 generations. Individuals in a generation are evaluated by using 16 parallel tasks on a single dual-socket node.

3.2.5. Analysis

We have shown a basic implementation for finding optimal parameter sets for single cell models using Arbor and L2L. This enables researchers to fit experimental data to neuron models in Arbor, a workflow that is important in practice and lacking so far in Arbor's ecosystem. The approach shown here so far is implemented in a straightforward fashion but falls short to reach the desired configuration in a reasonable time frame.

A fitness function based on salient features is generally more successful in producing spiking behavior (Druckmann et al., 2007; Gouwens et al., 2018). We expect the current fitness implementation to evolve further, likely including more

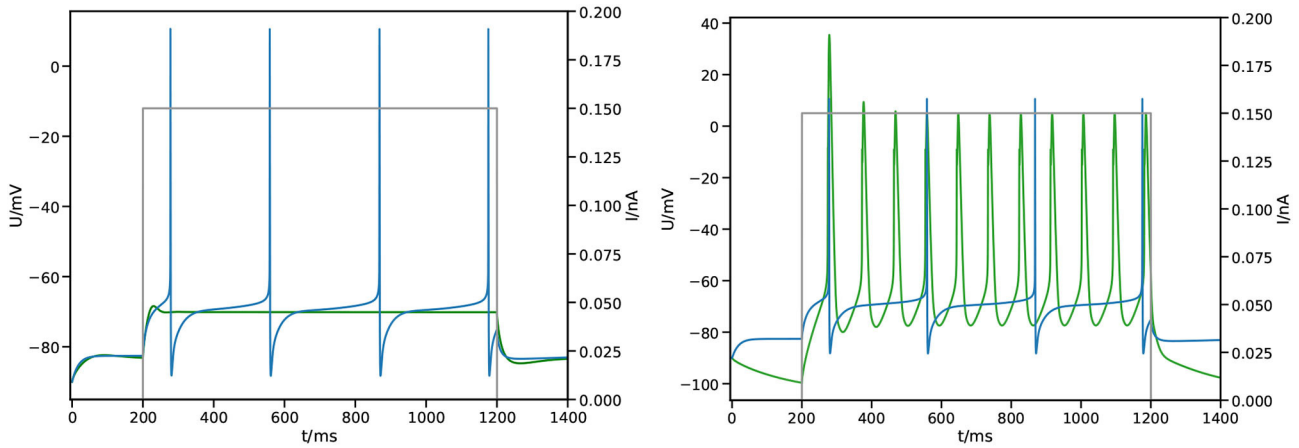


FIGURE 5 | Impact of the fitness function. Shown is the measured membrane potential at the center of the soma from the simulation • against reference • and applied stimulus •. **(Left)** Simple square loss, the best result after 100 generations. **(Right)** Feature based fitness, the best result after 100 generations.

features, such as the resting potential and mean spike frequency. Further, L2L does not normalize parameters, thus parameters that have significantly different ranges can pose issues to the optimization process, e.g., the test case here features parameters of magnitude 100 as well as 10^{-7}. Given the bounding annotations in our configuration, we implemented normalization within the optimizee and L2L handles uniform ranges $[0, 1]$[4]. To cope with common time-restrictions on the used resources in the mean-time, we implemented a method to resume optimization given an intermediate result. Currently, this workflow is being extended beyond the proof-of-concept state we presented here. A further open task is to investigate the impact of the hyper-parameters passed through L2L to DEAP, such as tournament size, population size, etc.

Another extension is the use of accelerators (GPUs), which allow for massively parallel evaluation of individuals. Arbor is able to use GPUs for simulations efficiently starting at a few thousands of cells per GPU. This would enable processing an entire generation of the optimization process at once. Given the current number of 100 cells per generation, this is not yet profitable, but for larger generation sizes and additional stimulus protocols, it becomes attractive. L2L was extended to enable a vectorized version of the evolutionary algorithm similar to the multi-gradient descent approach presented in use case 4 (Section 3.4).

3.3. Use Case 3: Foraging Behavior With Netlogo and NEST or SpikingLab

In this use case, we describe optimizing the foraging behavior in a simulated ant colony. The colony consists of 15 ants, all of which are searching for food (big green patches, **Figure 6**). Any food found must be brought back to the nest. Ants communicate with each other by dropping pheromones on the ground (blue patches) whenever the food is found or the nest is reached. The pheromone can be smelled by other ants which then can follow the trail left on the ground. Each ant is controlled by an SNN, which is an identical copy for every ant. Here, we use L2L to configure its weights and delays so that the ants bring food back to the nest as efficiently as possible.

3.3.1. Description of the Simulation Tools

NetLogo is a multi-agent simulator and modeling environment (Tisue and Wilensky, 2004). It is widely used as an educational and scientific tool for the study of emergent behavior in complex systems. Agents are expressed as objects that can communicate with each other. In our setting, NetLogo helps us to observe and manipulate the state of every neuron and synapse. For the simulations, we have two backends: NEST (see Section 3.1.1) and SpikingLab (Jimenez-Romero and Johnson, 2017). SpikingLab is an engine directly integrated within NetLogo and can be easily and quickly used for small scale networks, as we present in our use case. Invoking NEST from NetLogo causes a minimal communication overhead since NEST needs to be called as an external process. For larger networks, it is preferable to use NEST since its higher simulation efficiency compensates for the communication overhead.

3.3.2. Optimizee: Simulated Ant Brain

In the first iteration, the optimizee creates the individual inside the `create_individual` function. The individual consists of network weights and delays. The weights are uniformly distributed in $[-20, 20]$, while the delays range between $[1, \ldots, 7)_{\mathbb{N}^+}$. The network has an input, a hidden, and an output layer, the neurons are all-to-all connected for every layer as depicted in **Figure 7**. The input layer consists of 12 neurons. The first three neurons are receptors to smell the direction of the pheromone. The next three neurons are responsible to locate the nest. The queen receptor indicates the middle of the nest. Reward and nociceptors determine the reward and punishment for the ant. The green and red photoreceptors are

[4]Note that this can introduce different issues with numerical precision if said ranges span too many orders of magnitude.

FIGURE 6 | The ant colony is searching for food (big green patches with brown leaves). The ants are communicating *via* pheromones which are dropped on the ground (blue-white patches) when food is found or when the ants return to the nest (black-brown patch). Green colored ants are transporting the food, while orange colored ants are exploring the environment or following the pheromone trail. The red border around the world is an impenetrable wall and prevents ants from crossing from one side to the other. The pheromone trail decays with time if it is not reinforced by other ants.

triggered when food or a wall is seen. Finally, the heartbeat neuron stimulates the network in every timestep with a small direct current to keep a low dynamic ongoing in the network. The four output neurons are responsible for the movement and for dropping the pheromone. Similar to the first use case in Section 3.1.5, the total number of individuals is 98. The total number of connection weights (250) and delays (250) is derived as follows: 110 connections from the input to the middle layer, 10 connections from the heartbeat neuron to the middle layer, 90 connections in the middle layer, and 40 connections from the middle layer to the output ($110 + 10 + 90 + 40 = 250$). The weights and delays can be min-max normalized if specified. The optimizee saves these parameters as a csv file before starting the simulation. The model is invoked by a Python subprocess[5] in the `simulate` function, which then calls the headless mode of NetLogo to start the run. The optimizee waits until the simulation is finished and collects the fitness value from a resulting csv file which is written after the simulation ends.

The user has to set whether NEST or SpikingLab is invoked as a backend inside the simulation. NEST is known as a subprocess by NetLogo, while SpikingLab is directly accessed by the model. In the case that NEST is selected, the parameters have to be passed to it as well since the network needs to be constructed with the new parameters. This can be done either by loading the parameter in a csv file within NEST, or NetLogo can read the csv file and pass the values to the simulation.

The parameters are restricted within the `bounding_func` function if their values exceed the specified ranges after the optimization process. Weights are clipped to the range of $[-20, 20]$ and delays to $[1, 5]$.

3.3.3. Fitness Metric

The fitness function for the ant colony optimization problem rewards finding food and bringing it back to the nest while punishing excessive movement.

We define the ant colony fitness f_i of optimizee i as:

$$f_i = \sum_{t=1}^{T} \left(\sum_{j=1}^{J} \mathcal{N}_{i,j}^{(t)} + \mathcal{F}_{i,j}^{(t)} - \mathcal{C}_{i,j}^{(t)} \right), \tag{6}$$

where $t = 1, \ldots, T$ is the simulation step, T is the total simulation time, J is the total number of ants in the colony, and j indexes the ants. $\mathcal{N}$ is the reward for coming back to the nest with food, $\mathcal{F}$ is a reward for touching the food, and $\mathcal{C}$ is the movement cost. Every movement, rotation, and pheromone dropping is added toward $\mathcal{C}$. We set the cost as follows: Rotation -0.02, pheromone dropping -0.05, and movement -0.25. The movement has a higher cost since we would like to restrict vast movements and force them to return to the nest. We also punished resting with -0.5 to speed up the movement and to slightly induce exploration. The rewards are returning to the nest 220 and touching food 1.5. A high reward for coming back to the nest is necessary, otherwise, the ants are spending a long time exploring the environment even when the food is found. This slows down learning and hinders solving the task.

3.3.4. Optimizer: Genetic Algorithm

We use a genetic algorithm to optimize the weights and delays in the ant brain network. This is the same class of optimizers as used in Section 3.2.

3.3.5. Analysis

Figure 8 depicts the evolution of the fitness of the ant colony over 800 generations. Initially, the ants move a lot without retrieving food, resulting in a negative maximum fitness. After around 200 generations, the mean fitness is consistently positive and the best solution is close to 10,000. In following generations, the mean fitness saturates at around 5,000, with increasing best fitness. After 800 generations, the L2L run is stopped with the best individual fitness close to 15,000. Similarly to use case 3.1, L2L enables us to execute 98 individuals in

[5]https://docs.python.org/3/library/subprocess.html

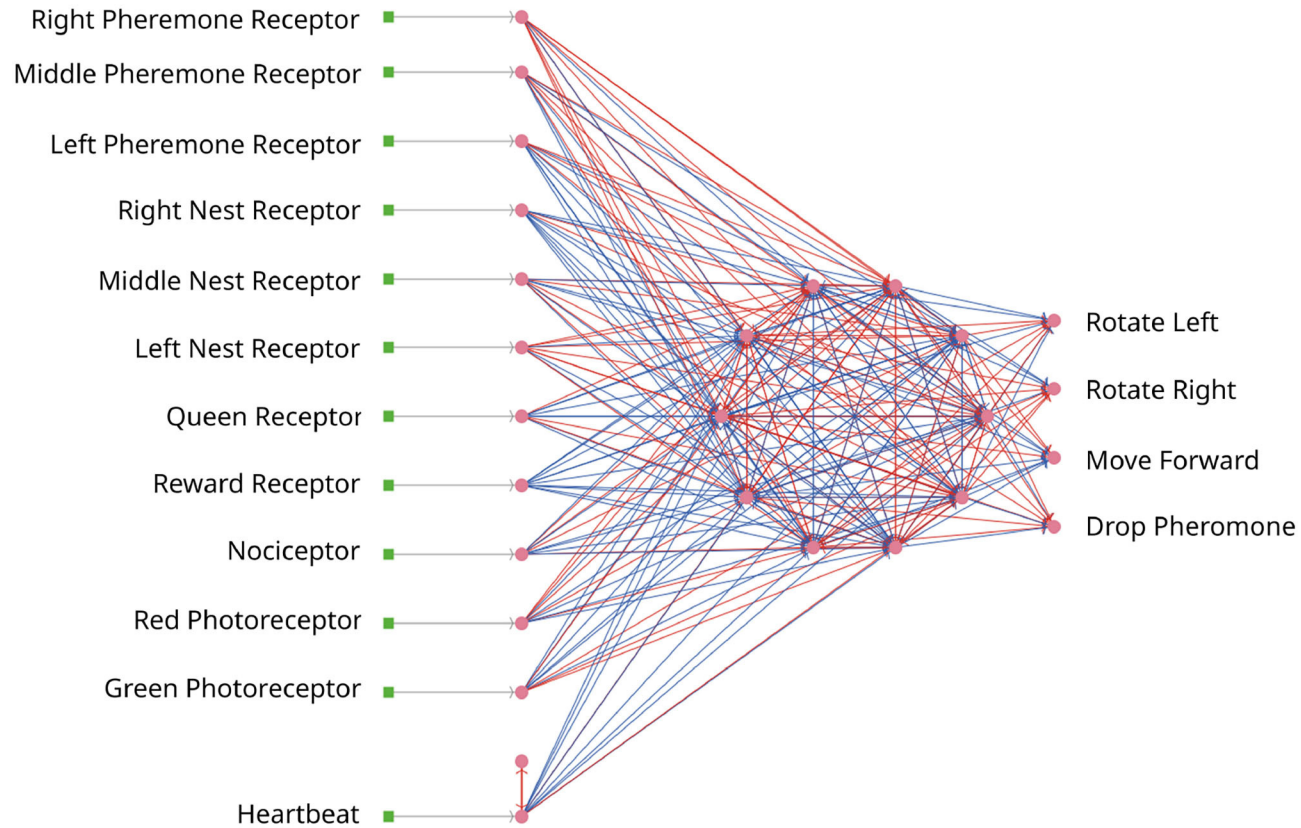


FIGURE 7 | The SNN for the ant colony. Every ant is steered by an SNN. Neurons are depicted as pink dots and excitatory/inhibitory connections as red/blue lines. All networks are identical.

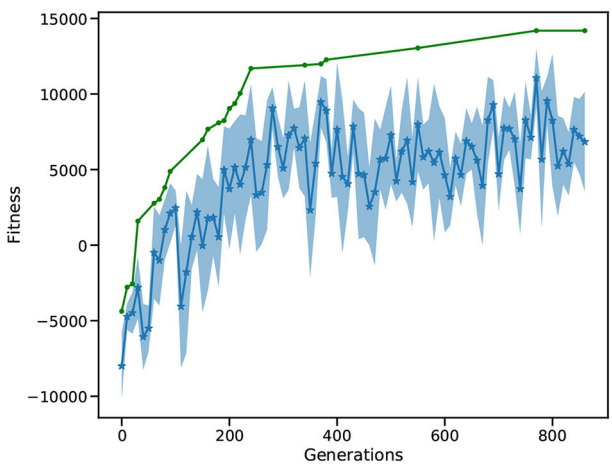


FIGURE 8 | Fitness of the ant colony. The blue curve shows the mean fitness and the shaded area is the SD. The green curve indicates the best solution found so far, and thus rises monotonically.

parallel, where a generation is optimized in less than 2 min. A grid search algorithm with 20 values to explore weight and delay combinations would require 20^{500} possibilities to test for. The mutation and cross-over steps of the GA increase the parameter space and avoid local minima, without loosing performance. The best individuals are saved in the Hall of Fame (HoF) if they have better fitness than their predecessors. If an optimization step produces underperforming individuals, it is possible to recombine the new set utilizing the HoF. Due to the parallel distribution of individuals and the GA optimizer, we are able to find well performing individuals in less than 400 generations. In contrast to other literature optimizing ant colonies using rule-based systems, our work describes the optimization of an SNN that learns the foraging behavior of an ant. The decision making of each ant is not based on fixed rules (e.g., if food is found turn around 180° and go back to the nest), instead, it depends on the firing activity of the network in response to the perceived environment. Compared to the ant colony model provided by NetLogo (Wilensky, 1997), which solves the foraging task within $\approx$ 15,000 steps, our SNN solution takes between 15,000 and 20,000 steps with a diffusion rate of 20 and evaporation rate of 1. However, when the environmental conditions change to the detriment of the pheromone communication (e.g., the evaporation rate increases and diffusion rate decreases), the performance of the two implementations becomes closer. In general, utilizing the network solution enables the ants to be more adaptable toward environmental modifications.

3.4. Use Case 4: Fitting Functional Connectivity With TVB

This use case describes tuning the parameters of a whole brain simulation using the GPU models of The Virtual Brain simulator (TVB; Sanz Leon et al., 2013) to give the best match to empirical structural data.

To do clinical research with TVB, it is often necessary to configure the parameters of a model for a specific person such that it matches obtained empirical data. First, the brain is parcellated into different regions, based on many available atlases (Bansal et al., 2018). The connectivity of these regions is determined using diffusion weighted imaging, estimating the density of white matter tracts between the regions, resulting in a connectivity matrix which is regarded as the structural connectivity. Finally, a model that represents the regional brain activity must be chosen. To optimize the match between a specific person and the TVB simulation, obtained fMRI can be used to further personalize the structural connectivity (Deco et al., 2014).

Due to the high dimensionality of TVB models and the wide variation in possible parameter values, fitting patient data often requires extensive parameter explorations over large ranges. In this use case, the simulated functional connectivity is matched to the structural connectivity. The task has the underlying assumption that regions that are anatomically connected often show a functional connection (Honey et al., 2009). In this task, we want to find the values for the `global_coupling` and `global_speed` variables, characteristic of the connectome of a TVB stimulation, which gives rise to the strongest correlation between the structure of the brain and the functional connectivity, i.e., the relationship between spatially separated brain regions.

3.4.1. Description of the Simulation Tools

The Virtual Brain is a simulation tool which enables researchers to capture brain activity at mesoscopic level using different modalities such as EEG, MEG of fMRI, using realistic biological connectivity. A TVB brain network consists of coupled neural mass models (NMM) whose dynamics can be expressed by a single or system of ordinary differential equations. The coupling of the NMMs is defined by the connectivity matrix. The NMMs describe, e.g., the membrane potential or firing rate of groups of neurons using differential equations, which are then solved numerically. In this use case, we utilize an Euler based solver. RateML (van der Vlag et al., 2022), the model generator of TVB, enables us to create the desired TVB model written in CUDA for the GPU and a driver to simulate the model, from a high level model XML file.

Unlike the use cases discussed above, in this case, we exploit GPU-parallelization by defining an optimizer that can process a vector of fitnesses and create new individuals for multiple TVB simulations executed in parallel on the GPU. An overview of this process is shown in **Figure 9**. The optimizee in the inner loop spawns a number of threads (here: 1,024) according to the users defined parameters ranges and resolution. Each thread represents a TVB instance, simulating a unique set of parameters. The fitness is computed for each instance, and the outer loop optimizer selects the best fitness by using the gradient ascent strategy. The arrows indicate the independent iterations of the vector of fitnesses. In the figure, six TVB simulations run in parallel, thus the optimizer needs to iterate a vector of six fitnesses.

3.4.2. Optimizee: Whole Brain Simulation

The `create_individual` function initializes a first instance for the TVB simulation. The structural connectivity is usually obtained from the patient but in this case, the standard TVB connectivity for 76 nodes is used. We model the regions with the `Generic2DimensionOscillator` (G2DO; Ott and Antonsen, 2008). A dictionary is created which contains initial random values for the optimization parameters, `connection_speed` and `coupling_strength`.

For subsequent simulation generations, the optimizee reads the adapted values from a text file written by the optimizer and utilizes the Python subprocess module to spawn a new TVB simulator object with the corresponding parameterization. When the TVB simulation is complete, the fitness for each TVB instance is computed and written to a separate text file. The text files are read by the optimizee reformatted for processing by the optimizer.

3.4.3. Fitness Metric

The computation of the fitness for this task is 2-fold. In the first step, the simulated functional connectivity is determined by computing the Pearson product-moment correlation coefficient, ρ_{xy}, of the simulated 76 regions according to Equation 7.

$$\rho_{xy} = \frac{\mathrm{Cov}(x, y)}{\sigma_x \sigma_y}, \tag{7}$$

where $\mathrm{Cov}(x, y)$ is the covariance of variables x and y and σ_x and σ_y are the SD. This first step determines how strong the dynamics of the simulated regions correspond to one another. A strong functional correlation means that the simulated activity between the spatially separated brain regions is more similar. The second step is to determine the correlation between the obtained functional and the structural connectivity, the weight matrix used in the simulation, also using Equation 7.

Listing 2 | Implementation of the correlation computation between functional and structural connectivity.

```
SC = connectivity.weights / connectivity.weights.max()
for i in range(couplings * speeds):
    FCSC[i] = np.corrcoef(FC[:, :, i].ravel(), SC.ravel())[0, 1]
```

The Python implementation of the second step is shown in **Listing 2**, where `SC` is the structural connectivity and `FC` is the

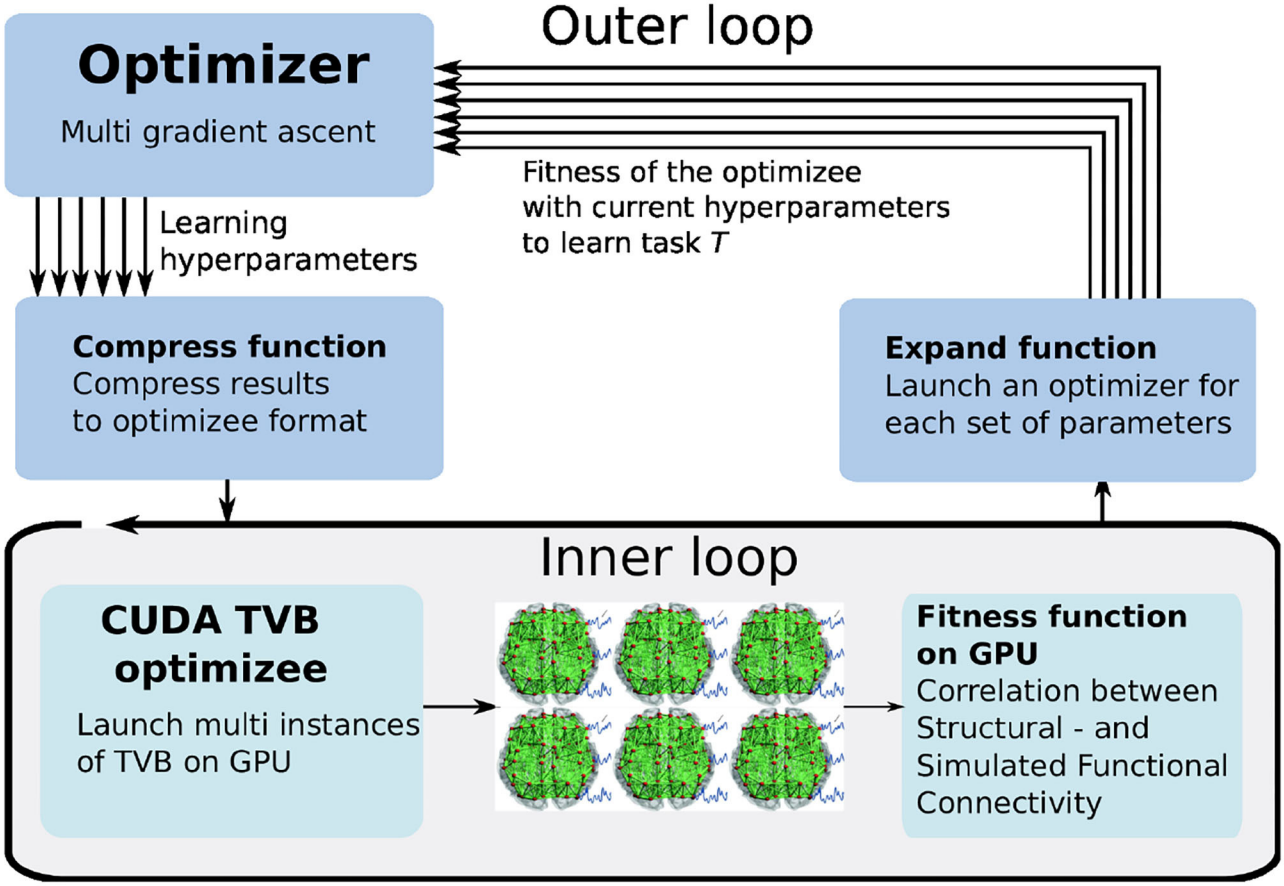


FIGURE 9 | The multi-gradient ascent implementation of L2L. The inner loop launches multiple instances of TVB on the GPU simulating different sets of parameters. The outer loop selects the best fitnesses and produces a new parameters range.

simulated functional connectivity that was computed previously. On line 1, the weights are normalized. In the for-loop on line 2, the correlation with the structural connectivity is computed. The FCSC holds these correlations and is the array of fitnesses returned to the optimizer.

3.4.4. Optimizer: Multi-Gradient Ascent

The best fitness is selected with a gradient ascent optimizer. The existing optimizer has been adapted for processing the vector of fitnesses returned by the GPU, named multi-gradient ascent (MGA). In order to adapt it to vector processing, the fitnesses need to be expanded before processing and compressed afterward, as is shown in **Figure 9**. The expansion transforms the obtained fitnesses from the optimizee process to a data structure in which the obtained fitnesses are linked to the used parameters, thus enabling the multi-gradient ascent optimizer the possibility to select the best fitness and define a range for the new parameters to be sent to the optimizee. When the optimizer has selected the parameters for the optimizee, it compresses the new individuals to a data structure that just contains the new parameter combinations for the optimizee. Aside from the expanding and compressing, the MGA algorithm determines the new values for the individuals similar to gradient ascent.

3.4.5. Analysis

The results in **Figure 10** show the evolution of the mean and best fitness for a generation of 1,024 parameter combinations for the `global_speed` and `global_coupling` variables, with a learning rate of 0.01 and four individuals. These four individuals each spawn 1,024 TVB simulations on the GPU, enlarging the chance of success. Each generation contains a TVB simulation of 4,000 simulation steps with a $dt = 0.1$. These results were obtained using a NVIDIA V100 GPU on the JUSUF[6] cluster. Our results show that after 30 generations the best attainable fitness (green curve) is reached (c.f. Deco et al., 2014).

Comparing the GPU population based on a single L2L implementation, the latter would need more generations before the best fitness is attained. The likelihood of finding a suitable solution in earlier generations rises with the size of the

[6]https://fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUSUF/JUSUF_node.html

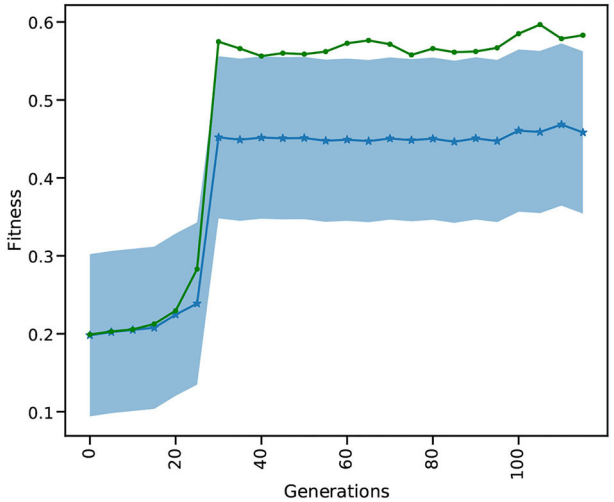


FIGURE 10 | SD of the mean and best out of 1,024 fitnesses for 116 generations for the multi-gradient TVB parameter optimization TVB simulation. The blue curve is the mean fitness over the population of 1, 024 and the shaded area gives the SD. The green curve shows the best fitness for each generation.

population: the more configurations considered in a single generation, the faster it converges to the best value. The GPU implementation has already considered 30 × 1,024 different parameters values, after which the optimal fitness is found (**Figure 10**), while the single implementation would have only 30. A single implementation would need at least 30,720 generations to find the same result, but would very likely need many more. Additionally, the GPU makes it very convenient to execute many simulations in parallel by not having to split them up onto multiple nodes, without communication overhead and decreasing wall clock time even further.

3.5. Use Case 5: Solving the Mountain Car Task With OpenAI Gym and NEST

In this use case, we describe a solution to the OpenAI Gym Mountain Car (MC) problem. The MC task is interesting since it requires the agent to find a policy in a continuous state space constituted by the position and velocity of the car. At the same time, the action space is discrete, limited to three possible actions: accelerate left, accelerate right, and do nothing. The initial position and velocity of the car are set randomly by the environment; the aim is to reach the goal position (yellow flag) as depicted in **Figure 11**. As the car's motor is weak, consistently reaching the goal at the top of the hill requires the agent to learn a policy that swings the car back and forth in order to build up momentum. The challenge is considered solved if the car reaches the goal position in an average of 110 steps over 100 consecutive trials. We implement a feed-forward LIF SNN in NEST to encode a policy and optimize the weights so as to improve the ability of the network to solve the task.

3.5.1. Description of the Simulation Tools

The OpenAI Gym (Brockman et al., 2016) is a software library that provides an interface to a wide range of environments for experimentation with reinforcement learning techniques. NEST has been described in Section 3.1. Both simulators are instantiated and invoked by the optimizee process which implements the closed-loop interactions. These interactions are synchronized in such a way that for each simulation step of the MC environment, the SNN is simulated for an interval of 20 ms in NEST. On completion of a simulation interval, the state of the network is sampled and fed back as an action to the MC environment.

3.5.2. Optimizee: Spiking Feed-Forward Policy Network

The SNN of LIF neurons that controls the actions of the car is implemented in NEST. The inputs to the SNN are the position $[-1.2, 0.6]$ and velocity [-0.7, 0.7] variables which are discretized and encoded using 30 input neurons for each variable. For the discretization (binning) of the continuous variables, the width (w) of the bins is given by the minimum (min) and maximum (max) value of the interval divided by the number of input neurons (n) available for each variable. Each value within the range is discretized into a bin which corresponds to one input neuron:

$$w = \frac{min + max}{n} \tag{8}$$

Once a value falls into a bin, its corresponding neuron is activated by a dc current as provided by a connected dc generator resulting in a firing rate of 500 Hz. The 60 encoding neurons have all-to-all connections to an intermediate layer of five neurons, which in turn have all-to-all connections to the three neurons in the output layer corresponding to the three possible actions. The action sent to the OpenAI Gym environment depends on the activity of the three neurons in the output (third) layer. Each output neuron represents one of the possible actions. Following a winner-takes-all approach, the neuron with the highest spiking activity determines which action is sent to the OpenAI Gym environment. **Figure 11** illustrates the spiking network and the closed-loop interaction with the MC environment on the basis of input variables and output actions.

Similar to the Netlogo use case (see Section 3.3), at the beginning, the optimizee creates the individual inside the `create_individual()` function. The total number of individuals per generation is 32. Each individual consists of network weights, which are initially uniformly distributed in $[-20, 20]$. There are 315 weights corresponding to the $(60 \times 5) + (5 \times 3) = 315$ synaptic connections in the network. The instantiation and orchestration of OpenAI Gym and NEST simulator (including the set-up of the SNN) is carried out by the optimizee. Each simulation runs for 110 simulation steps (where a simulation step corresponds to an action being sent to the environment) or until the goal position is reached. Once the simulation is completed, the optimizee

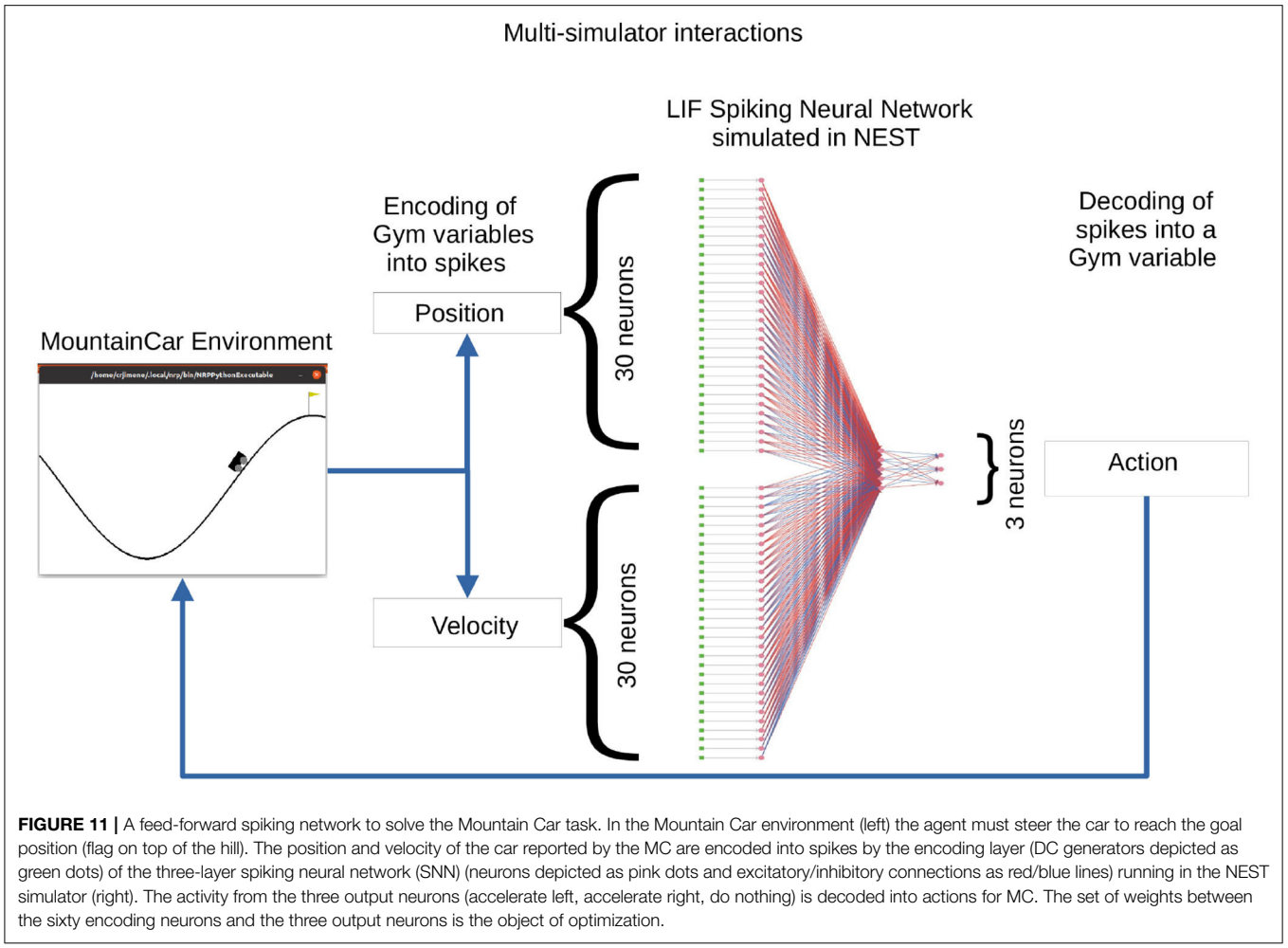


FIGURE 11 | A feed-forward spiking network to solve the Mountain Car task. In the Mountain Car environment (left) the agent must steer the car to reach the goal position (flag on top of the hill). The position and velocity of the car reported by the MC are encoded into spikes by the encoding layer (DC generators depicted as green dots) of the three-layer spiking neural network (SNN) (neurons depicted as pink dots and excitatory/inhibitory connections as red/blue lines) running in the NEST simulator (right). The activity from the three output neurons (accelerate left, accelerate right, do nothing) is decoded into actions for MC. The set of weights between the sixty encoding neurons and the three output neurons is the object of optimization.

returns the calculated fitness value to the optimizer. The `bounding_func()` function ensures the weights are clipped to the range $[-20, 20]$ if the values exceed this range after the optimization process.

3.5.3. Fitness Metric

The fitness function for the MC optimization problem is defined as the maximum horizontal position reached by car during an episode comprised of 110 simulation steps, i.e.,

$$f = \max_T(\vec{\mathcal{P}}_{\mathcal{T}})$$

Where $\max_T$ returns the item with the highest value in a vector and $\vec{\mathcal{P}}_{\mathcal{T}}$ contains the position of the car on each simulation step up to $T = 110$.

3.5.4. Optimizer: Genetic Algorithm

The optimization method is identical to that used in Section 3.3. Afterward, the optimized parameters are sent back to the optimizee, which then initializes the next generation of individuals.

3.5.5. Analysis

Figure 12 depicts the fitness of the SNN over 400 generations. After 50 generations, the fitness becomes positive showing that the car is moving toward the goal position. The best solution (goal position of 0.5) is first reached around generation 160. In following generations, the mean fitness saturates at around 0.3, while the best fitness reaches the maximum of 0.5. After 400 generations, the L2L run is stopped with the best individual fitness being 0.5. Finally, we confirmed that the fittest individual could solve the MC problem. We ran a thousand episodes (each episode lasting for a maximum of 200 simulation steps); the spiking network achieved the required average of 110 or less simulation steps over 100 episodes. Our solution requires 101 simulation steps on average to reach the goal position and thus solves the task (data not shown).

The Mountain Car problem has been approached using several ML techniques most of them focusing on reinforcement learning (Heidrich-Meisner and Igel, 2008; Weidel et al., 2021) and gradient descent (Young et al., 2019). Current implementations are able to solve the challenge while delivering a good performance in terms of speed of convergence and

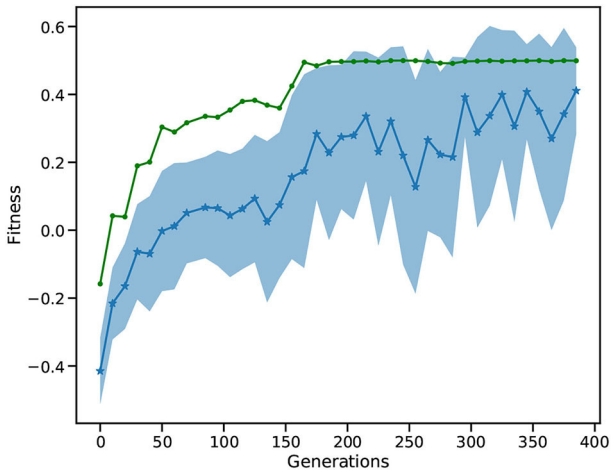


FIGURE 12 | Fitness of the MC run. The blue, starred line shows the mean of all individuals while the shaded area is the SD. The green line is the best fitness per generation.

the obtained final score. We took an evolutionary approach by using a GA to optimize an SNN that is able to solve the MC obtaining consistently a high reward (over 100 trials). Evolutionary strategies have shown comparable performance to reinforcement learning and gradient descent algorithms in problems where learning to sense and act in response to the environment are required (Salimans et al., 2017; Such et al., 2017; Stanley et al., 2019). Another advantage with the evolutionary approach is the parallel exploration of the solution space. In L2L, each individual is run as an independent optimizee process. The framework enables us to execute a large number of parallel optimizees in multi-core CPUs and HPC infrastructures.

4. DISCUSSION AND FUTURE WORK

Simulations in different science domains tend to become more and more complex and span over multiple disciplines and scales. These simulations usually have a large number of parameters to configure, and researchers spend a long time tuning the model parameters manually, which is difficult and time-consuming. To tackle these issues, it is necessary to have an automated tool that can be easily executed on local machines or likewise on super-computers. We present the L2L framework as a flexible tool to optimize and explore ranges of parameter spaces. Because the tool does not require a particular type of simulation, i.e., it is agnostic to the model in the inner-loop, it enables the optimization of any type of parameter resulting from the model, as long as fitness can be calculated and sent to the outer loop.

In Section 3, we described several neuroscientific use cases at different scales. The optimizations range from finding the correct set of parameter configurations to determining network dynamics to solving optimization problems up to exploring values for specific growth rules. In all cases, the optimization methods in the outer loop treated the inner loop simulations as black box problems and similarly, the optimization technique was unknown to the inner loop.

In terms of implementation, every optimizee follows the same structure by providing three functions: 1. creating the individual, i.e., the parameters to optimized, 2. starting and managing the optimizee run and providing fitness to asses the simulation performance, and 3. optionally constraining the parameter exploration range. The framework offers a plethora of built-in optimization techniques. Most of them are population based optimizers, which require several individuals and fitness or a fitness vector. Both the fitness and the population approach are incorporated into the optimization. For example, with genetic algorithms and the EnKF, the fitness is used to rank the individuals. A large population enables a wider range to explore parameters and find possible good initializations, which leads to a faster convergence. In order to not get stuck in local optima, most of the optimizers offer techniques to perturb the individuals and additionally enlarge the parameter space (which of course can be bounded if needed).

Clearly, executing a high number of individuals leads to an increase in computational requirements. By utilizing MPI in combination with the JUBE back-end, it is easy to deploy simulation and optimization on high performance computers in an automated fashion. From the users' perspective, only a few parameters have to be configured in a run script. The optimizees for the inner loop are created and the simulations are executed in parallel. One of the practical reasons for the population based optimizers is that the simulations are very easily parallelizable: each simulation can be conducted independently. Only the parameters have to be collected in a single step and fed into the optimizer. Afterward, the optimized parameters are distributed for the next generation and the new simulations can be started.

The TVB use case is an example of demonstrating a parallelized simulation in the optimizee. We show that we successfully reconfigured the gradient ascent optimizer to a version that can process a vector of fitnesses. We used this optimizer to find the best parameter setting for a TVB model such that the match between simulated functional and structural connectivity is optimal. Results from performance testing for the RateML (van der Vlag et al., 2022) models show that for a double state model such as the `G2DO`, on a GPU with 40 GB of memory, up to $\approx$ 62,464 (61 times more parameters), can be simulated in a single generation, taking approximately the same amount of wall time due to the architecture of the GPU. This would reduce the time it takes for each generation and increases the range and resolution of the to be optimized processes even further; opening up possibilities for experiments requiring greater computational power. Moreover, this particular optimizer is not limited to TVB simulations only. Any process which uses a parallel architecture, e.g., GPU, CPU or FPGA, for which the output is a vector of fitnesses, can be adapted as an optimizee for the MGA optimizer. The utilization of the `subprocess` library and information

transfers *via* in- and output text files, makes usage of this optimizer generic for any process. The MGA is just one example of an optimizer adapted to process multiple fitnesses, in theory, any of the optimizers can be adjusted to handle multi fitness optimizees.

4.1. Choice of Fitness Function and Optimizer

One important point to mention is the challenge of creating the fitness function. Every fitness function is a problem specific and finding a suitable function is often a complex task. In some cases, the fitness is given by the design of the problem (c.f. Section 3.1, in this case supervised learning). To illustrate the point, the task in Section 3.3 can be extended so that the ants are punished whenever they collide. However, just adding a simple cost value for the collision makes the training and optimization much harder, the ants exhibit erratic behaviors, such as spinning around or stopping moving after a few steps. Potentially, this behavior might resolve with enough generations, but it is more likely that the fitness function would need to be adapted. Even for the simple example shown here, the fitness function had to be carefully balanced in terms of the punishment and reward cost, which lead to several trials and manual adjustments. Thus, the exploratory and exploitative behavior is influenced by the fitness function. With a strict fitness function, i.e., every action in the simulation generates a reward or a punishment, it may be possible to exploit local optima; however, it may restrict the exploration of different, better optima. Conversely, making the fitness function too lax may lead to an overly exploratory behavior that does not exhibit any exploitation.

The choice of the optimizer is based on experience, the familiarity with the task and often includes a trial and error approach. Furthermore, the choice may be dependent on the task itself. For instance, in a supervised learning scheme, the "observable" parameter of the EnKF can be modified to support labels and enable this optimizer for supervised training. However, other optimization techniques may not be suitable as they cannot incorporate the concept of labels into their optimization process without extensive changes. It is not easy to recommend general optimization solutions for a variety of problems, and it is out of the scope of this work, we instead refer here to further literature (Okwu and Tartibu, 2020; Malik et al., 2021; Oliva et al., 2021). However, we would like to discuss some pointers which may be helpful in choosing an optimization technique when using L2L. Gradient descent and Kalman filtering can provide a directed and fast search within the parameter space. If it is known that the optimization problem space is smooth and ideally convex, the *gradient descent algorithm* is known for providing an efficient solution. The EnKF can also provide a fast convergence for non-convex problems with several optima and is especially suited for problems where calculation of the gradient is not possible or requires complex approximations. This can be particularly useful for problems where fast optimization with adequate results is more important than thorough explorations of vast parameter spaces to identify the optimal parameter configuration. Both the EnKF and gradient descent are suitable for optimization in high dimensional parameter spaces, such as the weight optimization of neural networks.

In contrast, if the solution space is not known and exploration is the focus, *genetic algorithms*—from the family of optimizers inspired by nature—may be the correct choice. By creating new individuals using mutation and cross-over, genetic algorithms can cover a vast space and still be very performant. For example, we also used genetic algorithms to optimize the network in use case 1 and obtained reasonable optimization results but did not reach as high a performance as with the ensemble Kalman filter (data not shown). The dimensionality of the parameter space in combination with the optimization algorithm chosen plays a key role in the outcome of the optimization. From our experience with the use cases presented here, we have seen that genetic algorithms work well with parameter spaces in the range of tens to thousands of dimensions.

Learning to learn provides several additional optimizers beyond those introduced in the use cases, which also have advantages in certain applications. The *evolution strategies* optimizer creates new individuals by perturbing, i.e., adding Gaussian noise, to the fittest individuals to create new ones and falls into the same category as the GA but uses stochastic gradient descent as an optimization technique. For example, the authors of Salimans et al. (2017) optimize large networks which are then able to play Atari games. Similarly, the *natural evolution strategies* (NES) optimizer samples from a multivariate Gaussian distribution to obtain new individuals. Wierstra et al. (2014) employ NES on several benchmark tasks with different parameter dimensions. They conclude that NES is applicable on low dimensional and high-dimensional and multi-modal problems.

The performance of *simulated annealing* depends heavily on the annealing schedule selected. L2L provides a variety of schedules to choose from the exploration progress and they define the ratio between exploration and exploitation of the algorithm. Simulated annealing can be an excellent tool to perform initial explorations of large parameter spaces and progressively move from exploration to exploitation as experience with the simulated model increases. The L2L version includes a cooling factor that allows the user to explore the balance between exploration and exploitation.

Cross entropy is highly directed and fast to converge. It is well suited for dealing with noisy optimization problems and large parameter spaces. In contrast, L2L also provides the *grid-search*, a technique that just iterates over the given parameter range in a brute force manner. This technique can be used for rather small parameter ranges if nothing is known about the problem space.

4.2. Outlook

Specifically regarding our presented use cases, future work will include multi-objective optimization to decouple the objectives from a specific fitness function and optimize the fitness functions in interchangeable steps. The L2L framework already supports

multi-objective optimization since it can handle several fitness values. Alternatively, the optimizee can be written in such a way that it exchanges the fitness function in certain generations and still returns one fitness value.

A visualization of the trajectories through generations may give further insights for a follow-up analysis of the parameters. We aim to implement a visualization tool that can plot the evolution of the parameters using simple diagrams such as histograms, correlations, and similar statistics. A desirable feature would be to interact with the plot while the simulation is ongoing, as demonstrated by Tensorboard[7]. A challenge here is to interact with the results whenever the run is conducted on an HPC, as many super computing centers no longer allow X-forwarding—a network protocol to control and display a remote software from a local computer. Instead, other mechanisms for interactive computing need to be considered such as virtual network computing[8].

In preliminary work, we were already able to run the L2L simulations on an HPC while instructing the run from a local machine. By utilizing UNICORE (Streit et al., 2005), a tool for distributed computing, we could successfully send an optimizee to a specified HPC, initialize the L2L framework, run the optimizations, and collect the results. For this approach to work, we have to ensure that the L2L framework is correctly deployed on the remote side. Seamless integration of all tools in the process chain is required. This approach also leads toward a vision of L2L as a service, where users can submit optimization workloads using a simple API. Despite the advantages of this approach, new aspects should be considered to protect user data and any sensitive data that can be used or produced during simulations. In order to deploy this service, full integration with the EBRAINS[9]. infrastructure is our target for the near future, as this will enable L2L to support the neuroscience community while being part of a well-established research platform.

Another necessary element, which is currently only available in a preliminary form, is check-pointing the run, i.e., the possibility to continue the inner and outer loop processes to a later time. This would allow us to execute jobs in a very long period without any HPC time restriction. At the moment, the run-script (see Section 2.3) has to be changed with a few more routines to load the trajectories from an earlier run and to continue it. In an upcoming release, this component will be integrated into the L2L framework.

Finally, we would like to extend the set of optimization techniques with optimizers that have more capabilities. This would be for example a neural network, along the lines of the approach proposed by Andrychowicz et al. (2016). For instance, the network could learn the distribution of the parameter space and predict the next set of parameters. One other interesting direction is to include Bayesian Optimization *via* Bayesian hierarchical modeling. In this case, the parameters are not optimized directly as depicted in this work, instead, uncertainty measures and prediction uncertainty are inferred (Finn et al., 2018; Gordon et al., 2018; Yoon et al., 2018).

In conclusion, with this work, we have presented L2L as a software framework for the hyper-parameter optimization of computing workloads, especially focusing on neuroscience use cases. The flexibility of this framework is designed to support the broad and interdisciplinary nature of brain research and provides easier access to HPC for ML-based optimization tasks.

[7]https://www.tensorflow.org/tensorboard/

[8]https://trac.version.fz-juelich.de/vis/wiki/vnc3d

[9]https://ebrains.eu/

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: The Modified National Institute of Standards and Technology (MNIST) database, http://yann.lecun.com/exdb/mnist/.

AUTHOR CONTRIBUTIONS

AS, AY, WK, and SD-P worked on the design of the framework. AY, AS, SD-P, and WK worked on the implementation. AY, TH, CJ-R, WK, AP, MV, and SD-P implemented the use cases and produced the results reported in the manuscript. All authors conceived of the project, designed the set of use cases, reviewed, contributed, and approved the final version of the manuscript.

FUNDING

The research leading to these results has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements no. 785907 (Human Brain Project SGA2) and 945539 (Human Brain Project SGA3). This research has also been partially funded by the Helmholtz Association through the Helmholtz Portfolio Theme Supercomputing and Modeling for the Human Brain. Open Access publication funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)-491111487.

ACKNOWLEDGMENTS

We would like to thank Dr. Alexander Peyser for his ideas, support and useful input to implement the L2L framework and take it into HPC. We would also like to thank Prof. Wolfgang Maass for his input and feedback during the progress of the project. Finally, we would like to thank the HBP community and collaborators around the learning to learn concept who provided a platform for fruitful discussions, identify requirements and expand the potential of the L2L framework. We acknowledge the use of Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the grant agreement No. 800858. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss/centre.eu)

for funding this project by providing computing time on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fncom.2022.885207/full#supplementary-material

SUPPLEMENTAL DATA

All code used to produce the results in this paper as well as the L2L framework can be accessed in this repository: https://github.com/Meta-optimization/L2L/tree/frontiers_submission.

Installation instructions for the framework can be found in the README file of the repository.

An additional use case using structural plasticity in NEST can be found in the **Supplementary Material**.

REFERENCES

Akar, N. A., Cumming, B., Karakasis, V., Ksters, A., Klijn, W., Peyser, A., et al. (2019). "Arbor - a morphologically-detailed neural network simulation library for contemporary high-performance computing architectures," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (Pavia, Italy), 274–282.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., et al. (2016). "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems* (Barcelona, Spain), 3981–3989.

Antoniou, A., Edwards, H., and Storkey, A. (2018). How to train your MAML. *arXiv preprint arXiv:1810.09502*. doi: 10.48550/arXiv.1810.09502

Bansal, K., Nakuci, J., and Muldoon, S. F. (2018). Personalized brain network models for assessing structure-function relationships. *Curr. Opin Neurobiol.* 52, 42–47. doi: 10.1016/j.conb.2018.04.014

Bergstra, J., and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, 281–305.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*. doi: 10.48550/arXiv.1606.01540

Cao, Y., Chen, T., Wang, Z., and Shen, Y. (2019). "Learning to optimize in swarms," in *Advances in Neural Information Processing Systems, Vol. 32*, eds H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett (Pavia, Italy: Curran Associates, Inc.).

Deco, G., McIntosh, A. R., Shen, K., Hutchison, R. M., Menon, R. S., Everling, S., et al. (2014). Identification of optimal structural connectivity using functional connectivity and neural modeling. *J. Neurosci.* 34, 7910–7916. doi: 10.1523/JNEUROSCI.4423-13.2014

Deepu, R., Spreizer, S., Trensch, G., Terhorst, D., Vennemo, S. B., Mitchell, J., et al. (2021). *NEST 3.1*. Zenodo. doi: 10.5281/zenodo.5508805

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

Finn, C., Abbeel, P., and Levine, S. (2017). "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning* (PMLR), 1126–1135.

Finn, C., and Levine, S. (2017). Meta-learning and universality: deep representations and gradient descent can approximate any learning algorithm. *arXiv:1710.11622 [cs]*. doi: 10.48550/arXiv.1710.11622

Finn, C., Rajeswaran, A., Kakade, S., and Levine, S. (2019). "Online meta-learning," in *International Conference on Machine Learning* (Long Beach, CA: PMLR), 1920–1930.

Finn, C., Xu, K., and Levine, S. (2018). "Probabilistic model-agnostic meta-learning," in *Advances in Neural Information Processing System, vol. 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Curran Associates, Inc.), 1–14. Available online at: https://proceedings.neurips.cc/paper/2018/file/8e2c381d4dd04f1c55093f22c59c3a08-Paper.pdf

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: evolutionary algorithms made easy. *J. Mach. Learn. Res.* 13, 2171–2175. doi: 10.1145/2330784.2330799

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2, 1430. doi: 10.4249/scholarpedia.1430

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. E. (2018). Meta-learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*. doi: 10.48550/arXiv.1805.09921

Gouwens, N. W., Berg, J., Feng, D., Sorensen, S. A., Zeng, H., Hawrylycz, M. J., et al. (2018). Systematic generation of biophysically detailed models for diverse cortical neuron types. *Nat. Commun.* 9, 1–13. doi: 10.1038/s41467-017-02718-3

He, X., Zhao, K., and Chu, X. (2021). AutoML: a survey of the state-of-the-art. *Knowl. Based Syst.* 212, 106622. doi: 10.1016/j.knosys.2020.106622

Heidrich-Meisner, V., and Igel, C. (2008). "Variable metric reinforcement learning methods applied to the noisy mountain car problem," in *Recent Advances in Reinforcement Learning. EWRL 2008. Lecture Notes in Computer Science, vol. 5323*, eds S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko (Berlin; Heidelberg: Springer), 136–150.

Hold-Geoffroy, Y., Gagnon, O., and Parizeau, M. (2014). "Once you SCOOP, no need to fork," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment* (New York, NY: ACM), 60.

Honey, C. J., Sporns, O., Cammoun, L., Gigandet, X., Thiran, J. P., Meuli, R., et al. (2009). Predicting human resting-state functional connectivity from structural connectivity. *Proc. Natl. Acad. Sci. U.S.A.* 106, 2035–2040. doi: 10.1073/pnas.0811168106

Hutter, F., Kotthoff, L., and Vanschoren, J. (Eds.). (2019). *Automated Machine Learning-Methods, Systems, Challenges*. Cham, Switzerland: Springer.

Iglesias, M. A., Law, K. J., and Stuart, A. M. (2013). Ensemble kalman methods for inverse problems. *Inverse Probl.* 29, 045001. doi: 10.1088/0266-5611/29/4/045001

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*. doi: 10.48550/arXiv.1711.09846

Jimenez-Romero, C., and Johnson, J. (2017). SpikingLab: modelling agents controlled by spiking neural networks in netlogo. *Neural Comput. Appl.* 28, 755–764. doi: 10.1007/s00521-016-2398-1

Jordan, J., Ippen, T., Helias, M., Kitayama, I., Sato, M., Igarashi, J., et al. (2018). Extremely scalable spiking neuronal network simulation code: from laptops to exascale computers. *Front. Neuroinform.* 12:2. doi: 10.3389/fninf.2018.00002

Kennedy, J., and Eberhart, R. (1995). "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks, vol. 4* (Perth, WA: IEEE), 1942–1948.

LeCun, Y., Cortes, C., and Burges, C. (2010). *MNIST Handwritten Digit Database. ATandT Labs [Online]*. Available: http://yann. lecun. *com/exdb/mnist* 2:18.

Lein, E. S., Hawrylycz, M. J., Ao, N., Ayres, M., Bensinger, A., Bernard, A., et al. (2007). Genome-wide atlas of gene expression in the adult mouse brain. *Nature* 445, 168–176. doi: 10.1038/nature05453

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*. doi: 10.48550/arXiv.1707.09835

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Malik, H., Iqbal, A., Joshi, P., Agrawal, S., and Bakhsh, F. I. (2021). *Metaheuristic and Evolutionary Computation: Algorithms and Applications*. Cham, Switzerland: Springer.

Okwu, M. O., and Tartibu, L. K. (2020). *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications, volume 927*. Switzerland: Springer Nature.

Oliva, D., Houssein, E. H., and Hinojosa, S. (2021). *Metaheuristics in Machine Learning: Theory and Applications*. Cham, Switzerland: Springer.

Ott, E., and Antonsen, T. M. (2008). Low dimensional behavior of large systems of globally coupled oscillators. *Chaos* 18, 37113. doi: 10.1063/1.2930766

Pehle, C., and Pedersen, J. E. (2021). *Norse - A Deep Learning Library for Spiking Neural Networks*. Available online at: https://norse.ai/docs/.

Rasmussen, D. (2018). NengoDL: combining deep learning and neuromorphic modelling methods. *arXiv 1805.11144:1–22*. doi: 10.48550/arXiv.1805.11144

Ravi, S., and Larochelle, H. (2017). "Optimization as a model for few-shot learning," in *International Conference on Learning Representations (ICLR)* (Toulon, France).

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*. doi: 10.48550/arXiv.1703.03864

Sanz Leon, P., Knock, S. A., Woodman, M. M., Domide, L., Mersmann, J., McIntosh, A. R., et al. (2013). The Virtual Brain: a simulator of primate brain network dynamics. *Front. Neuroinform.* 7:10. doi: 10.3389/fninf.2013.00010

Song, X., Gao, W., Yang, Y., Choromanski, K., Pacchiano, A., and Tang, Y. (2019). Es-maml: simple hessian-free meta learning. *arXiv preprint arXiv:1910.01215*. doi: 10.48550/arXiv.1910.01215

Speck, R., Knobloch, M., Lhrs, S., and Gocht, A. (2021). "Using performance analysis tools for a parallel-in-time integrator," in *Parallel-in-Time Integration Methods, volume 356 of Springer Proceedings in Mathematics and Statistics, Cham 9th Workshop on Parallel-in-Time Integration, online (online), 8 Jun 2020 - 12 Jun 2020* (Cham: Springer International Publishing), 51–80.

Stanley, K., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nat. Mach. Intell.* 1, 24–35. doi: 10.1038/s42256-018-0006-z

Streit, A., Erwin, D., Lippert, T., Mallmann, D., Menday, R., Rambadt, M., et al. (2005). UNICOREfrom project results to production grids. *Adv. Parallel Comput.* 14, 357–376. doi: 10.1016/S0927-5452(05)80018-8

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *ArXiv*, abs/1712.06567. doi: 10.48550/arXiv.1712.06567

Thrun, S., and Pratt, L. (2012). *Learning to Learn*. Cham, Switzerland: Springer Science and Business Media.

Tisue, S., and Wilensky, U. (2004). "Netlogo: a simple environment for modeling complexity," in *International Conference on Complex Systems, vol. 21* (Boston, MA), 16–21.

van der Vlag, M., Woodman, M., Fousek, J., Diaz-Pier, S., Perez Martin, A., Jirsa, V., et al. (2022). RateML: a code generation tool for brain network models (accepted). *Front. Netw. Physiol.* 2:826345. doi: 10.3389/fnetp.2022.826345

Van Geit, W., Gevaert, M., Chindemi, G., Rssert, C., Courcol, J.-D., Muller, E. B., et al. (2016). BluePyOpt: leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. *Front. Neuroinform.* 10:17. doi: 10.3389/fninf.2016.00017

Weidel, P., Duarte, R., and Morrison, A. (2021). Unsupervised learning and clustered connectivity enhance reinforcement learning in spiking neural networks. *Front. Comput. Neurosci.* 15:543872. doi: 10.3389/fncom.2021.543872

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. (2014). Natural evolution strategies. *J. Mach. Learn. Res.* 15, 949–980. doi: 10.48550/arXiv.1106.4487

Wijesinghe, P., Srinivasan, G., Panda, P., and Roy, K. (2019). Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines. *Front. Neurosci.* 13:504. doi: 10.3389/fnins.2019.00504

Wilensky, U. (1997). *Netlogo Ants Model*. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University.

Yegenoglu, A., Krajsek, K., Pier, S. D., and Herty, M. (2020). "Ensemble kalman filter optimizing deep neural networks: an alternative approach to non-performing gradient descent," in *International Conference on Machine Learning, Optimization, and Data Science* (Siena – Tuscany, Italy: Springer), 78–92.

Yoo, A. B., Jette, M. A., and Grondona, M. (2003). "Slurm: simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing* (Seattle, Washington USA: Springer), 44–60.

Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian model-agnostic meta-learning. *Adv. Neural Inf. Process. Syst.* (Montréal, Canada), 31.

Young, K., Wang, B., and Taylor, M. (2019). "Metatrace actor-critic: online step-size tuning by meta-gradient descent for reinforcement learning control," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence Main Track* (Macao, China), 4185–4191.

Zhou, Y., Jin, Y., and Ding, J. (2020). Surrogate-assisted evolutionary search of spiking neural architectures in liquid state machines. *Neurocomputing* 406, 12–23. doi: 10.1016/j.neucom.2020.04.079

Zoph, B., and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv:1611.01578 [cs]*. doi: 10.48550/arXiv.1611.01578

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroinformatics

TECHNOLOGY AND CODE
published: 27 May 2022
doi: 10.3389/fninf.2022.835657

NNMT: Mean-Field Based Analysis Tools for Neuronal Network Models

Moritz Layer[1,2]*, *Johanna Senk*[1], *Simon Essink*[1,2], *Alexander van Meegen*[1,3], *Hannah Bos*[1] *and Moritz Helias*[1,4]

[1] *Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6) and JARA-Institute Brain Structure-Function Relationships (INM-10), Jülich Research Centre, Jülich, Germany,* [2] *RWTH Aachen University, Aachen, Germany,* [3] *Institute of Zoology, Faculty of Mathematics and Natural Sciences, University of Cologne, Cologne, Germany,* [4] *Department of Physics, Faculty 1, RWTH Aachen University, Aachen, Germany*

Mean-field theory of neuronal networks has led to numerous advances in our analytical and intuitive understanding of their dynamics during the past decades. In order to make mean-field based analysis tools more accessible, we implemented an extensible, easy-to-use open-source Python toolbox that collects a variety of mean-field methods for the leaky integrate-and-fire neuron model. The Neuronal Network Mean-field Toolbox (NNMT) in its current state allows for estimating properties of large neuronal networks, such as firing rates, power spectra, and dynamical stability in mean-field and linear response approximation, without running simulations. In this article, we describe how the toolbox is implemented, show how it is used to reproduce results of previous studies, and discuss different use-cases, such as parameter space explorations, or mapping different network models. Although the initial version of the toolbox focuses on methods for leaky integrate-and-fire neurons, its structure is designed to be open and extensible. It aims to provide a platform for collecting analytical methods for neuronal network model analysis, such that the neuroscientific community can take maximal advantage of them.

Keywords: mean-field theory, (spiking) neuronal network, integrate-and-fire neuron, open-source software, parameter space exploration, (hybrid) modeling, python, computational neuroscience

OPEN ACCESS

Edited by:
John David Griffiths, University of Toronto, Canada

Reviewed by:
Caglar Cakan, Technical University of Berlin, Germany
Richard Gast, Max Planck Institute for Human Cognitive and Brain Sciences, Germany

***Correspondence:**
Moritz Layer
m.layer@fz-juelich.de

Received: *14 December 2021*
Accepted: *17 March 2022*
Published: *27 May 2022*

Citation:
Layer M, Senk J, Essink S, van Meegen A, Bos H and Helias M (2022) NNMT: Mean-Field Based Analysis Tools for Neuronal Network Models. Front. Neuroinform. 16:835657. doi: 10.3389/fninf.2022.835657

1. INTRODUCTION

Biological neuronal networks are composed of large numbers of recurrently connected neurons, with a single cortical neuron typically receiving synaptic inputs from thousands of other neurons (Braitenberg and Schüz, 1998; DeFelipe et al., 2002). Although the inputs of distinct neurons are integrated in a complex fashion, such large numbers of weak synaptic inputs imply that average properties of entire populations of neurons do not depend strongly on the contributions of individual neurons (Amit and Tsodyks, 1991). Based on this observation, it is possible to develop analytically tractable theories of population properties, in which the effects of individual neurons are averaged out and the complex, recurrent input to individual neurons is replaced by a self-consistent effective input (reviewed, e.g., in Gerstner et al., 2014). In classical physics terms (e.g., Goldenfeld, 1992), this effective input is called *mean-field*, because it is the self-consistent mean of a *field*, which here is just another name for the input the neuron is receiving. The term *self-consistent* refers to the fact that the population of neurons that receives the effective input is the same that contributes to this very input in a recurrent fashion: the population's output determines its input and vice-versa. The stationary statistics of the effective input therefore can be found in a

self-consistent manner: the input to a neuron must be set exactly such that the caused output leads to the respective input.

Mean-field theories have been developed for many different kinds of synapse, neuron, and network models. They have been successfully applied to study average population firing rates (van Vreeswijk and Sompolinsky, 1996, 1998; Amit and Brunel, 1997b), and the various activity states a network of spiking neurons can exhibit, depending on the network parameters (Amit and Brunel, 1997a; Brunel, 2000; Ostojic, 2014), as well as the effects that different kinds of synapses have on firing rates (Fourcaud and Brunel, 2002; Lindner, 2004; Schuecker et al., 2015; Schwalger et al., 2015; Mattia et al., 2019). They have been used to investigate how neuronal networks respond to external inputs (Lindner and Schimansky-Geier, 2001; Lindner and Longtin, 2005), and they explain why neuronal networks can track external input on much faster time scales than a single neuron could (van Vreeswijk and Sompolinsky, 1996, 1998). Mean-field theories allow studying correlations of neuronal activity (Sejnowski, 1976; Ginzburg and Sompolinsky, 1994; Lindner et al., 2005; Trousdale et al., 2012) and were able to reveal why pairs of neurons in random networks, despite receiving a high proportion of common input, can show low output correlations (Hertz, 2010; Renart et al., 2010; Tetzlaff et al., 2012; Helias et al., 2014), which for example has important implication for information processing. They describe pair-wise correlations in network with spatial organization (Rosenbaum and Doiron, 2014; Rosenbaum et al., 2017; Dahmen et al., 2022) and can be generalized to correlations of higher orders (Buice and Chow, 2013). Mean-field theories were utilized to show that neuronal networks can exhibit chaotic dynamics (Sompolinsky et al., 1988; van Vreeswijk and Sompolinsky, 1996, 1998), in which two slightly different initial states can lead to totally different network responses, which has been linked to the network's memory capacity (Toyoizumi and Abbott, 2011; Schuecker et al., 2018). Most of the results mentioned above have been derived for networks of either rate, binary, or spiking neurons of a linear integrate-and-fire type. But various other models have been investigated with similar tools as well; for example, just to mention a few, Hawkes processes, non-linear integrate-and-fire neurons (Brunel and Latham, 2003; Fourcaud-Trocmé et al., 2003; Richardson, 2007, 2008; Grabska-Barwinska and Latham, 2014; Montbrió et al., 2015), or Kuramoto-type models (Stiller and Radons, 1998; van Meegen and Lindner, 2018). Additionally, there is an ongoing effort showing that many of the results derived for distinct models are indeed equivalent and that those models can be mapped to each other under certain circumstances (Ostojic and Brunel, 2011; Grytskyy et al., 2013; Senk et al., 2020).

Other theories for describing mean population rates in networks with spatially organized connectivity, based on taking a continuum limit, have been developed. These theories, known as neural field theories, have deepened our understanding of spatially and temporally structured activity patterns emerging in cortical networks, starting with the seminal work by Wilson and Cowan (1972, 1973), who investigated global activity patterns, and Amari (1975, 1977), who studied stable localized neuronal activity. They were successfully applied to explain hallucination patterns (Ermentrout and Cowan, 1979; Bressloff et al., 2001), as well as EEG and MEG rhythms (Nunez, 1974; Jirsa and Haken, 1996, 1997). The neural field approach has been used to model working memory (Laing et al., 2002; Laing and Troy, 2003), motion perception (Giese, 2012), cognition (Schöner, 2008), and more; for extensive reviews of the literature, we refer the reader to Coombes (2005), Bressloff (2012), and Coombes et al. (2014).

Clearly, analytical theories have contributed to our understanding of neuronal networks and they provide a plethora of powerful and efficient methods for network model analysis. Comparing the predictions of analytical theories to simulations, experimental data, or other theories necessitates a numerical implementation applicable to various network models, depending on the research question. Such an implementation is often far from straightforward and at times requires investing substantial time and effort. Commonly, such tools are implemented as the need arises, and their reuse is not organized systematically and restricted to within a single lab. This way, not only are effort and costs spent by the neuroscientific community duplicated over and over again, but also are many scientists deterred from taking maximal advantage of those methods although they might open new avenues for investigating their research questions.

In order to make analytical tools for neuronal network model analysis accessible to a wider part of the neuroscientific community, and to create a platform for collecting well-tested and validated implementations of such tools, we have developed the Python toolbox NNMT (Layer et al., 2021), short for Neuronal Network Mean-field Toolbox. We would like to emphasize that NNMT is not a simulation tool; NNMT is a collection of numerically solved mean-field equations that directly relate the parameters of a microscopic network model to the statistics of its dynamics. NNMT has been designed to fit the diversity of mean-field theories, and the key features we are aiming for are modularity, extensibility, and a simple usability. Furthermore, it features an extensive test suite to ensure the validity of the implementations as well as a comprehensive user documentation. The current version of NNMT mainly comprises tools for investigating networks of leaky integrate-and-fire neurons as well as some methods for studying binary neurons and neural field models. The toolbox is open-source and publicly available on GitHub.[1]

In the following, we present the design considerations that led to the structure and implementation of NNMT as well as a representative set of use cases. Section 2 first introduces its architecture. Section 3 then explains its usage by reproducing previously published network model analyses from Schuecker et al. (2015), Bos et al. (2016), Sanzeni et al. (2020), and Senk et al. (2020). Section 4 compares NNMT to other available toolboxes for neuronal network model analysis, discusses its use cases from a more general perspective, indicates current limitations and prospective advancements of NNMT, and explains how new tools can be contributed.

[1] https://github.com/INM-6/nnmt

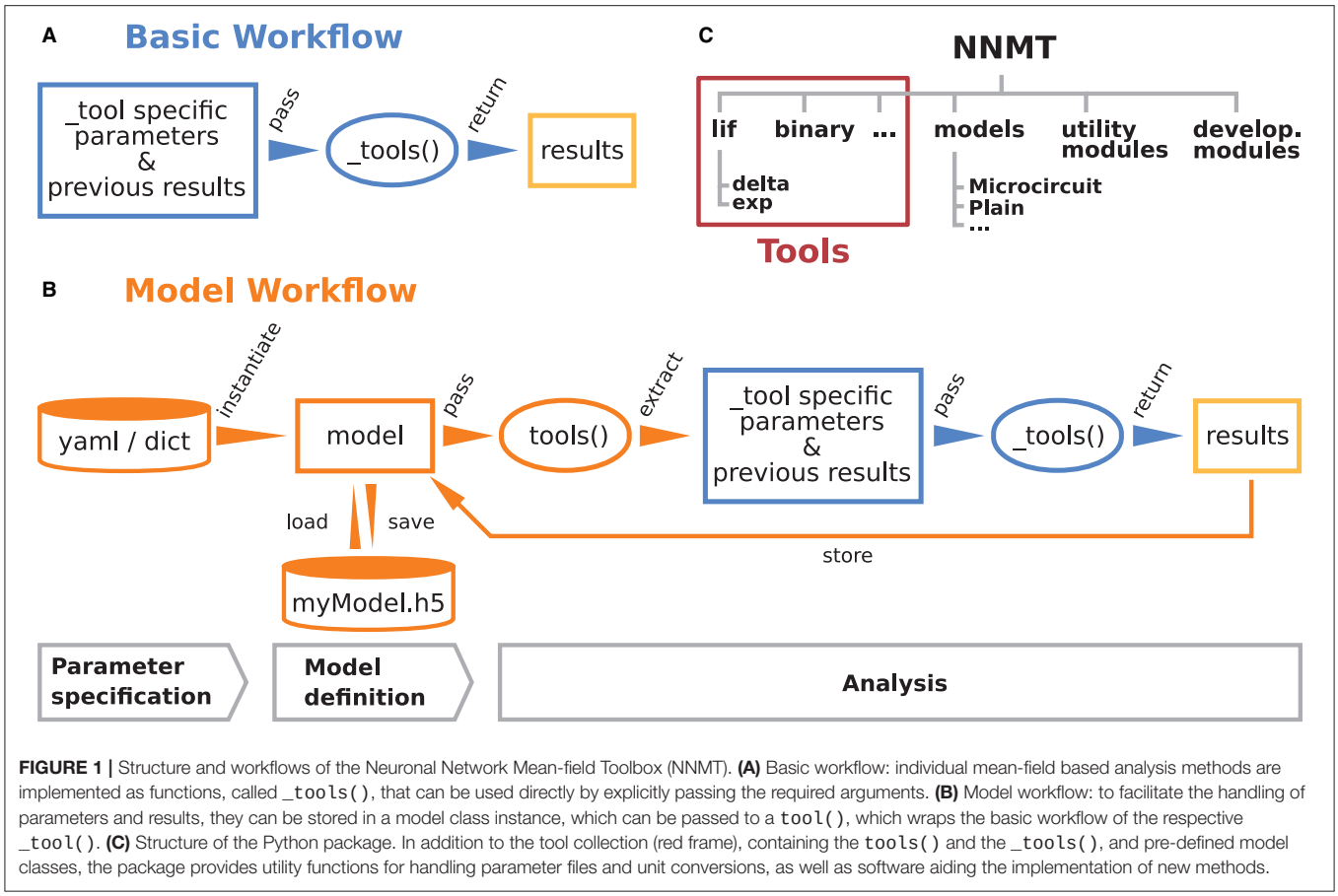


FIGURE 1 | Structure and workflows of the Neuronal Network Mean-field Toolbox (NNMT). **(A)** Basic workflow: individual mean-field based analysis methods are implemented as functions, called `_tools()`, that can be used directly by explicitly passing the required arguments. **(B)** Model workflow: to facilitate the handling of parameters and results, they can be stored in a model class instance, which can be passed to a `tool()`, which wraps the basic workflow of the respective `_tool()`. **(C)** Structure of the Python package. In addition to the tool collection (red frame), containing the `tools()` and the `_tools()`, and pre-defined model classes, the package provides utility functions for handling parameter files and unit conversions, as well as software aiding the implementation of new methods.

```
# basic workflow
result = nnmt.<submodule>.<_tool>(*args, **kwargs)

# model workflow
my_model = nnmt.models.<model>(
    <network_params>, <analysis_params>)
result = nnmt.<submodule>.<tool>(my_model)
```

Listing 1: The two modes of using NNMT: In the basic workflow (top), quantities are calculated by passing all required arguments directly to the underscored tool functions available in the submodules of NNMT. In the model workflow (bottom), a model class is instantiated with parameter sets and the model instance is passed to the non-underscored tool functions which automatically extract the relevant parameters.

2. WORKFLOWS AND ARCHITECTURE

What are the requirements a package for collecting analytical methods for neuronal network model analysis needs to fulfill? To begin with, it should be adaptable and modular enough to accommodate many and diverse analytical methods while avoiding code repetition and a complex interdependency of package components. It should enable the application of the collected algorithms to various network models in a simple and transparent manner. It should make the tools easy to use for new users, while also providing experts with direct access to all parameters and options. Finally, the methods need to be thoroughly tested and well documented.

These are the main considerations that guided the development of NNMT. **Figures 1A,B** illustrate how the toolbox can be used in to two different workflows, depending on the preferences and goals of the user. In the *basic workflow* the individual method implementations called *tools* are directly accessed, whereas the *model workflow* provides additional functionality for the handling of parameters and results.

2.1. Basic Workflow

The core of NNMT is a collection of low-level functions that take specific parameters (or pre-computed results) as input arguments and return analytical results of network properties. In **Figure 1A**, we refer to such basic functions as `_tools()`, as their names always start with an underscore. We term this lightweight approach of directly using these functions the basic workflow. The top part of **Listing 1** demonstrates this usage; for example, the quantity to be computed could be the mean firing rate of a neuronal population and the arguments could be parameters which define neuron model and external drive. While the basic workflow gives full flexibility and direct access to every

parameter of the calculation, it remains the user's responsibility to insert the arguments correctly, e.g., in the right units.

2.2. Model Workflow

The model workflow is a convenient wrapper of the basic workflow (**Figure 1B**). A *model* in this context is an object that stores a larger set of parameters and can be passed directly to a `tool()`, the non-underscored wrapper of the respective `_tool()`. The `tool()` automatically extracts the relevant parameters from the model, passes them as arguments to the corresponding core function `_tool()`, returns the results, and stores them in the model. The bottom part of **Listing 1** shows how a model is initialized with parameters and then passed to a `tool()` function.

Models are implemented as Python classes and can be found in the submodule `nnmt.models`. We provide the class `nnmt.models.Network` as a parent class and a few child classes which inherit the generic methods and properties but are tailored to specific network models; custom models can be created straightforwardly. The parameters distinguish network parameters, which define neuron models and network connectivity, and analysis parameters; an example for an analysis parameter is a frequency range over which a function is evaluated. Upon model instantiation, parameter sets defining values and corresponding units are passed as Python dictionaries or `yaml` files. The model constructor takes care of reading in these parameters, computing dependent parameters from the imported parameters, and converting all units to SI units for internal computations. Consequently, the parameters passed as arguments and the functions for computing dependent parameters of a specific child class need to be aligned. This design encourages a clear separation between a concise set of base parameters and functionality that transforms these parameters to the generic (vectorized) format that the tools work with. To illustrate this, consider the weight matrix of a network of excitatory and inhibitory neuron populations in which all excitatory connections have the same weight and all inhibitory ones another weight. As argument one could pass just a tuple of two different weight values and the corresponding model class would take care of constructing the full weight matrix. This happens in the example presented in Section 3.2.2: The parameter file `network_params_microcircuit.yaml` contains the excitatory synaptic weight and the ratio of inhibitory to excitatory weights. On instantiation, the full weight matrix is constructed from these two parameters, following the rules defined in `nnmt.models.Microcircuit`.

When a `tool()` is called, it checks whether the provided model object contains all required parameters and previously computed results. Then the `tool()` extracts the required arguments, calls the respective `_tool()`, and caches and returns the result. If the user attempts to compute the same property twice, using identical parameters, the `tool()` will retrieve the already computed result from the model's cache and return that value. Results can be exported to an HDF5 file and also loaded.

Using the model workflow instead of the basic workflow comes with the initial overhead of choosing a suitable combination of parameters and a model class, but has the advantages of a higher level of automation with built-in mechanisms for checking correctness of input (e.g., regarding units), reduced redundancy, and the options to store and load results. Both modes of using the toolbox can also be combined.

2.3. Structure of the Toolbox

The structure of the Python package NNMT is depicted in **Figure 1C**. It is subdivided into submodules containing the tools (e.g., `nnmt.lif.exp`, or `nnmt.binary`), the model classes (`nnmt.models`), helper routines for handling parameter files and unit conversions, as well as modules that collect reusable code employed in implementations for multiple neuron models (cf. Section 4.4). The tools are organized in a modular, extensible fashion with a streamlined hierarchy. To give an example, a large part of the currently implemented tools apply to networks of leaky integrate-and-fire (LIF) neurons, and they are located in the submodule `nnmt.lif`. The mean-field theory for networks of LIF neurons distinguishes between neurons with instantaneous synapses, also called delta synapses, and those with exponentially decaying post-synaptic currents. Similarly, the submodule for LIF neurons is split further into the two submodules `nnmt.lif.delta` and `nnmt.lif.exp`. NNMT also collects different implementations for computing the same quantity using different approximations or numerics, allowing for a comparison of different approaches.

Apart from the core package, NNMT comes with an extensive online documentation,[2] including a quickstart tutorial, all examples presented in this paper, a complete documentation of all tools, as well as a guide for contributors.

Furthermore, we provide an extensive test suite that validates the tools by checking them against previously published results and alternative implementations where possible. This ensures that future improvements of the numerics do not break the tools.

3. HOW TO USE THE TOOLBOX

In this section, we demonstrate the practical use of NNMT by replicating a variety of previously published results. The examples presented have been chosen to cover a broad range of common use cases and network models. We include analyses of both stationary and dynamic network features, as mean-field theory is typically divided into two parts: stationary theory, which describes time-independent network properties of systems in a stationary state, and dynamical theory, which describes time-dependent network properties. Additionally, we show how to use the toolbox to map a spiking to a simpler rate model, as well as how to perform a linear stability analysis. All examples, including the used parameter files, are part of the online documentation.[2]

3.1. Installation and Setup

The toolbox can be either installed using pip:

```
pip install nnmt
```

or by installing it directly from the repository, which is described in detail in the online

[2]https://nnmt.readthedocs.io/

documentation. After the installation, the module can be imported:

```
import nnmt
```

3.2. Stationary Quantities

3.2.1. Response Nonlinearities

Networks of excitatory and inhibitory neurons (EI networks, **Figure 2A**) are widely used in computational neuroscience (Gerstner et al., 2014), e.g., to show analytically that a balanced state featuring asynchronous, irregular activity emerges dynamically in a broad region of the parameter space (van Vreeswijk and Sompolinsky, 1996, 1998; Brunel, 2000; Hertz, 2010; Renart et al., 2010). Remarkably, such balance states emerge in inhibition dominated networks for a variety of neuron models if the indegree is large, $K \gg 1$, and the weights scale as $J \propto 1/\sqrt{K}$ (Sanzeni et al., 2020; Ahmadian and Miller, 2021). Furthermore, in a balanced state, a network responds linearly to external input in the limit $K \to \infty$ (van Vreeswijk and Sompolinsky, 1996, 1998; Brunel, 2000; Sanzeni et al., 2020; Ahmadian and Miller, 2021). How do EI networks of LIF neurons respond to external input at finite indegrees? Sanzeni et al. (2020) uncover five different types of nonlinearities in the network response depending on the network parameters. Here, we show how to use the toolbox to reproduce their result (**Figures 2B–F**).

The network consists of two populations, E and I, of identical LIF neurons with instantaneous (*delta*) synapses (Gerstner et al., 2014). The subthreshold dynamics of the membrane potential V_i of neuron i obeys

$$\tau_\mathrm{m} \dot{V}_i = -V_i + RI_i\,, \tag{1}$$

where τ_m denotes the membrane time constant, R the membrane resistance, and I_i the input current. If the membrane potential exceeds a threshold V_th, a spike is emitted and the membrane voltage is reset to the reset potential V_0 and clamped to this value during the refractory time τ_r. After the refractory period, the dynamics continue according to Equation (1). For instantaneous synapses, the input current is given by

$$RI_i(t) = \tau_\mathrm{m} \sum_j J_{ij} \sum_k \delta(t - t_{j,k} - d_{ij})\,, \tag{2}$$

where J_{ij} is the synaptic weight from presynaptic neuron j to postsynaptic neuron i (with $J_{ij} = 0$ if there is no synapse), the $t_{j,k}$ are the spike times of neuron j, and d_{ij} is a synaptic delay (in this example $d_{ij} = d$ for all pairs of neurons). In total, there are N_E and N_I neurons in the respective populations. Each neuron is connected to a fixed number of randomly chosen presynaptic neurons (fixed in-degree); additionally, all neurons receive external input from independent Poisson processes with rate ν_X. The synaptic weights and in-degrees of recurrent and external connections are population-specific:

$$\boldsymbol{J} = \begin{pmatrix} J_\mathrm{EE} & -J_\mathrm{EI} \\ J_\mathrm{IE} & -J_\mathrm{II} \end{pmatrix}, \boldsymbol{J}_\mathrm{ext} = \begin{pmatrix} J_\mathrm{EX} \\ J_\mathrm{IX} \end{pmatrix}, \\ \boldsymbol{K} = \begin{pmatrix} K_\mathrm{EE} & K_\mathrm{EI} \\ K_\mathrm{IE} & K_\mathrm{II} \end{pmatrix}, \boldsymbol{K}_\mathrm{ext} = \begin{pmatrix} K_\mathrm{EX} \\ K_\mathrm{IX} \end{pmatrix}. \tag{3}$$

All weights are positive, implying an excitatory external input.

The core idea of mean-field theory is to approximate the input to a neuron as Gaussian white noise $\xi(t)$ with mean $\langle \xi(t) \rangle = \mu$ and noise intensity $\langle \xi(t)\xi(t') \rangle = \tau_\mathrm{m}\sigma^2\delta(t - t')$. This approximation is well-suited for asynchronous, irregular network states (van Vreeswijk and Sompolinsky, 1996, 1998; Amit and Brunel, 1997b). For a LIF neuron driven by such Gaussian white noise, the firing rate is given by (Siegert, 1951; Tuckwell, 1988; Amit and Brunel, 1997b)

$$\phi(\mu, \sigma) = \left(\tau_\mathrm{r} + \tau_\mathrm{m}\sqrt{\pi} \int_{\widetilde{V}_0(\mu,\sigma)}^{\widetilde{V}_\mathrm{th}(\mu,\sigma)} e^{s^2}(1 + \mathrm{erf}(s))\mathrm{d}s \right)^{-1}, \tag{4}$$

where the rescaled reset- and threshold-voltages are

$$\widetilde{V}_0(\mu, \sigma) = \frac{V_0 - \mu}{\sigma}\,, \qquad \widetilde{V}_\mathrm{th}(\mu, \sigma) = \frac{V_\mathrm{th} - \mu}{\sigma}\,. \tag{5}$$

The first term in Equation (4) is the refractory period and the second term is the mean first-passage time of the membrane voltage from reset to threshold. The mean and the noise intensity of the input to a neuron in a population $a \in \{E, I\}$, which control the mean first-passage time through Equation (5), are determined by (Amit and Brunel, 1997b)

$$\mu_a = \tau_\mathrm{m}(J_{a\mathrm{E}}K_{a\mathrm{E}}\nu_\mathrm{E} - J_{a\mathrm{I}}K_{a\mathrm{I}}\nu_\mathrm{I} + J_{a\mathrm{X}}K_{a\mathrm{X}}\nu_\mathrm{X})\,, \tag{6}$$

$$\sigma_a^2 = \tau_\mathrm{m}(J_{a\mathrm{E}}^2K_{a\mathrm{E}}\nu_\mathrm{E} + J_{a\mathrm{I}}^2K_{a\mathrm{I}}\nu_\mathrm{I} + J_{a\mathrm{X}}^2K_{a\mathrm{X}}\nu_\mathrm{X})\,, \tag{7}$$

respectively, where each term reflects the contribution of one population, with the corresponding firing rates of the excitatory ν_E, inhibitory ν_I, and external population ν_X. Note that we use the letters $i, j, k, \ldots$ to index single neurons and $a, b, c, \ldots$ to index neuronal populations. Both μ_a and σ_a depend on the firing rate of the neurons ν_a, which is in turn given by Equation (4). Thus, one arrives at the self-consistency problem

$$\nu_a = \phi(\mu_a, \sigma_a)\,, \tag{8}$$

which is coupled across the populations due to Equation (6) and Equation (7).

Our toolbox provides two algorithms to solve Equation (8): (1) Integrating the auxiliary ordinary differential equation (ODE) $\dot{\nu}_a = -\nu_a + \phi(\mu_a, \sigma_a)$ with initial values $\nu_a(0) = \nu_{a,0}$ using `scipy.integrate.solve_ivp` (Virtanen et al., 2020) until it reaches a fixed point $\dot{\nu}_a = 0$, where Equation (8) holds by construction. (2) Minimizing the quadratic deviation $\sum_a \left[\nu_a - \phi(\mu_a, \sigma_a)\right]^2$, using the least squares (LSTSQ) solver `scipy.optimize.least_squares` (Virtanen et al., 2020) starting from an initial guess $\nu_{a,0}$. The ODE method is robust to changes in the initial values and hence a good first choice. However, it cannot find self-consistent solutions that correspond to an unstable fixed point of the auxiliary ODE (note that the stability of the auxiliary ODE does not indicate the stability of the solution). To this end, the LSTSQ method can be used. Its drawback is that it needs a good initial guess, because otherwise the found minimum might be a local one where the quadratic deviation does not vanish, $\sum_a \left[\nu_a - \phi(\mu_a, \sigma_a)\right]^2 > 0$, and which

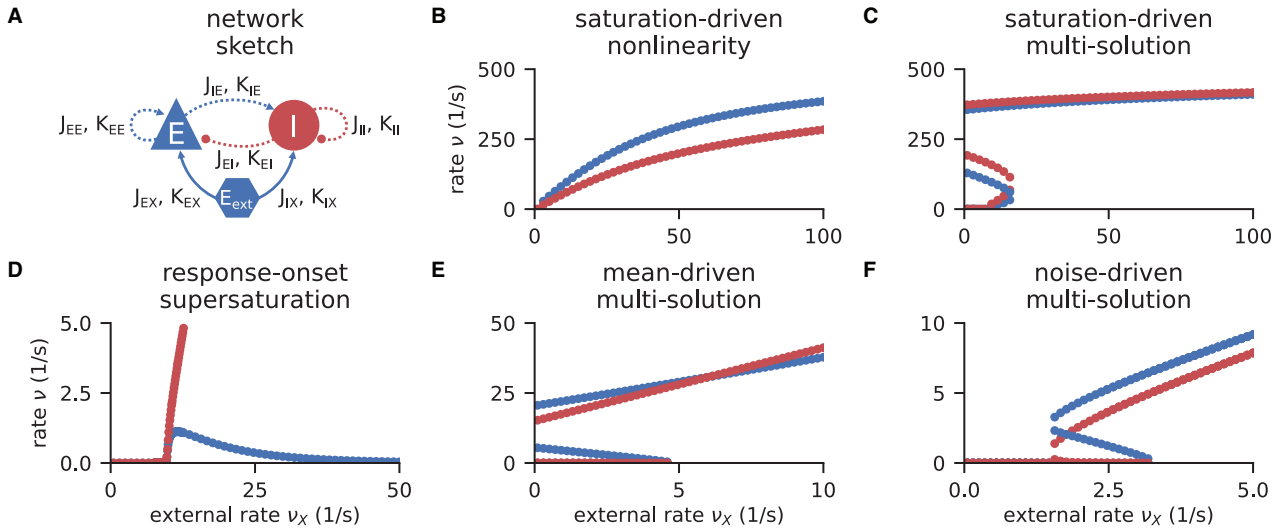


FIGURE 2 | Response nonlinearities in EI-networks. **(A)** Network diagram with nodes and edges according to the graphical notation proposed by Senk et al. (in press). **(B–F)** Firing rate of excitatory (blue) and inhibitory (red) population for varying external input rate ν_X. Specific choices for synaptic weights ($\boldsymbol{J}$, $\boldsymbol{J}_{\text{ext}}$) and in-degrees ($\boldsymbol{K}$, $\boldsymbol{K}_{\text{ext}}$) lead to five types of nonlinearities: **(B)** saturation-driven nonlinearity, **(C)** saturation-driven multi-solution, **(D)** response-onset supersaturation, **(E)** mean-driven multi-solution, and **(F)** noise-driven multi-solution. See Figure 8 in Sanzeni et al. (2020) for parameters.

accordingly does not correspond to a self-consistent solution, $\nu_a \neq \phi(\mu_a, \sigma_a)$. A prerequisite for both methods is a numerical solution of the integral in Equation (4); this is discussed in **Section A.1** in the **Appendix**.

The solutions of the self-consistency problem Equation (8) for varying ν_X and fixed $\boldsymbol{J}$, $\boldsymbol{J}_{\text{ext}}$, $\boldsymbol{K}$, and $\boldsymbol{K}_{\text{ext}}$ reveal the five types of response nonlinearities (**Figure 2**). Different response nonlinearities arise through specific choices of synaptic weights, $\boldsymbol{J}$ and $\boldsymbol{J}_{\text{ext}}$, and in-degrees, $\boldsymbol{K}$ and $\boldsymbol{K}_{\text{ext}}$, which suggests that already a simple EI-network possesses a rich capacity for nonlinear computations. Whenever possible, we use the ODE method and resort to the LSTSQ method only if the self-consistent solution corresponds to an unstable fixed point of the auxiliary ODE. Combining both methods, we can reproduce the first columns of Figure 8 in Sanzeni et al. (2020), where all five types of nonlinearities are presented.

In all cases, we chose appropriate initial values $\nu_{a,0}$ for either method. Note that an exploratory analysis is necessary if the stability properties of a network model are unknown, and potentially multiple fixed points are to be uncovered because there are, to the best of our knowledge, no systematic methods in $d > 1$ dimensions that provide all solutions of a nonlinear system of equations.

In **Listing 2**, we show a minimal example to produce the data shown in **Figure 2B**. After importing the function that solves the self-consistency Equation (8), we collect the neuron and network parameters in a dictionary. Then, we loop through different values for the external rate ν_X and determine the network rates using the ODE method, which is sufficient in this example. In **Listing 2** and to produce **Figure 2B**, we use the basic workflow because only one isolated tool of NNMT (`nnmt.lif.delta._firing_rates()`) is employed, which requires only a few parameters defining the simple EI-network.

```python
import numpy as np
from nnmt.lif.delta import _firing_rates

params = dict(
    # membrane and refractory time constants (in s)
    tau_m=20.*1e-3, tau_r=2.*1e-3,
    # relative reset and threshold potentials (in V)
    V_0_rel=10.*1e-3, V_th_rel=20.*1e-3,
    # recurrent and external weights (in V)
    J=np.array([[0.2, -1.6], [0.2, -1.4]])*1e-3,
    J_ext=np.array([0.2, 0.2])*1e-3,
    # recurrent and external in-degrees
    K=np.array([[400, 100], [400, 100]]),
    K_ext=np.array([1600, 800]),
    # set the method for the fixpoint finder
    fixpoint_method='ODE',
    # initial guess for the firing rate
    nu_0=(0, 0))

# determine self-consistent rates (in 1/s)
nu_ext = np.linspace(1, 100, 50) # external rates (in 1/s)
nu_E, nu_I = np.zeros_like(nu_ext), np.zeros_like(nu_ext)
for i, nu_X in enumerate(nu_ext):
    nu_E[i], nu_I[i] = _firing_rates(nu_ext=nu_X,
                                     **params)
```

Listing 2: Example script to produce the data shown in **Figure 2B** using the ODE method (initial value $\nu_{a,0} = 0$ for population $a \in \{E, I\}$).

3.2.2. Firing Rates of Microcircuit Model

Here we show how to use the model workflow to calculate the firing rates of the cortical microcircuit model by Potjans and Diesmann (2014). The circuit is a simplified point

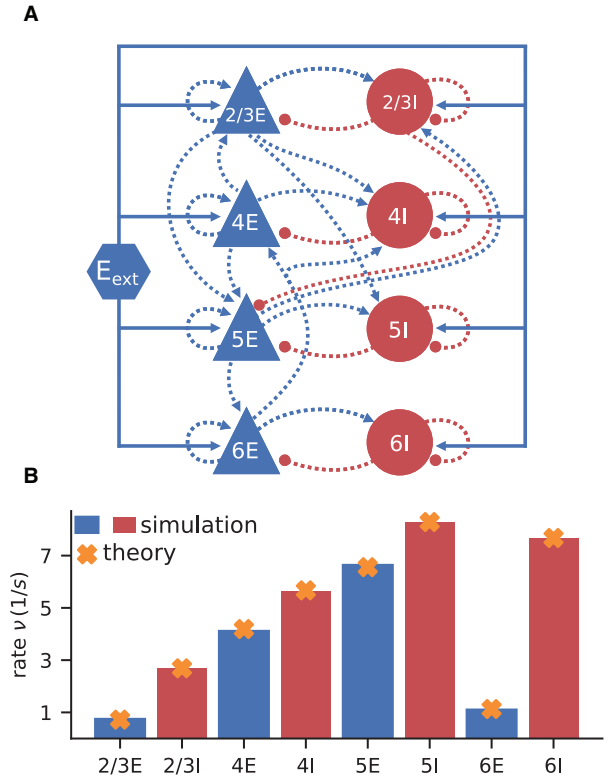


FIGURE 3 | Cortical microcircuit model by Potjans and Diesmann (2014). **(A)** Network diagram (only the strongest connections are shown as in Figure 1 of the original publication). Same notation as in **Figure 2A**. **(B)** Simulation and mean-field estimate for average population firing rates using the parameters from Bos et al. (2016).

neuron network model with biologically plausible parameters, which has been recently used in a number of other works: for example, to study network properties such as layer-dependent attentional processing (Wagatsuma et al., 2011), connectivity structure with respect to oscillations (Bos et al., 2016), and the effect of synaptic weight resolution on activity statistics (Dasbach, Tetzlaff, Diesmann, and Senk, 2021); to assess the performance of different simulator technologies such as neuromorphic hardware (van Albada et al., 2018) and GPUs (Knight and Nowotny, 2018; Golosio et al., 2021); to demonstrate forward-model prediction of local-field potentials from spiking activity (Hagen et al., 2016); and to serve as a building block for large-scale models (Schmidt et al., 2018).

The model consists of eight populations of LIF neurons, corresponding to the excitatory and inhibitory populations of four cortical layers: 2/3E, 2/3I, 4E, 4I, 5E, 5I, 6E, and 6I (see **Figure 3A**). It defines the number of neurons in each population, the number of connections between the populations, the single neuron properties, and the external input. Simulations show that the model yields realistic firing rates for the different populations as observed in particular in the healthy resting-state of early sensory cortex (Potjans and Diesmann, 2014, Table 6).

In contrast to the EI-network model investigated in Section 3.2.1, the neurons in the microcircuit model have exponentially shaped post-synaptic currents: Equation (2) is replaced by Fourcaud and Brunel (2002)

$$\tau_s R \frac{\mathrm{d}I_i}{\mathrm{d}t}(t) = -RI_i(t) + \tau_m \sum_j J_{ij} \sum_k \delta(t - t_{j,k} - d_{ij}), \quad (9)$$

with synaptic time constant τ_s. Note that J_{ij} is a measure in volts here. As discussed in Section 3.2.1, in mean-field theory the second term, representing the neuronal input, is approximated by Gaussian white noise. The additional synaptic filtering leads to the membrane potential (Equation 1) receiving colored noise input. Fourcaud and Brunel (2002) developed a method for calculating the firing rate for this synapse type. They have shown that, if the synaptic time constant τ_s is much smaller than the membrane time constant τ_m, the firing rate for LIF neurons with exponential synapses can be calculated using Equation (4) with shifted integration boundaries

$$\begin{aligned} \widetilde{V}_{\mathrm{cn},0}(\mu,\sigma) &= \widetilde{V}_0(\mu,\sigma) + \frac{\alpha}{2}\sqrt{\frac{\tau_s}{\tau_m}}, \\ \widetilde{V}_{\mathrm{cn,th}}(\mu,\sigma) &= \widetilde{V}_{\mathrm{th}}(\mu,\sigma) + \frac{\alpha}{2}\sqrt{\frac{\tau_s}{\tau_m}}, \end{aligned} \quad (10)$$

with the rescaled reset- and threshold-voltages from Equation (5) and $\alpha = \sqrt{2}\,|\zeta(1/2)| \approx 2.07$, where $\zeta(x)$ denotes the Riemann zeta function; the subscript cn stands for "colored noise".

The microcircuit has been implemented as an NNMT model (`nnmt.models.Microcircuit`). We here use the parameters of the circuit as published in Bos et al. (2016) which is slightly differently parameterized than the original model (see **Table A1** in the **Appendix**). The parameters of the model are specified in a `yaml` file, which uses Python-like indentation and a dictionary-style syntax. List elements are indicated by hyphens, and arrays can be defined as nested lists. Parameters with units can be defined by using the keys `val` and `unit`, whereas unitless variables can be defined without any keys. **Listing 3** shows an example of how some of the microcircuit network parameters used here are defined. Which parameters need to be provided in the yaml file depends on the model used and is indicated in their respective docstrings.

Once the parameters are defined, a microcircuit model is instantiated by passing the respective parameter file to the model constructor; the units are automatically converted to SI units. Then the firing rates are computed. For comparison, we finally load the simulated rates from Bos et al. (2016):

```
# create the network model using a network parameter yaml
# file
microcircuit = nnmt.models.Microcircuit(
    'network_params_microcircuit.yaml')
# calculate firing rates
firing_rates = nnmt.lif.exp.firing_rates(microcircuit)
# load simulated results
simulated_firing_rates = \
    nnmt.input_output.load_h5('Bos2016_rates.h5')['rates']
```

The simulated rates have been obtained by a numerical network simulation (for simulation details see Bos et al., 2016) in which

```yaml
# membrane time constant
tau_m:
  val: 10.0
  unit: ms

# neuron numbers
N:
  - 20683
  - 5834
  - 21915
```

Listing 3: Some microcircuit network parameters defined in a yaml file. A dictionary-like structure with the keys `val` (value) and `unit` is used to define the membrane time constant, which is the same across all populations. The numbers of neurons in each population are defined as a list. Only the numbers for the first three populations are displayed.

the neuron populations are connected according to the model's original connectivity rule: "random, fixed total number with multapses (autapses prohibited)", see Senk et al. (in press) as a reference for connectivity concepts. The term *multapses* refers to multiple connections between the same pair of neurons and *autapses* are self-connections; with this connectivity rule multapses can occur in a network realization but autapses are not allowed. For simplicity, the theoretical predictions assume a connectivity with a fixed in-degree for each neuron. Dasbach et al. (2021) show that simulated spike activity data of networks with these two different connectivity rules are characterized by differently shaped rate distributions ("reference" in their Figures 3d and 4d). In addition, the weights in the simulation are normally distributed while the theory replaces each distribution by its mean; this corresponds to the case $N_{\text{bins}} = 1$ in Dasbach et al. (2021). Nevertheless, our mean-field theoretical estimate of the average population firing rates is in good agreement with the simulated rates (**Figure 3B**).

3.3. Dynamical Quantities

3.3.1. Transfer Function

One of the most important dynamical properties of a neuronal network is how it reacts to external input. A systematic way to study the network response is to apply an oscillatory external input current leading to a periodically modulated mean input $\mu(t) = \mu + \delta\mu\,\mathrm{Re}\left(\mathrm{e}^{\mathrm{i}\omega t}\right)$ (cf. Equation 6), with fixed frequency ω, phase, and amplitude $\delta\mu$, and observe the emerging frequency, phase, and amplitude of the output. If the amplitude of the external input is small compared to the stationary input, the network responds in a linear fashion: it only modifies phase and amplitude, while the output frequency equals the input frequency. This relationship is captured by the input-output transfer function $N(\omega)$ (Brunel and Hakim, 1999; Brunel et al., 2001; Lindner and Schimansky-Geier, 2001), which describes the frequency-dependent modulation of the output firing rate of a neuron population

$$\nu(t) = \nu + \mathrm{Re}\Big(N(\omega)\,\delta\mu\,\mathrm{e}^{\mathrm{i}\omega t}\Big).$$

Note that in this section we only study the linear response to a modulation of the mean input, although in general, a modulation of the noise intensity (Equation 7) can also be included (Lindner and Schimansky-Geier, 2001; Schuecker et al., 2015). The transfer function $N(\omega)$ is a complex function: Its absolute value describes the relative modulation of the firing rate. Its phase, the angle relative to the real axis, describes the phase shift that occurs between input and output. We denote the transfer function for a network of LIF neurons with instantaneous synapses in linear-response approximation as

$$N(\omega) = \frac{\sqrt{2}\nu}{\sigma}\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{m}}}\,\frac{\Phi'_\omega\big|_{\sqrt{2}\widetilde{V}_0}^{\sqrt{2}\widetilde{V}_{\mathrm{th}}}}{\Phi_\omega\big|_{\sqrt{2}\widetilde{V}_0}^{\sqrt{2}\widetilde{V}_{\mathrm{th}}}}, \tag{11}$$

with the rescaled reset- and threshold-voltages $\widetilde{V}_0$ and $\widetilde{V}_{\mathrm{th}}$ as defined in Equation (5) and $\Phi_\omega(x) = e^{\frac{x^2}{4}}\,U\left(\mathrm{i}\omega\tau_{\mathrm{m}} - \frac{1}{2}, x\right)$ using the parabolic cylinder functions $U\left(\mathrm{i}\omega\tau_{\mathrm{m}} - \frac{1}{2}, x\right)$ as defined in (Abramowitz and Stegun, 1974, Section 19.3) and (Olver et al., 2021, Section 12.2). Φ'_ω denotes the first derivative by x. A comparison of our notation and the transfer function given in Schuecker et al. (2015, Equation 29) can be found in **Section A.2.1** in the **Appendix**.

For a neuronal network of LIF neurons with exponentially shaped post-synaptic currents, introduced in Section 3.2.2, Schuecker et al. (2014, 2015) show that an analytical approximation of the transfer function can be obtained by a shift of integration boundaries, akin to Equation (10):

$$N_{\mathrm{cn}}(\omega) = \frac{\sqrt{2}\nu}{\sigma}\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{m}}}\,\frac{\Phi'_\omega\big|_{\sqrt{2}\widetilde{V}_{\mathrm{cn,0}}}^{\sqrt{2}\widetilde{V}_{\mathrm{cn,th}}}}{\Phi_\omega\big|_{\sqrt{2}\widetilde{V}_{\mathrm{cn,0}}}^{\sqrt{2}\widetilde{V}_{\mathrm{cn,th}}}}. \tag{12}$$

To take into account the effect of the synaptic dynamics, we include an additional low-pass filter:

$$N_{\mathrm{cn,s}}(\omega) = N_{\mathrm{cn}}(\omega)\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{s}}}. \tag{13}$$

If the synaptic time constant is much smaller than the membrane time constant ($\tau_{\mathrm{s}} \ll \tau_{\mathrm{m}}$), an equivalent expression for the transfer function is obtained by a Taylor expansion around the original boundaries (cf. Schuecker et al. 2015, Equation 30). The toolbox implements both variants and offers choosing between them by setting the argument `method` of `nnmt.lif.exp.transfer_function` to either `shift` or `taylor`.

Here, we demonstrate how to calculate the analytical "shift version" of the transfer function for different means and noise intensities of the input current (see **Figure 4**) and thereby reproduce Figure 4 in Schuecker et al. (2015).

The crucial parts for producing **Figure 4** using NNMT are shown in **Listing 4** for one example combination of mean and noise intensity of the input current. Instead of using the model workflow with `nnmt.lif.exp.transfer_function`, we here employ the basic workflow, using

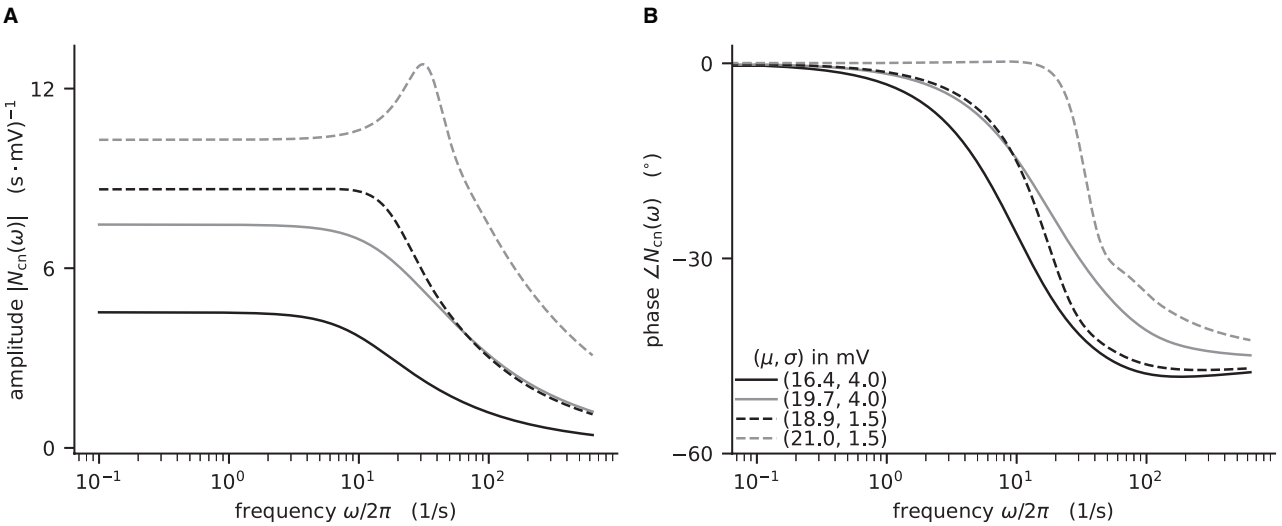


FIGURE 4 | Colored-noise transfer function N_{cn} of LIF model in different regimes. **(A)** Absolute value and **(B)** phase of the "shift" version of the transfer function as a function of the log-scaled frequency. Neuron parameters are set to $V_{th} = 20$ mV, $V_0 = 15$ mv, $\tau_m = 20$ ms, and $\tau_s = 0.5$ ms. For given noise intensities of input current, $\sigma = 4$ mV (solid line) and $\sigma = 1.5$ mV (dashed line), the mean input μ is chosen such that firing rates $\nu = 10$ Hz (black) and $\nu = 30$ Hz (gray) are obtained.

`nnmt.lif.exp._transfer_function` directly. This allows changing the mean input and its noise intensity independently of a network model's structure, but requires two additional steps: First, the necessary parameters are loaded from a `yaml` file, converted to SI units and then stripped off the units using the utility function `nnmt.utils._convert_to_si_and_strip_units`. Second, the analysis frequencies are defined manually. In this example we choose logarithmically spaced frequencies, as we want to plot the results on a log-scale. Finally, the complex-valued transfer function is calculated and then split into its absolute value and phase. **Figure 4** shows that the transfer function acts as a low-pass filter that suppresses the amplitude of high frequency activity, introduces a phase lag, and can lead to resonance phenomena for certain configurations of mean input current and noise intensity.

The replication of the results from Schuecker et al. (2015) outlined here is also used in the integration tests of the toolbox. Note that the implemented analytical form of the transfer function by Schuecker et al. (2015) is an approximation for low frequencies, and deviations from a simulated ground truth are expected for higher frequencies ($\omega/2\pi \gtrsim 100$ Hz at the given parameters).

3.3.2. Power Spectrum

Another frequently studied dynamical property is the power spectrum, which describes how the power of a signal is distributed across its different frequency components, revealing oscillations of the population activity. The power is the Fourier transformed auto-correlation of the population activities (c.f. Bos et al. 2016, Equations 16-18). Linear response theory on top of a mean-field approximation, allows computing the power, dependent on the network architecture, the stationary firing rates, and the neurons' transfer function (Bos et al., 2016). The corresponding analytical expression for the power spectra of population a at angular frequency ω is given by the diagonal elements of the correlation matrix

$$\begin{aligned} P_a(\omega) &= C_{aa}(\omega) \\ &= \left[\left(\mathbf{1} - \widetilde{\mathbf{M}}_{\mathrm{d}}(\omega)\right)^{-1} \operatorname{diag}(\boldsymbol{\nu} \oslash \boldsymbol{n}) \left(\mathbf{1} - \widetilde{\mathbf{M}}_{\mathrm{d}}(-\omega)\right)^{-\mathrm{T}} \right]_{aa}, \end{aligned} \tag{14}$$

with $\oslash$ denoting the elementwise (Hadamard) division, the effective connectivity matrix $\widetilde{\boldsymbol{M}}_{\mathrm{d}}(\omega) = \tau_{\mathrm{m}} \boldsymbol{N}_{\mathrm{cn,s}}(\omega) \cdot \boldsymbol{J} \odot \boldsymbol{K} \odot \boldsymbol{D}(\omega)$, where the dot denotes the scalar product, while $\odot$ denotes the elementwise (Hadamard) product, the mean population firing rates $\boldsymbol{\nu}$, and the numbers of neurons in each population $\boldsymbol{n}$. The effective connectivity combines the static, anatomical connectivity $\boldsymbol{J} \odot \boldsymbol{K}$, represented by synaptic weight matrix $\boldsymbol{J}$ and in-degree matrix $\boldsymbol{K}$, and dynamical quantities, represented by the transfer functions $N_{\mathrm{cn,s},a}(\omega)$ (Equation (13)), and the contribution of the delays in (Equation 13), represented by their Fourier transformed distributions $D_{ab}(\omega)$ (cf. Bos et al. 2016, Equations 14, 15).

The modular structure in combination with the model workflow of this toolbox permits a step-by-step calculation of the power spectra, as shown in **Listing 5**. The inherent structure of the theory is emphasized in these steps: After instantiating the network model class with given network parameters, we determine the working point, which characterizes the statistics of the model's stationary dynamics. It is defined by the population firing rates, the mean, and the standard deviation of the input to a neuron of the respective population. This is necessary for determining the transfer functions. The calculation of the delay distribution matrix is then required for calculating the effective connectivity and to finally get an estimate of the power spectra.

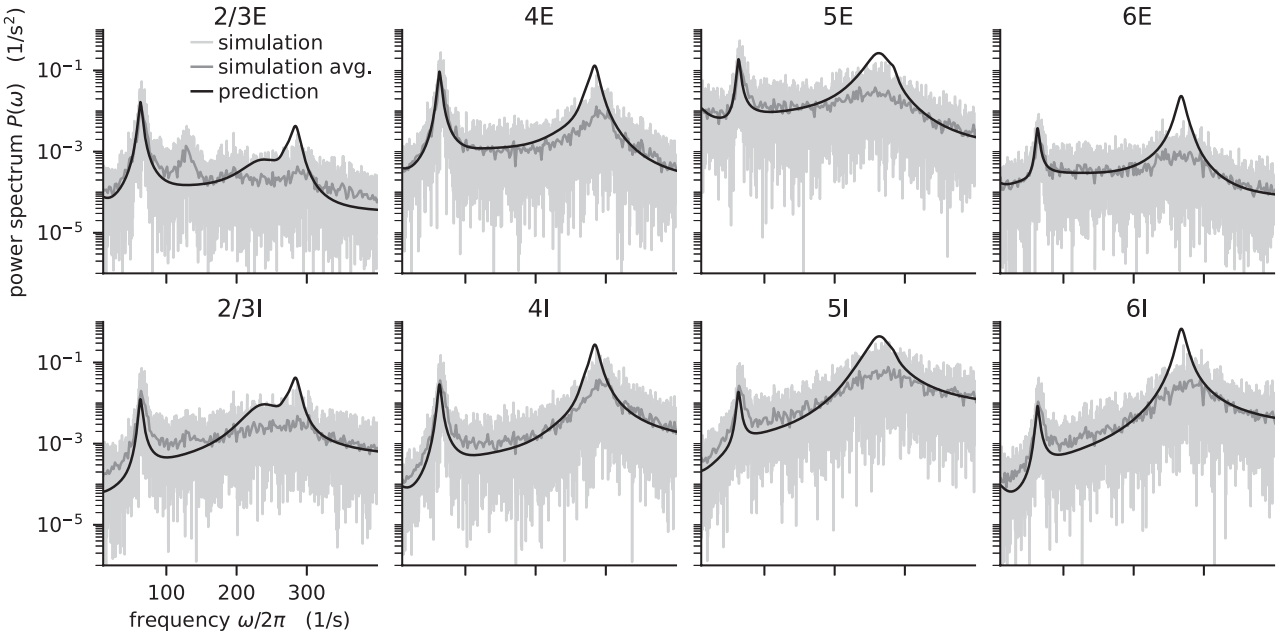


FIGURE 5 | Power spectra of the population spiking activity in the adapted cortical microcircuit from Bos et al. (2016). The spiking activity of each population in a 10 s simulation of the model is binned with 1 ms resolution and the power spectrum of the resulting histogram is calculated by a fast Fourier transform (FFT; light gray curves). In addition, the simulation is split into 500 ms windows, the power spectrum calculated for each window and averaged across windows (gray curves). Black curves correspond to analytical prediction obtained with NNMT as described in **Listing 5**. The panels show the spectra for the excitatory (top) and inhibitory (bottom) populations within each layer of the microcircuit.

Figure 5 reproduces Figure 1E in Bos et al. (2016) and shows the spectra for each population of the adjusted version (see **Table A1** in the **Appendix**) of the microcircuit model.

The numerical predictions obtained from the toolbox largely coincide with simulated data taken from the original publication (Bos et al., 2016) and reveal dominant oscillations of the population activities in the low-γ range around 63 Hz. Furthermore, faster oscillations with peak power around 300 Hz are predicted with higher magnitudes in the inhibitory populations 4I, 5I, and 6I.

The deviation between predicted and simulated power spectra seen at $\sim$ 130 Hz in population 2/3E could be a harmonic of the correctly predicted, prominent 63 Hz peak; a non-linear effect not captured in linear response theory. Furthermore, the systematic overestimation of the power spectrum at large frequencies is explained by the limited validity of the analytical approximation of the transfer function for high frequencies.

3.3.3. Sensitivity Measure

The power spectra shown in the previous section exhibit prominent peaks at certain frequencies, which indicate oscillatory activity. Naturally, this begs the question: which mechanism causes these oscillations? Bos et al. (2016) expose the crucial role that the microcircuit's connectivity plays in shaping the power spectra of this network model. They have developed a method called *sensitivity measure* to directly relate the influence of the anatomical connections, especially the in-degree matrix, on the power spectra.

The power spectrum of the a-th population $P_a(\omega)$ receives a contribution from each eigenvalue λ_b of the effective connectivity matrix, $P_a(\omega) \propto 1/\left(1-\lambda_b(\omega)\right)^2$. Such a contribution consequently diverges as the complex-valued λ_b approaches $1 + 0\mathrm{i}$ in the complex plane, which is referred to as the point of instability. This relation can be derived by replacing the effective connectivity matrix $\widetilde{\mathbf{M}}_\mathrm{d}(\omega)$ in Equation (14) by its eigendecomposition. The sensitivity measure leverages this relationship and evaluates how a change in the in-degree matrix affects the eigenvalues of the effective connectivity and thus indirectly the power spectrum. Bos et al. (2016) introduce a small perturbation α_{cd} of the in-degree matrix, which allows writing the effective connectivity matrix as $\widehat{M}_{ab}(\omega) = (1+\alpha_{cd}\delta_{ca}\delta_{db})\,\widetilde{M}_{ab}(\omega)$, where we dropped the delay subscript d. The sensitivity measure $Z_{b,cd}(\omega)$ describes how the b-th eigenvalue of the effective connectivity matrix varies when the cd-th element of the in-degree matrix is changed

$$Z_{b,cd}(\omega) = \left.\frac{\partial\lambda_b(\omega)}{\partial\alpha_{cd}}\right|_{\alpha_{cd}=0} = \frac{v_{b,c}\widetilde{M}_{cd}u_{b,d}}{\mathbf{v}_b^\mathrm{T}\cdot\mathbf{u}_b}\,, \tag{15}$$

where $\frac{\partial\lambda_b(\omega)}{\partial\alpha_{cd}}$ is the partial derivative of the eigenvalue with respect to a change in connectivity, $\mathbf{v}_b^\mathrm{T}$ and $\mathbf{u}_b$ are the left and right eigenvectors of $\widetilde{\boldsymbol{M}}$ corresponding to eigenvalue $\lambda_b(\omega)$.

The complex sensitivity measure can be understood in terms of two components: $\mathbf{Z}_b^\mathrm{amp}$ is the projection of the matrix $\mathbf{Z}_b$ onto the direction in the complex plane defined by $1-\lambda_b(\omega)$;

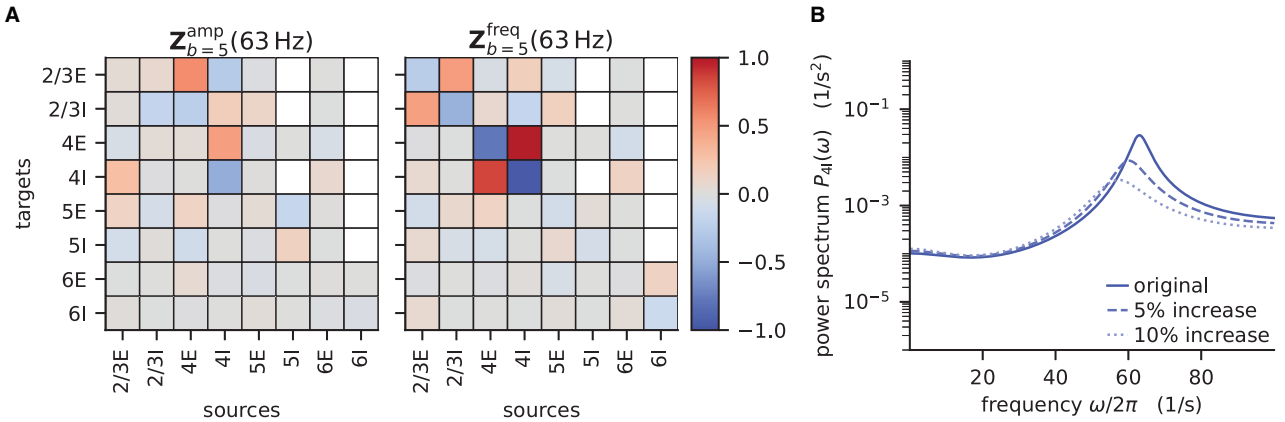


FIGURE 6 | Sensitivity measure at low-γ frequency and corresponding power spectrum of microcircuit with adjusted connectivity. **(A)** Sensitivity measure of one eigenmode of the effective connectivity relevant for low-γ oscillations. The sensitivity measure for this mode is evaluated at the frequency where the corresponding eigenvalue is closest to the point of instability $1 + 0\text{i}$ in complex plane. $\boldsymbol{Z}^{\text{amp}}_b(\omega)$ (left subpanel) visualizes the influence of a perturbation of a connection on the peak amplitude of the power spectrum. $\boldsymbol{Z}^{\text{freq}}_b(\omega)$ (right subpanel) shows the impact on the peak frequency. Non-existent connections are masked white. **(B)** Mean-field prediction of power spectrum of population 4I with original connectivity parameters (solid line), 5% increase (dashed line) and 10% increase (dotted line) in connections $K_{4\text{I}\to 4\text{I}}$. The increase in inhibitory input to population 4I was counteracted by an increase of the excitatory external input $K_{\text{ext}\to 4\text{I}}$ to maintain the working point.

it describes how, when the in-degree matrix is perturbed, the complex-valued $\lambda_b(\omega)$ moves toward or away from the instability $1 + 0\text{i}$, and consequently how the amplitude of the power spectrum at frequency ω increases or decreases. $\mathbf{Z}^{\text{freq}}_b$ is the projection onto the perpendicular direction and thus describes how the peak frequency of the power spectrum changes with the perturbation of the in-degree matrix. For a visualization of these projections, refer to Figure 5B in Bos et al. (2016).

The toolbox makes this intricate measure accessible by supplying two tools: After computing the required working point, transfer function, and delay distribution, the tool `nnmt.lif.exp.sensitivity_measure` computes the sensitivity measure at a given frequency for one specific eigenvalue. By default, this is the eigenvalue which is closest to the instability $1 + 0\text{i}$. To perform the computation, we just need to add one line to **Listing 5**:

```
sensitivity_dict = nnmt.lif.exp.sensitivity_measure(
    microcircuit, frequency)
```

The result is returned in form of a dictionary that contains the sensitivity measure and its projections. The tool `nnmt.lif.exp.sensitivity_measure_all_eigenmodes` wraps that basic function and calculates the sensitivity measure for all eigenvalues at the frequency for which each eigenvalue is closest to instability.

According to the original publication (Bos et al., 2016), the peak around 63 Hz has contributions from one eigenvalue of the effective connectivity matrix. **Figure 6** shows the projections of the sensitivity measure at the frequency for which this eigenvalue is closest to the instability, as illustrated in Figure 4 of Bos et al. (2016). The sensitivity measure returns one value for each connection between populations in the network model. For $\mathbf{Z}^{\text{amp}}_b$ a negative value indicates that increasing the in-degrees of a specific connection causes the amplitude of the power spectrum at the evaluated frequency to drop. If the value is positive, the amplitude is predicted to grow as the in-degrees increase. Similarly, for positive $\mathbf{Z}^{\text{freq}}_b$ the frequency of the peak in the power spectrum shifts toward higher values as in-degrees increase, and vice versa. The main finding in this analysis is that the low-γ peak seems to be affected by excitatory-inhibitory loops in layer 2/3 and layer 4.

To decrease the low-γ peak in the power spectrum, one could therefore increase the 4I to 4I connections (cp. **Figure 6A**):

```
# 5 percent increase
K_new = microcircuit.network_params['K'].copy()
K_new[3,3] = 1001 # originally 953
K_ext_new = microcircuit.network_params['K_ext'].copy()
K_ext_new[3] = 2034 # originally 1900
microcircuit_new = microcircuit.change_parameters(
    {'K': K_new, 'K_ext': K_ext_new})
```

and calculate the power spectrum as in **Listing 5** again to validate the change. Note that a change in connectivity leads to a shift in the working point. We are interested in the impact of the modified connectivity on the fluctuation dynamics at the same working point and thus need to counteract the change in connectivity by adjusting the external input. In the chosen example this is ensured by satisfying $J_{4\text{I}\to 4I}\Delta K_{4\text{I}\to 4I}\nu_{4\text{I}} = -J_{\text{ext}\to 4\text{I}}\Delta K_{\text{ext}\to 4\text{I}}\nu_{\text{ext}}$, which yields $\Delta K_{\text{ext}\to 4\text{I}} = -\frac{J_{4\text{I}\to 4I}\Delta K_{4\text{I}\to 4I}\nu_{4\text{I}}}{J_{\text{ext}\to 4\text{I}}\nu_{\text{ext}}}$.

If several eigenvalues of the effective connectivity matrix influence the power spectra in the same frequency range, adjustments of the connectivity are more involved. This is because a change in connectivity would inevitably affect all eigenvalues simultaneously. Further care has to be taken because the sensitivity measure is subject to the same constraints as the current implementation of the transfer function, which is only valid for low frequencies and enters the sensitivity measure directly.

```python
# load parameters in custom units
params = nnmt.input_output.load_val_unit_dict_from_yaml(
    'Schuecker2015_parameters.yaml')

# convert parameters to SI units
nnmt.utils._convert_to_si_and_strip_units(params)

# define the analysis frequencies
frequencies = np.logspace(
    params['f_start_exponent'],
    params['f_end_exponent'],
    params['n_freqs'])
# add the zero frequency
frequencies = np.insert(frequencies, 0, 0.0)
omegas = 2 * np.pi * frequencies

# extract necessary parameters from params dictionary
mean_input = params['mean_input']
...  # here we leave out similar statements

# calculate the transfer function
transfer_function = nnmt.lif.exp._transfer_function(
    mu, sigma,
    tau_m, tau_s, tau_r,
    V_th_rel, V_0_rel,
    omegas,
    method='shift',
    synaptic_filter=False)

# calculate properties plotted in Schuecker et al. (2015)
absolute_value = np.abs(transfer_function)
phase = np.angle(transfer_function) / 2 / np.pi * 360
```

Listing 4: Example script for computing a transfer function shown in **Figure 4** using the method of shifted integration boundaries.

```python
# create network model microcircuit
microcircuit = nnmt.models.Microcircuit(
    network_params='Bos2016_network_params.yaml',
    analysis_params='Bos2016_analysis_params.yaml')

# calculate working point for exponentially shaped post-
    synaptic currents
nnmt.lif.exp.working_point(microcircuit, method='taylor')
# calculate the transfer function
nnmt.lif.exp.transfer_function(microcircuit,
    method='taylor')
# calculate the delay distribution matrix
nnmt.network_properties.delay_dist_matrix(microcircuit)
# calculate the effective connectivity matrix
nnmt.lif.exp.effective_connectivity(microcircuit)
# calculate the power spectra
power_spectra = nnmt.lif.exp.power_spectra(microcircuit)
```

Listing 5: Example script to produce the theoretical prediction (black lines) shown in **Figure 5B**.

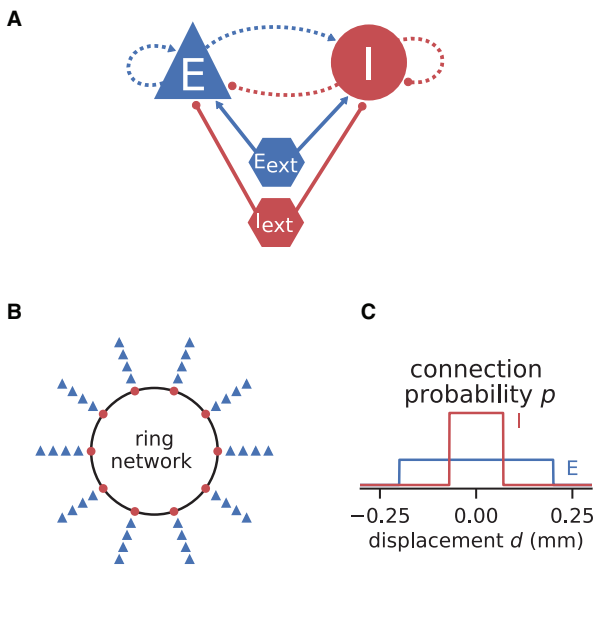


FIGURE 7 | Illustrations of spiking network model by Senk et al. (2020). **(A)** Excitatory and inhibitory neuronal populations randomly connected with fixed in-degree and multapses allowed (autapses prohibited). External excitatory and inhibitory Poisson drive to all neurons. Same notation as in **Figure 2A**. **(B)** One inhibitory and four excitatory neurons per grid position on a one-dimensional domain with periodic boundary conditions (ring network). **(C)** Normalized, boxcar-shaped connection probability with wider excitation than inhibition; the grid spacing is here 10^{-3} mm. For model details and parameters, see Tables II–IV of Senk et al. (2020); the specific values given in the caption of their Figure 6 are used throughout here.

3.4. Fitting Spiking to Rate Model and Predicting Pattern Formation

If the neurons of a network are spatially organized and connected according to a distance-dependent profile, the spiking activity may exhibit pattern formation in space and time, including wave-like phenomena. Senk et al. (2020) set out to scrutinize the non-trivial relationship between the parameters of such a network model and the emerging activity patterns. The model they use is a two-population network of excitatory E and inhibitory I spiking neurons, illustrated in **Figure 7**. All neurons are of type LIF with exponentially shaped post-synaptic currents. The neuron populations are recurrently connected to each other and themselves and they receive additional external excitatory E_{ext} and inhibitory I_{ext} Poisson spike input of adjustable rate as shown in **Figure 7A**. The spatial arrangement of neurons on a ring is illustrated in **Figure 7B** and the boxcar-shaped connectivity profiles in **Figure 7C**.

In the following, we consider a mean-field approximation of the spiking model with spatial averaging, that is a time and space continuous approximation of the discrete model as derived in Senk et al. (2020, Section E. Linearization of spiking network model). We demonstrate three methods used in the original study: First, Section 3.4.1 explains how a model can be brought to a defined state characterized by its working point. The working point is given by the mean μ and noise intensity σ of the input to a neuron, which are both quantities derived from network parameters and require the calculation of the firing rates. With the spiking model in that defined state, Section 3.4.2 then maps its transfer function to the one of a rate model. Section 3.4.3 finally shows that this working-point dependent rate model allows for an analytical linear stability analysis of the network accounting for its spatial structure. This analysis can reveal transitions to spatial and temporal oscillatory states which, when mapped back

to the parameters of the spiking model, may manifest in distinct patterns of simulated spiking activity after a startup transient.

3.4.1. Setting the Working Point by Changing Network Parameters

With network and analysis parameters predefined in `yaml` files, we set up a network model using the example model class `Basic`:

```
space_model = nnmt.models.Basic(
    network_params='Senk2020_network_params.yaml',
    analysis_params='Senk2020_analysis_params.yaml')
```

Upon initialization the given parameters are automatically converted into the format used by NNMT's tools. For instance, relative spike reset and threshold potentials are derived from the absolute values, connection strengths in units of volt are computed from the post-synaptic current amplitudes in ampere, and all values are scaled to SI units.

We aim to bring the network to a defined state by fixing the working point but also want to explore if the procedure of fitting the transfer function still works for different network states. For a parameter space exploration, we use a method to change parameters provided by the model class and scan through a number of different working points of the network. To obtain the required input for a target working point, we adjust the external excitatory and inhibitory firing rates accordingly; NNMT uses a vectorized version of the equations given in Senk et al. (2020, Appendix F: Fixing the working point) to calculate the external rates needed:

```
# relative to spike threshold (in V)
mu = 10. * 1e-3; sigma = 10. * 1e-3
nu_ext = nnmt.lif.exp.external_rates_for_fixed_input(
    space_model, mu_set=mu, sigma_set=sigma)
space_model = space_model.change_parameters(
    changed_network_params={'nu_ext': nu_ext})
```

The implementation uses only one excitatory and one inhibitory Poisson source to represent the external input rates which typically originate from a large number of external source neurons. These two external sources are connected to the network with the same relative inhibition g as used for the internal connections. The resulting external rates for different choices of (μ, σ) are color-coded in the first two plots of **Figure 8A**. The third plot shows the corresponding firing rates of the neurons, which are stored in the results of the model instance when computing the working point explicitly:

```
nnmt.lif.exp.working_point(space_model)
```

Although the external rates are substantially higher than the firing rates, since a neuron is recurrently connected to hundreds of neurons, the total external and recurrent inputs are of the same order.

3.4.2. Parameter Mapping by Fitting the Transfer Function

We map the parameters of the spiking model to a corresponding rate model by, first, computing the transfer function $\boldsymbol{N}_{\mathrm{cn,s}}$ given in Equation (13) of the spiking model, and second, fitting the simpler transfer function of the rate model, for details see Senk et al. (2020, Section F. Comparison of neural-field and spiking models). The dynamics of the rate model can be written as a differential equation for the linearized activity r_a with populations $a, b \in \{\mathrm{E}, \mathrm{I}\}$:

$$\tau \frac{\mathrm{d}}{\mathrm{d}t} r_a(t) = -r_a(t) + \sum_b w_b r_b(t-d) \tag{16}$$

with the delay d; τ is the time constant and w_b are the unitless weights that only depend on the presynaptic population. The transfer function is just the one of a low-pass filter, $N_{\mathrm{LP}} = 1/(1+\lambda\tau)$, where λ is the frequency in Laplace domain. The tool to fit the transfer function requires that the actual transfer function $\boldsymbol{N}_{\mathrm{cn,s}}$ has been computed beforehand and fits $N_{\mathrm{LP}}\boldsymbol{w}$ to $\tau_{\mathrm{m}}\boldsymbol{N}_{\mathrm{cn,s}} \cdot \boldsymbol{J} \odot \boldsymbol{K}$ for the same frequencies together with τ, $\boldsymbol{w}$, and the combined fit error η:

```
nnmt.lif.exp.transfer_function(space_model)
nnmt.lif.exp.fit_transfer_function(space_model)
```

The absolute value of the transfer function is fitted with non-linear least-squares using the solver `scipy.optimize.curve_fit`. **Figure 8B** illustrates the amplitude and phase of the transfer function and its fit for a few (μ, σ) combinations. The plots of **Figure 8C** show the fitted time constants, the fitted excitatory weight, and the combined fit error. The inhibitory weight is proportional to the excitatory one in the same way as the post-synaptic current amplitudes.

3.4.3. Linear Stability Analysis of Spatially Structured Model With Delay

Sections 3.4.1 and 3.4.2 considered a mean-field approximation of the spiking model without taking space into account. In the following, we assume a spatial averaging of the discrete network depicted in **Figure 7** and introduce the spatial connectivity profiles $p_a(x)$. Changing Equation (16) to the integro-differential equation

$$\tau \frac{\partial}{\partial t} r_a(x,t) = -r_a(x,t) + \sum_b w_b \int_{-\infty}^{\infty} p_b(x-y) r_b(y, t-d)\,\mathrm{d}y \tag{17}$$

yields a neural field model defined in continuous space x. This model lends itself to analytical linear stability analysis, as described in detail in Senk et al. (2020, Section A. Linear stability analysis of a neural-field model). In brief, we analyze the stability of a fixed-point solution to this differential equation system which, together with parameter continuation methods and bifurcation analysis, allows us to determine points in parameter space where transitions from homogeneous steady states to oscillatory states can occur. These transitions are given as a function of a bifurcation parameter, here the constant delay d, which is the same for all connections. The complex-valued, temporal eigenvalue λ of the linearized time-delay system is an indicator for the system's overall stability and can serve as a predictor for temporal oscillations, whereas the spatial oscillations are characterized by the real-valued wave number k. Solutions that relate λ and k with the model parameters are given by a characteristic equation, which in our case reads (Senk et al., 2020, Equation 7):

$$\lambda_B(k) = -\frac{1}{\tau} + \frac{1}{d} W_B\left(c(k)\frac{d}{\tau}\mathrm{e}^{\frac{d}{\tau}}\right), \tag{18}$$

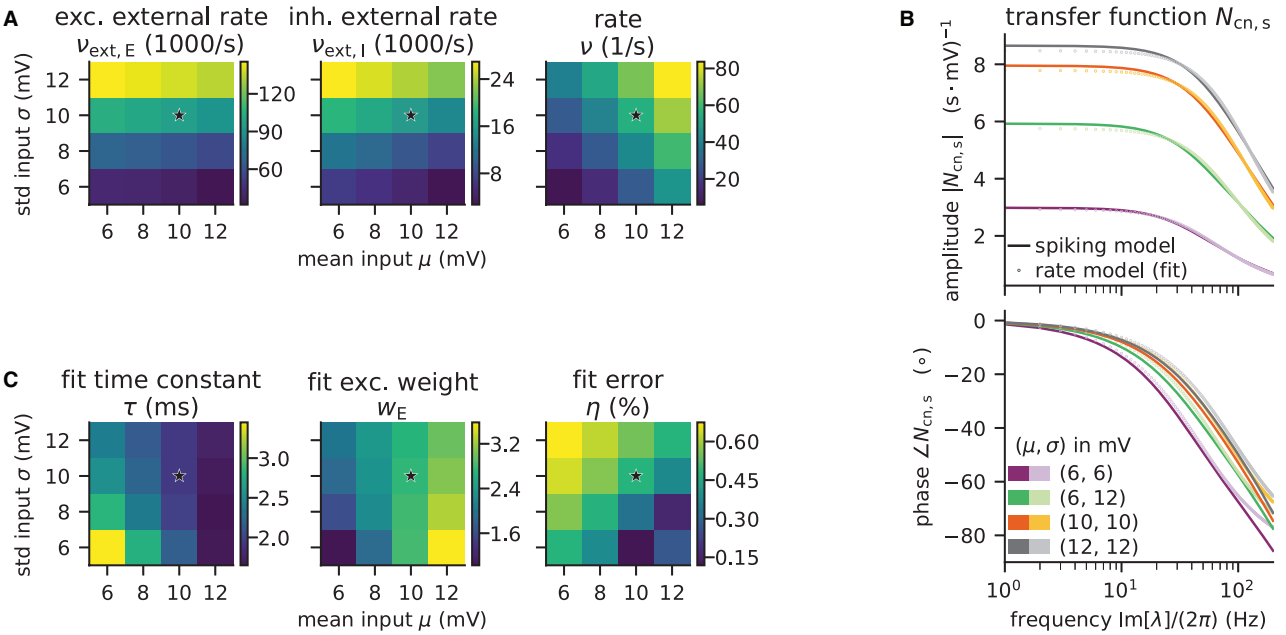


FIGURE 8 | Network parameters and mean-field results from scanning through different working points. Working point (μ, σ) combines mean input μ and noise intensity of input σ. **(A)** External excitatory $\nu_{\text{ext,E}}$ and inhibitory $\nu_{\text{ext,I}}$ Poisson rates required to set (μ, σ) and resulting firing rates ν. **(B)** Transfer function $N_{\text{cn,s}}$ of spiking model and fitted rate-model approximation with low-pass filter for selected (μ, σ) (top: amplitude, bottom: phase). **(C)** Fit results (time constants τ and excitatory weights w_{E}) and fit errors η. The inhibitory weights are $w_{\text{I}} = -gw_{\text{E}}$ with $g = 5$. Star marker in panels **(A)** and **(C)** denotes target working point (10, 10) mV. Similar displays as in Senk et al. (2020, Figure 5).

with the time constant of the rate model τ, the multi-valued Lambert W_B function[3] on branch B (Corless et al., 1996), and the effective connectivity profile $c\left(k\right)$, which combines the weights w_b and the Fourier transforms of the spatial connectivity profiles. Note that the approach generalizes from the boxcar-shaped profiles used here to any symmetric probability density function. NNMT provides an implementation to solve this characteristic equation in its `linear_stability` module using the `spatial` module for the profile:

```
import nnmt.spatial as spatial
import nnmt.linear_stability as linstab

connectivity = (
    W_rate * spatial._ft_spatial_profile_boxcar(
        k_wavenumber,
        space_model.network_params['width']))

eigenvalue = (
    linstab._solve_chareq_lambertw_constant_delay(
        branch_nr, tau_rate,
        space_model.network_params['delay'],
        connectivity))
```

Figure 9A shows that the computed eigenvalues come for the given network parameters in complex conjugate pairs. The branch with the largest real part is the principal branch ($B = 0$). Temporal oscillations are expected to occur if the real part of the eigenvalue on the principal branch becomes positive; the oscillation frequency can then be read off the imaginary part of that eigenvalue. In this example, the largest eigenvalue λ^* on the principal branch has a real part that is just above zero. There exists a supercritical Hopf bifurcation and the delay as the bifurcation parameter is chosen large enough such that the model is just beyond the bifurcation point separating the stable from the instable state. The respective wave number k^* is positive, which indicates spatial oscillations as well. The oscillations in both time and space predicted for the rate model imply that the activity of the corresponding spiking model might exhibit wave trains, i.e., temporally and spatially periodic patterns. The predicted propagation speed of the wave trains is given by the phase velocity Im $[\lambda^*]$ $/k^*$.

To determine whether the results obtained with the rate model are transferable to the spiking model, **Figure 9B** interpolates the analytical solutions of the rate model [$\alpha = 0$, evaluating Equation (18)] to solutions of the spiking model ($\alpha = 1$, accounting for the transfer function $\boldsymbol{N}_{\text{cn,s}}$), which can only be computed numerically. Thus, the parameter α interpolates between the characteristic equations of these two models which primarily differ in their transfer function; for details see Senk et al. (2020, Section F.2 Linear interpolation between the transfer functions). Since the eigenvalues estimated this way show only little differences between rate and spiking model, we conclude that predictions from the rate model should hold also for the spiking model in the parameter regime tested. Following the

[3]The Lambert W_B function is defined as $z = W_B(z)\,\mathrm{e}^{W_B(z)}$ for $z \in \mathbb{C}$ and has infinitely many solutions, numbered by the branches B.

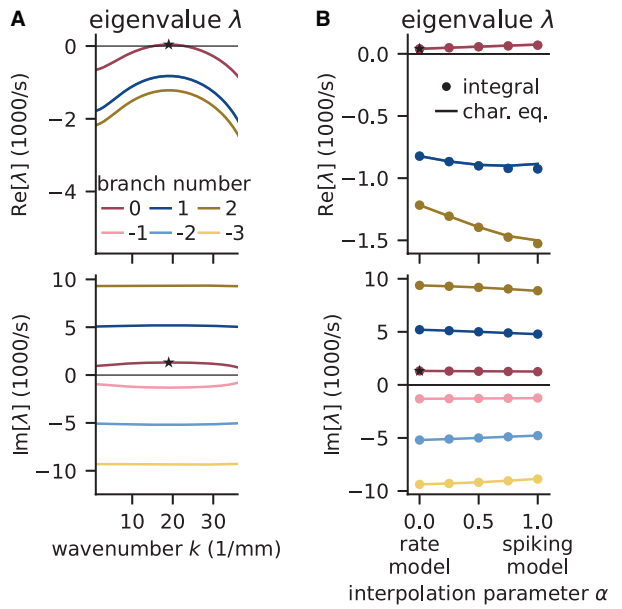


FIGURE 9 | Linear stability analysis of spatially structured network model. **(A)** Analytically exact solution for real (top) and imaginary (bottom) part of eigenvalue λ vs. wavenumber k using rate model derived by fit of spiking model at working point $(\mu, \sigma) = (10, 10)$ mV. Color-coded branches of Lambert W_B function; maximum real eigenvalue (star marker) on principal branch ($B = 0$). **(B)** Linear interpolation between rate ($\alpha = 0$) and spiking model ($\alpha = 1$) by numerical integration of Senk et al. (2020, Equation 30) (solid line) and by numerically solving the characteristic equation in Senk et al. (2020, Equation 29) (circular markers). Star markers at same data points as in **(A)**. Similar displays as in Senk et al. (2020, Figure 6).

argument of Senk et al. (2020), the predicted pattern formation could next be tested in a numerical simulation of the discrete spiking network model. Their Figure 7c for the delay $d = 1.5$ ms shows such results with the same parameters as used here; this figure also illustrates transitions from homogeneous states to oscillatory states by changing the delay (panels b, c, and e).

4. DISCUSSION

Mean-field theory grants important insights into the dynamics of neuronal networks. However, the lack of a publicly available numerical implementation for most methods entails a significant initial investment of time and effort prior to any scientific investigations. In this paper, we present the open-source toolbox NNMT, which currently focuses on methods for LIF neurons but is intended as a platform for collecting standard implementations of various neuronal network model analyses based on mean-field theory that have been thoroughly tested and validated by the neuroscientific community (Riquelme and Gjorgjieva, 2021). As use cases, we reproduce known results from the literature: the non-linear relation between the firing rates and the external input of an E-I-network (Sanzeni et al., 2020), the firing rates of a cortical microcircuit model, its response to oscillatory input, its power spectrum, and the identification of the connections that predominantly contribute to the model's low frequency oscillations (Schuecker et al., 2015; Bos et al., 2016), and pattern formation in a spiking network, analyzed by mapping it to a rate model and conducting a linear stability analysis (Senk et al., 2020).

In the remainder of the discussion, we compare NNMT to other tools suited for network model analysis. We expand on the different use cases of NNMT and also point out the inherent limitations of analytical methods for neuronal network analysis. We conclude with suggestions on how new tools can be added to NNMT and how the toolbox may grow and develop in the future.

4.1. Comparison to Other Tools

There are various approaches and corresponding tools that can help to gain a better understanding of a neuronal network model. There are numerous simulators that numerically solve the dynamical equations for concrete realizations of a network model and all its stochastic components, often focusing either on the resolution of single-neurons, for example NEST (Gewaltig and Diesmann, 2007), Brian (Stimberg et al., 2019), or Neuron (Hines and Carnevale, 2001), or on the population level, for example TheVirtualBrain (Sanz Leon et al., 2013). Similarly, general-purpose dynamical system software like XPPAUT (Ermentrout, 2002) can be used. Simulation tools, like DynaSim (Sherfey et al., 2018), come with enhanced functionality for simplifying batch analysis and parameter explorations. All these tools yield time-series of activity, and some of them even provide the methods for analyzing the generated data. However, simulations only indirectly link a model's parameters with its activity: to gain an understanding of how a model's parameters influence the statistics of their activity, it is necessary to run many simulations with different parameters and analyze the generated data subsequently.

Other approaches provide a more direct insight into a model's behavior on an abstract level: TheVirtualBrain and the Brain Dynamics Toolbox (Heitmann et al., 2018), for example, allow plotting a model's phase space vector field while the parameters can be changed interactively, allowing for exploration of low-dimensional systems defined by differential equations without the need for simulations. XPPAUT has an interface to AUTO-07P (Doedel and Oldeman, 1998), a software for performing numerical bifurcation and continuation analysis. It is worth noting that such tools are limited to models that are defined in terms of differential equations. Models specified in terms of update rules, such as binary neurons, need to be analyzed differently, for example using mean-field theory.

A third approach is to simplify the model analytically and simulate the simplified version. The simulation platform DiPDE[4] utilizes the population density approach to simulate the statistical evolution of a network model's dynamics. Schwalger et al. (2017) start from a microscopic model of generalized integrate-and-fire neurons and derive mesoscopic mean-field population equations, which reproduce the statistical and qualitative behavior of the homogeneous neuronal sub-populations. Similarly, Montbrió et al. (2015) derive a set of non-linear

[4]http://alleninstitute.github.io/dipde

differential equations describing the dynamics of the rate and mean membrane potentials of a population of quadratic integrate-and-fire (QIF) neurons. The simulation platform PyRates (Gast et al., 2019) provides an implementation of this QIF mean-field model, and allows simulating them to obtain the temporal evolution of the population activity measures.

However, mean-field and related theories can go beyond such reduced dynamical equations: they can directly link model parameters to activity statistics, and they can even provide access to informative network properties that might not be accessible otherwise. The spectral bound (Rajan and Abbott, 2006) of the effective connectivity matrix in linear response theory (Lindner et al., 2005; Pernice et al., 2011; Trousdale et al., 2012) is an example of such a property. It is a measure for the stability of the linearized system and determines, for example, the occurrence of slow dynamics and long-range correlations (Dahmen et al., 2022). Another example is the sensitivity measure presented in Section 3.3.3, which directly links a network model's connectivity with the properties of its power spectrum. These measures are not accessible via simulations. They require analytical calculation.

Similarly, NNMT is not a simulator. NNMT is a collection of mean-field equation implementations that directly relate a model's parameters to the statistics of its dynamics or to other informative properties. It provides these implementations in a format that makes them applicable to as many network models as possible. This is not to say that NNMT does not involve numerical integration procedures; solving self-consistent equations, such as in the case of the firing rates calculations in Section 3.2.1 and Section 3.2.2, is a common task, and a collection of respective solvers is part of NNMT.

4.2. Use Cases

In Section 3, we present concrete examples of how to apply some of the tools available. Here, we revisit some of the examples to highlight the use cases NNMT lends itself to, as well as provide some ideas for how the toolbox could be utilized in future projects.

Analytical methods have the advantage of being fast, and typically they only require a limited amount of computational resources. The computational costs for calculating analytical estimates of dynamical network properties like firing rates, as opposed to the costs of running simulations of a network model, are independent of the number of neurons the network is composed of. This is especially relevant for parameter space explorations, for which many simulations have to be performed. To speed up prototyping, a modeler can first perform a parameter scan using analytical tools from NNMT to get an estimate of the right parameter regimes and subsequently run simulations on this restricted set of parameters to arrive at the final model parameters. An example of such a parameter scan is given in Section 3.2.1, where the firing rates of a network are studied as a function of the external input.

Additionally to speeding up parameter space explorations, analytical methods may guide parameter space explorations in another way: namely, by providing an analytical relation between network model parameters and network dynamics, which allows a targeted adjustment of specific parameters to achieve a desired network activity. The prime example implemented in NNMT is the sensitivity measure presented in Section 3.3.3, which provides an intuitive relation between the network connectivity and the peaks of the power spectrum corresponding to the dominant oscillation frequencies. As shown in the final part of Section 3.3.3, the sensitivity measure identifies the connections which need to be adjusted in order to modify the dominant oscillation mode in a desired manner. This illustrates a mean-field method that provides a modeler with additional information about the origin of a model's dynamics, such that a parameter space exploration can be restricted to the few identified crucial model parameters.

A modeler investigating which features of a network model are crucial for the emergence of certain activity characteristics observed in simulations might be interested in comparing models of differing complexity. The respective mappings can be derived in mean-field theory, and one variant included in NNMT, which is presented in Section 3.4, allows mapping a LIF network to a simpler rate network. This is useful to investigate whether spiking dynamics is crucial for the observed phenomenon.

On a general note, which kind of questions researchers pursue is limited by and therefore depends on the tools they have at hand (Dyson, 2012). The availability of sophisticated neural network simulators for example has lead to the development of conceptually new and more complex neural network models, precisely because their users could focus on actual research questions instead of implementations. We hope that collecting useful implementations of analytical tools for network model analysis will have a similar effect on the development of new tools and that it might lead to new, creative ways of applying them.

4.3. Limitations

As a collection of analytical methods, NNMT comes with inherent limitations that apply to any toolbox for analytical methods: it is restricted to network, neuron, and synapse models, as well as observables, for which a mean-field theory exists, and the tools are based on analytical assumptions, simplifications, and approximations, restricting their valid parameter regimes and their explanatory power, which we expand upon in the next paragraphs.

Analytical methods can provide good estimates of network model properties, but there are limitations that must be considered when interpreting results provided by NNMT: First of all, the employed numerical solvers introduce numerical inaccuracies, but they can be remedied by changing hyperparameters such as integration step sizes or iteration termination thresholds. More importantly, analytical methods almost always rely on approximations, which can only be justified if certain assumptions are fulfilled. Typical examples of such assumptions are fast or slow synapses, or a random connectivity. If such assumptions are not met, at least approximately, and the valid parameter regime of a tool is left, the corresponding method is not guaranteed to give reliable results. Hence, it is important to be aware of a tool's limitations, which we aim to document as thoroughly as possible.

An important assumption of mean-field theory is uncorrelated Poissonian inputs. As discussed in Section 3.2.1, asynchronous irregular activity is a robust feature of inhibition

dominated networks, and mean-field theory is well-suited to describe the activity of such models. However, if a network model features highly correlated activity, or strong external input common to many neurons, approximating the input by uncorrelated noise no longer holds and mean-field estimates become unreliable.

In addition to the breakdown of such assumptions, some approaches, like linear response theory, rely on neglecting higher order terms. This restricts the tools' explanatory power, as they cannot predict higher order effects, such as the presence of higher harmonics in a network's power spectrum. Addressing these deficiencies necessitates using more elaborate analyses, and users should be aware of such limitations when interpreting the results.

Finally, a specific limitation of NNMT is that it currently only collects methods for LIF neurons. However, one of the aims of this paper is to encourage other scientists to contribute to the collection, and we outline how to do so in the following section.

4.4. How to Contribute and Outlook

A toolbox like NNMT always is an ongoing project, and there are various aspects that can be improved. In this section, we briefly discuss how available methods could be improved, what and how new tools could be added, as well as the benefits of implementing a new method with the help of NNMT.

First of all, NNMT in its current state is partly vectorized but the included methods are not parallelized, e.g., using multiprocessing or MPI for Python (`mpi4py`). Vectorization relies on `NumPy` (Harris et al., 2020) and `SciPy` (Virtanen et al., 2020), which are thread-parallel for specific backends, e.g., IntelMKL. With the tools available in the toolbox at the moment, run-time only becomes an issue in extensive parameter scans, for instance, when the transfer function needs to be calculated for a large range of frequencies. To further reduce the runtime, the code could be made fully vectorized. Alternatively, parallelization of many tools in NNMT is straightforward, as many of them include `for` loops over the different populations of a network model and `for` loops over the different analysis frequencies. A third option is just-in-time compilation, as provided by `Numba` (Lam et al., 2015).

Another aspect to consider is the range of network models a tool can be applied to. Thus far, the toolbox primarily supports arbitrary block structured networks. Future developments could extend the class of networks to even more general models.

Due to the research focus at our lab, NNMT presently mainly contains tools for LIF neurons in the fast synaptic regime and networks with random connectivity. Nonetheless, the structure of NNMT allows for adding methods for different neuron types, like for example binary (Ginzburg and Sompolinsky, 1994) or conductance-based neurons (Izhikevich, 2007; Richardson, 2007), as well as more elaborate network models. Another way to improve the toolbox is adding tools that complement the existing ones: As discussed in Section 4.3, many mean-field methods only give valid results for certain parameter ranges. Sometimes, there exist different approximations for the same quantity, valid in complementary parameter regimes. A concrete example is the currently implemented version of the transfer function for leaky integrate-and-fire neurons, based on Schuecker et al. (2015), which gives a good estimate for small synaptic time constants compared to the membrane time constant $\tau_s/\tau_m \ll 1$. A complementary estimate for $\tau_s/\tau_m \gg 1$ has been developed by Moreno-Bote and Parga (2006). Similarly, the current implementation of the firing rates of leaky integrate-and-fire neurons, based on the work of Fourcaud and Brunel (2002), is valid for $\tau_s/\tau_m \ll 1$. Recently, van Vreeswijk and Farkhooi (2019) have developed a method accurate for all combinations of synaptic and membrane time constants.

In the following, we explain how such implementations can be added and how using NNMT helps implementing new methods. Clearly, the implementations of NNMT help implementing methods that build on already existing ones. An example is the firing rate for LIF neurons with exponential synapses `nnmt.lif.exp._firing_rates()` which wraps the calculation of firing rates for LIF neurons with delta synapses `nnmt.lif.delta._firing_rates()`. Additionally, the toolbox may support the implementation of tools for other neuron models. As an illustration, let us consider adding the computation of the mean activity for a network of binary neurons (included in NNMT 1.1.0). We start with the equations for the mean input μ_a, its variance σ_a^2, and the firing rates $\boldsymbol{m}$ (Helias et al., 2014, Equations 4, 6, and 7)

$$\begin{aligned} \mu_a(\boldsymbol{m}) &= \sum_b K_{ab} J_{ab} m_b \,, \\ \sigma_a^2(\boldsymbol{m}) &= \sum_b K_{ab} J_{ab}^2 m_b (1 - m_b) \,, \\ m_a(\mu_a, \sigma_a) &= \frac{1}{2} \mathrm{erfc}\left(\frac{\Theta_a - \mu_a}{\sqrt{2}\sigma_a} \right) , \end{aligned} \tag{19}$$

with indegree matrix K_{ab} from population b to population a, synaptic weight matrix J_{ab}, and firing-threshold Θ_a. The sum $\sum_b$ may include an external population providing input to the model. This set of self-consistent equations has the same structure as the self-consistent equations for the firing rates of a network of LIF neurons, Equation (8): the input statistics are given as functions of the rate, and the rate is given as a function of the input statistics. Therefore, it is possible to reuse the firing rate integration procedure for LIF neurons, providing immediate access to the two different methods presented in Section 3.2.1. Accordingly, it is sufficient to implement Equation (19) in a new submodule `nnmt.binary` and apply the solver provided by NNMT to extend the toolbox to binary neurons.

The above example demonstrates the benefits of collecting analytical tools for network model analysis in a common framework. The more methods and corresponding solvers the toolbox comprises, the easier implementing new methods becomes. Therefore, contributions to the toolbox are highly welcome; this can be done via the standard pull request workflow on GitHub (see the "Contributors guide" of the official documentation of NNMT[2]). We hope that in the future, many scientists will contribute to this collection of analytical methods for neuronal network model analysis, such that, at some point, we will have tools from all parts of mean-field theory

of neuronal networks, made accessible in a usable format to all neuroscientists.

DATA AVAILABILITY STATEMENT

Publicly available datasets were used in this study, and the corresponding sources are cited in the main text. The toolbox's repository can be found at https://github.com/INM-6/nnmt, and the parameter files used in the presented examples can be found in the examples section of the online documentation https://nnmt.readthedocs.io/en/latest/.

AUTHOR CONTRIBUTIONS

HB and MH developed and implemented the code base and the initial version of the toolbox. ML, JS, and SE designed the current version of the toolbox. ML implemented the current version of the toolbox, vectorized and generalized tools, developed and implemented the test suite, wrote the documentation, and created the example shown in Section 3.2.2. AM improved the numerics of the firing rate integration (Methods) and created the example shown in Section 3.2.1. SE implemented integration tests, improved the functions related to the `sensitivity_measure`, and created the examples shown in Section 3.3. JS developed and implemented the tools used in Section 3.4 and created the respective example. ML, JS, SE, AM, and MH wrote this article. All authors approved the submitted version.

FUNDING

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement Nos. 720270 (HBP SGA1), 785907 (HBP SGA2), and 945539 (HBP SGA3), has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 368482240/GRK2416, and has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 491111487. This research was supported by the Joint Lab "Supercomputing and Modeling for the Human Brain".

ACKNOWLEDGMENTS

We would like to thank Jannis Schuecker, who has contributed to the development and implementation of the code base and the initial version of the toolbox, and Angela Fischer, who supported us designing **Figure 1**. Additionally, we would also like to thank our reviewers for the thorough and constructive feedback, which lead to significant improvements.

REFERENCES

Abramowitz, M., and Stegun, I. A. (1974). *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables* (New York: Dover Publications).

Ahmadian, Y., and Miller, K. D. (2021). What is the dynamical regime of cerebral cortex? *Neuron* 109, 3373–3391. doi: 10.1016/j.neuron.2021.07.031

Amari, S.-I. (1975). Homogeneous nets of neuron-like elements. *Biol. Cybern.* 17, 211–220. doi: 10.1007/BF00339367

Amari, S.-I. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biol. Cybern.* 27, 77–87. doi: 10.1007/bf00337259

Amit, D. J., and Brunel, N. (1997a). Dynamics of a recurrent network of spiking neurons before and following learning. *Netw. Comp. Neural Sys.* 8, 373–404. doi: 10.1088/0954-898x_8_4_003

Amit, D. J. and Brunel, N. (1997b). Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cereb. Cortex* 7, 237–252. doi: 10.1093/cercor/7.3.237

Amit, D. J., and Tsodyks, M. V. (1991). Quantitative study of attractor neural network retrieving at low spike rates I: substrate–spikes, rates and neuronal gain. *Network* 2, 259. doi: 10.1088/0954-898X_2_3_003

Bos, H., Diesmann, M., and Helias, M. (2016). Identifying anatomical origins of coexisting oscillations in the cortical microcircuit. *PLOS Comput. Biol.* 12, e1005132. doi: 10.1371/journal.pcbi.1005132

Braitenberg, V. and Schüz, A. (1998). *Cortex: Statistics and Geometry of Neuronal Connectivity, 2nd Edn.* Berlin: Springer-Verlag.

Bressloff, P. C. (2012). Spatiotemporal dynamics of continuum neural fields. *J. Phys. A* 45, 033001. doi: 10.1088/1751-8113/45/3/033001

Bressloff, P. C., Cowan, J. D., Golubitsky, M., Thomas, P. J., and Wiener, M. C. (2001). Geometric visual hallucinations, euclidean symmetry and the functional architecture of striate cortex. *Phil. Trans. R. Soc. B* 356, 299–330. doi: 10.1098/rstb.2000.0769

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci.* 8, 183–208. doi: 10.1023/a:1008925309027

Brunel, N., Chance, F. S., Fourcaud, N., and Abbott, L. F. (2001). Effects of synaptic noise and filtering on the frequency response of spiking neurons. *Phys. Rev. Lett.* 86, 2186–2189. doi: 10.1103/physrevlett.86.2186

Brunel, N., and Hakim, V. (1999). Fast global oscillations in networks of integrate-and-fire neurons with low firing rates. *Neural Comput.* 11, 1621–1671. doi: 10.1162/089976699300016179

Brunel, N., and Latham, P. (2003). Firing rate of the noisy quadratic integrate-and-fire neuron. *Neural Comput.* 15, 2281–2306. doi: 10.1162/089976603322362365

Buice, M. A., and Chow, C. C. (2013). Beyond mean field theory: statistical field theory for neural networks. *J. Stat. Mech.* 2013, P03003. doi: 10.1088/1742-5468/2013/03/P03003

Coombes, S. (2005). Waves, bumps, and patterns in neural field theories. *Biol. Cybern.* 93, 91–108. doi: 10.1007/s00422-005-0574-y

Coombes, S., bei Graben, P., Potthast, R., and Wright, J. (2014). *Neural Fields. Theory and Applications.* Berlin; Heidelberg: Springer-Verlag.

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., and Knuth, D. E. (1996). On the lambert w function. *Adv. Comput. Math.* 5, 329–359. doi: 10.1007/BF02124750

Dahmen, D., Layer, M., Deutz, L., Dąbrowska, P. A., Voges, N., von Papen, M., et al. (2022). Global organization of neuronal activity only requires unstructured local connectivity. *eLife* 11, e68422. doi: 10.7554/eLife.68422.sa0

Dasbach, S., Tetzlaff, T., Diesmann, M., and Senk, J. (2021). Dynamical characteristics of recurrent neuronal networks are robust against low synaptic weight resolution. *Front. Neurosci.* 15, 757790. doi: 10.3389/fnins.2021.757790

DeFelipe, J., Alonso-Nanclares, L., and Arellano, J. (2002). Microstructure of the neocortex: comparative aspects. *J. Neurocytol.* 31, 299–316. doi: 10.1023/A:1024130211265

Doedel, E. J., and Oldeman, B. (1998). *Auto-07p: Continuation and Bifurcation Software.* Montreal, QC: Concordia University Canada

Dyson, F. J. (2012). Is science mostly driven by ideas or by tools? *Science* 338, 1426–1427. doi: 10.1126/science.1232773

Ermentrout, B. (2002). *Simulating, Analyzing, and Animating Dynamical Systems: A Guide to Xppaut for Researchers and Students (Software, Environments, Tools)*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Ermentrout, G. B., and Cowan, J. D. (1979). A mathematical theory of visual hallucination patterns. *Biol. Cybern.* 34, 137–150. doi: 10.1007/BF00336965

Fourcaud, N., and Brunel, N. (2002). Dynamics of the firing probability of noisy integrate-and-fire neurons. *Neural Comput.* 14, 2057–2110. doi: 10.1162/089976602320264015

Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., and Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *J. Neurosci.* 23, 11628–11640. doi: 10.1523/JNEUROSCI.23-37-11628.2003

Gast, R., Rose, D., Salomon, C., Möller, H. E., Weiskopf, N., and Knösche, T. R. (2019). Pyrates - a python framework for rate-based neural simulations. *PLoS ONE* 14, e0225900. doi: 10.1371/journal.pone.0225900

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics. From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press.

Gewaltig, M.-O., and Diesmann, M. (2007). NEST (nEural simulation tool). *Scholarpedia* 2, 1430. doi: 10.4249/scholarpedia.1430

Giese, M. A. (2012). *Dynamic Neural Field Theory for Motion Perception, Vol. 469*. Berlin; Heidelberg: Springer Science & Business Media)

Ginzburg, I., and Sompolinsky, H. (1994). Theory of correlations in stochastic neural networks. *Phys. Rev. E* 50, 3171–3191. doi: 10.1103/PhysRevE.50.3171

Goldenfeld, N. (1992). *Lectures on Phase Transitions and the Renormalization Group*. Reading, MA: Perseus books.

Golosio, B., Tiddia, G., Luca, C. D., Pastorelli, E., Simula, F., and Paolucci, P. S. (2021). Fast simulations of highly-connected spiking cortical models using GPUs. *Front. Comput. Neurosci.* 15, 627620. doi: 10.3389/fncom.2021.627620

Grabska-Barwinska, A., and Latham, P. (2014). How well do mean field theories of spiking quadratic-integrate-and-fire networks work in realistic parameter regimes? *J. Comput. Neurosci.* 36, 469–481. doi: 10.1007/s10827-013-0481-5

Grytskyy, D., Tetzlaff, T., Diesmann, M., and Helias, M. (2013). A unified view on weakly correlated recurrent networks. *Front. Comput. Neurosci.* 7, 131. doi: 10.3389/fncom.2013.00131

Hagen, E., Dahmen, D., Stavrinou, M. L., Lindén, H., Tetzlaff, T., van Albada, S. J., et al. (2016). Hybrid scheme for modeling local field potentials from point-neuron networks. *Cereb. Cortex* 26, 4461–4496. doi: 10.1093/cercor/bhw237

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi: 10.1038/s41586-020-2649-2

Heitmann, S., Aburn, M. J., and Breakspear, M. (2018). The brain dynamics toolbox for matlab. *Neurocomputing* 315, 82–88. doi: 10.1016/j.neucom.2018.06.026

Helias, M., Tetzlaff, T., and Diesmann, M. (2014). The correlation structure of local cortical networks intrinsically results from recurrent dynamics. *PLoS Comput. Biol.* 10, e1003428. doi: 10.1371/journal.pcbi.1003428

Hertz, J. (2010). Cross-correlations in high-conductance states of a model cortical network. *Neural Comput.* 22, 427–447. doi: 10.1162/neco.2009.06-08-806

Hines, M. L., and Carnevale, N. T. (2001). NEURON: a tool for neuroscientists. *Neuroscientist* 7, 123–135. doi: 10.1177/107385840100700207

Izhikevich, E. M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA: MIT Press.

Jirsa, V. K., and Haken, H. (1996). Field theory of electromagnetic brain activity. *Phys. Rev. Lett.* 77, 960. doi: 10.1103/PhysRevLett.77.960

Jirsa, V. K., and Haken, H. (1997). A derivation of a macroscopic field theory of the brain from the quasi-microscopic neural dynamics. *Phys. D* 99, 503–526. doi: 10.1016/S0167-2789(96)00166-2

Knight, J. C., and Nowotny, T. (2018). GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Front. Neurosci.* 12, 941. doi: 10.3389/fnins.2018.00941

Laing, C. R., and Troy, W. C. (2003). Two-bump solutions of amari-type models of neuronal pattern formation. *Phys. D* 178, 190–218. doi: 10.1016/S0167-2789(03)00013-7

Laing, C. R., Troy, W. C., Gutkin, B., and Ermentrout, B. G. (2002). Multiple bumps in a neuronal model of working memory. *SIAM J. Appl. Math.* 63, 62–97. doi: 10.1137/s0036139901389495

Lam, S. K., Pitrou, A., and Seibert, S. (2015). "Numba: a llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, Austin, TX, 1–6

Layer, M., Senk, J., Essink, S., van Meegen, A., Bos, H., and Helias, M. (2021). NNMT (1.0.0). *Zenodo*. doi: 10.5281/zenodo.5779548

Lindner, B. (2004). Interspike interval statistics of neurons driven by colored noise. *Phys. Rev. E* 69, 0229011–0229014. doi: 10.1103/PhysRevE.69.022901

Lindner, B., Doiron, B., and Longtin, A. (2005). Theory of oscillatory firing induced by spatially correlated noise and delayed inhibitory feedback. *Phys. Rev. E* 72, 061919. doi: 10.1103/physreve.72.061919

Lindner, B., and Longtin, A. (2005). Effect of an exponentially decaying threshold on the firing statistis of a stochastic integate-and-fire neuron. *J. Theor. Biol.* 232, 505–521. doi: 10.1016/j.jtbi.2004.08.030

Lindner, B., and Schimansky-Geier, L. (2001). Transmission of noise coded versus additive signals through a neuronal ensemble. *Phys. Rev. Lett.* 86, 2934–2937. doi: 10.1103/physrevlett.86.2934

Mattia, M., Biggio, M., Galluzzi, A., and Storace, M. (2019). Dimensional reduction in networks of non-markovian spiking neurons: Equivalence of synaptic filtering and heterogeneous propagation delays. *PLoS Comput. Biol.* 15, e1007404. doi: 10.1371/journal.pcbi.1007404

Montbrió, E., Pazó, D., and Roxin, A. (2015). Macroscopic description for networks of spiking neurons. *Phys Rev X* 5, 021028. doi: 10.1103/PhysRevX.5.021028

Moreno-Bote, R., and Parga, N. (2006). Auto- and crosscorrelograms for the spike response of leaky integrate-and-fire neurons with slow synapses. *Phys. Rev. Lett.* 96, 028101. doi: 10.1103/PhysRevLett.96.028101

Nunez, P. L. (1974). The brain wave equation: a model for the eeg. *Math. Biosci.* 21, 279–297. doi: 10.1016/0025-5564(74)90020-0

Olver, F. W. J., Olde Daalhuis, A. B., Lozier, D. W., Schneider, B. I., Boisvert, R. F., Clark, C. W., et al. (2021). *NIST Digital Library of Mathematical Functions*. Available online at: http://dlmf.nist.gov/

Ostojic, S. (2014). Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons. *Nat. Neurosci.* 17, 594–600. doi: 10.1038/nn.3658

Ostojic, S., and Brunel, N. (2011). From spiking neuron models to linear-nonlinear models. *PLoS Comput. Biol.* 7, e1001056. doi: 10.1371/journal.pcbi.1001056

Pernice, V., Staude, B., Cardanobile, S., and Rotter, S. (2011). How structure determines correlations in neuronal networks. *PLoS Comput. Biol.* 7, e1002059. doi: 10.1371/journal.pcbi.1002059

Potjans, T. C., and Diesmann, M. (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cereb. Cortex* 24, 785–806. doi: 10.1093/cercor/bhs358

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*, 3rd edn. Cambridge University Press.

Rajan, K., and Abbott, L. F. (2006). Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.* 97, 188104. doi: 10.1103/PhysRevLett.97.188104

Renart, A., De La Rocha, J., Bartho, P., Hollender, L., Parga, N., Reyes, A., et al. (2010). The asynchronous state in cortical circuits. *Science* 327, 587–590. doi: 10.1126/science.1179850

Richardson, M. J. E. (2007). Firing-rate response of linear and nonlinear integrate-and-fire neurons to modulated current-based and conductance-based synaptic drive. *Phys. Rev. E* 76, 1–15. doi: 10.1103/PhysRevE.76.021919

Richardson, M. J. E. (2008). Spike-train spectra and network response functions for non-linear integrate-and-fire neurons. *Biol. Cybern.* 99, 381–392. doi: 10.1007/s00422-008-0244-y

Riquelme, J. L., and Gjorgjieva, J. (2021). Towards readable code in neuroscience. *Nat. Rev. Neurosci.* 22, 257–258. doi: 10.1038/s41583-021-00450-y

Rosenbaum, R. and Doiron, B. (2014). Balanced networks of spiking neurons with spatially dependent recurrent connections. *Phys. Rev. X* 4, 021039. doi: 10.1103/PhysRevX.4.021039

Rosenbaum, R., Smith, M. A., Kohn, A., Rubin, J. E., and Doiron, B. (2017). The spatial structure of correlated neuronal variability. *Nat. Neurosci.* 20, 107–114. doi: 10.1038/nn.4433

Sanz Leon, P., Knock, S., Woodman, M., Domide, L., Mersmann, J., McIntosh, A., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinform.* 7, 10. doi: 10.3389/fninf.2013.00010

Sanzeni, A., Histed, M. H., and Brunel, N. (2020). Response nonlinearities in networks of spiking neurons. *PLOS Comput. Biol.* 16, e1008165. doi: 10.1371/journal.pcbi.1008165

Schmidt, M., Bakker, R., Hilgetag, C. C., Diesmann, M., and van Albada, S. J. (2018). Multi-scale account of the network structure of macaque visual cortex. *Brain Struct. Func.* 223, 1409–1435. doi: 10.1007/s00429-017-1554-4

Schöner, G. (2008). "Dynamical systems approaches to cognition," in *Cambridge Handbook of Computational Cognitive Modeling*. Cambridge: Cambridge University Press, 101–126.

Schuecker, J., Diesmann, M., and Helias, M. (2014). Reduction of colored noise in excitable systems to white noise and dynamic boundary conditions. *arXiv[Preprint].arXiv:1410.8799.* doi: 10.48550/arXiv.1410.8799

Schuecker, J., Diesmann, M., and Helias, M. (2015). Modulated escape from a metastable state driven by colored noise. *Phys. Rev. E* 92, 052119. doi: 10.1103/PhysRevE.92.052119

Schuecker, J., Goedeke, S., and Helias, M. (2018). Optimal sequence memory in driven random networks. *Phys. Rev. X* 8, 041029. doi: 10.1103/PhysRevX.8.041029

Schwalger, T., Deger, M., and Gerstner, W. (2017). Towards a theory of cortical columns: From spiking neurons to interacting neural populations of finite size. *PLoS Comput. Biol.* 13, e1005507. doi: 10.1371/journal.pcbi.1005507

Schwalger, T., Droste, F., and Lindner, B. (2015). Statistical structure of neural spiking under non-poissonian or other non-white stimulation. *J. Comput. Neurosci.* 39, 29. doi: 10.1007/s10827-015-0560-x

Sejnowski, T. (1976). On the stochastic dynamics of neuronal interaction. *Biol. Cybern.* 22, 203–211. doi: 10.1007/BF00365086

Senk, J., Korvasová, K., Schuecker, J., Hagen, E., Tetzlaff, T., Diesmann, M., et al. (2020). Conditions for wave trains in spiking neural networks. *Phys. Rev. Res.* 2, 023174. doi: 10.1103/physrevresearch.2.023174

Senk, J., Kriener, B., Djurfeldt, M., Voges, N., Jiang, H.-J., Schüttler, L., et al. (in press). Connectivity concepts in neuronal network modeling. *PLOS Comput. Biol*.

Sherfey, J. S., Soplata, A. E., Ardid, S., Roberts, E. A., Stanley, D. A., Pittman-Polletta, B. R., et al. (2018). Dynasim: a matlab toolbox for neural modeling and simulation. *Front. Neuroinform.* 12, 10. doi: 10.3389/fninf.2018.00010

Siegert, A. J. (1951). On the first passage time probability problem. *Phys. Rev.* 81, 617–623. doi: 10.1103/PhysRev.81.617

Sompolinsky, H., Crisanti, A., and Sommers, H. J. (1988). Chaos in random neural networks. *Phys. Rev. Lett.* 61, 259–262. doi: 10.1103/PhysRevLett.61.259

Stiller, J., and Radons, G. (1998). Dynamics of nonlinear oscillators with random interactions. *Phys. Rev. E* 58, 1789. doi: 10.1103/PhysRevE.58.1789

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8, e47314. doi: 10.7554/elife.47314

Tetzlaff, T., Helias, M., Einevoll, G. T., and Diesmann, M. (2012). Decorrelation of neural-network activity by inhibitory feedback. *PLOS Comput. Biol.* 8, e1002596. doi: 10.1371/journal.pcbi.1002596

Toyoizumi, T., and Abbott, L. F. (2011). Beyond the edge of chaos: Amplification and temporal integration by recurrent networks in the chaotic regime. *Phys. Rev. E* 84, 051908. doi: 10.1103/PhysRevE.84.051908

Trousdale, J., Hu, Y., Shea-Brown, E., and Josic, K. (2012). Impact of network structure and cellular response on spike time correlations. *PLoS Comput. Biol.* 8, e1002408. doi: 10.1371/journal.pcbi.1002408

Tuckwell, H. C. (1988). *Introduction to Theoretical Neurobiology, Vol. 2* Cambridge: Cambridge University Press.

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Front. Neurosci.* 12, 291. doi: 10.3389/fnins.2018.00291

van Meegen, A., and Lindner, B. (2018). Self-consistent correlations of randomly coupled rotators in the asynchronous state. *Phys. Rev. Lett.* 121, 258302. doi: 10.1103/PhysRevLett.121.258302

van Vreeswijk, C., and Farkhooi, F. (2019). Fredholm theory for the mean first-passage time of integrate-and-fire oscillators with colored noise input. *Phys. Rev. E* 100, 060402. doi: 10.1103/PhysRevE.100.060402

van Vreeswijk, C., and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 1724–1726. doi: 10.1126/science.274.5293.1724

van Vreeswijk, C., and Sompolinsky, H. (1998). Chaotic balanced state in a model of cortical circuits. *Neural Comput.* 10, 1321–1371. doi: 10.1162/089976698300017214

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods* 17, 261–272. doi: 10.1038/s41592-019-0686-2

Wagatsuma, N., Potjans, T. C., Diesmann, M., and Fukai, T. (2011). Layer-dependent attentional processing by top-down signals in a visual cortical microcircuit model. *Front. Comput. Neurosci.* 5, 31. doi: 10.3389/fncom.2011.00031

Wilson, H. R., and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophys. J.* 12, 1 – 24. doi: 10.1016/S0006-3495(72)86068-5

Wilson, H. R., and Cowan, J. D. (1973). A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik* 13, 55–80. doi: 10.1007/BF00288786

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

A. APPENDIX

A.1. Siegert Implementation

Here, we describe how we solve the integral in Equation (4) numerically in a fully vectorized manner. The difficulty in Equation (4), $\phi(\mu,\sigma) = 1/[\tau_r + \tau_m\sqrt{\pi} I(\widetilde{V}_0, \widetilde{V}_{th})]$ where $\widetilde{V}_0 = \widetilde{V}_0(\mu,\sigma)$ and $\widetilde{V}_{th} = \widetilde{V}_{th}(\mu,\sigma)$ are determined by either Equation (5) or Equation (10), is posed by the integral

$$I(\widetilde{V}_0, \widetilde{V}_{th}) = \int_{\widetilde{V}_0}^{\widetilde{V}_{th}} e^{s^2}(1 + \mathrm{erf}(s))\mathrm{d}s. \tag{A1}$$

This integral is problematic due to the multiplication of e^{s^2} and $1 + \mathrm{erf}(s)$ in the integrand which leads to overflow and loss of significance.

To address this, we split the integral into different domains depending on the sign of the integration variable. Furthermore, we use the scaled complementary error function

$$\mathrm{erf}(s) = 1 - e^{-s^2}\mathrm{erfcx}(s) \tag{A2}$$

to extract the leading exponential contribution. Importantly, $\mathrm{erfcx}(s)$ decreases monotonically from $\mathrm{erfcx}(0) = 1$ with power law asymptotics $\mathrm{erfcx}(s) \sim 1/(\sqrt{\pi}s)$, hence it does not contain any exponential contribution. For positive s, the exponential contribution in the prefactor of $\mathrm{erfcx}(s)$ cancels the e^{s^2} factor in the integrand. For negative s, the integrand simplifies even further to $e^{s^2}(1 + \mathrm{erf}(-s)) = \mathrm{erfcx}(s)$ using $\mathrm{erf}(-s) = -\mathrm{erf}(s)$. In addition to $\mathrm{erfcx}(s)$, we employ the Dawson function

$$D(s) = e^{-s^2}\int_0^s e^{r^2}\mathrm{d}r \tag{A3}$$

to solve some of the integrals analytically. The Dawson function has a power law tail, $D(s) \sim 1/(2s)$; hence, it also does not carry an exponential contribution. Both $\mathrm{erfcx}(s)$ and the Dawson function are fully vectorized in `SciPy` (Virtanen et al., 2020).

Any remaining integrals are solved using Gauss–Legendre quadrature (Press et al., 2007). By construction, Gauss–Legendre quadrature of order k solves integrals of polynomials up to degree k on the interval $[-1, 1]$ exactly. Thus, it gives very good results if the integrand is well approximated by a polynomial of degree k. The quadrature rule itself is

$$\int_a^b f(s)\mathrm{d}s \approx \frac{b-a}{2}\sum_{i=1}^{k} w_i f\left(\frac{b-a}{2}u_i + \frac{b+a}{2}\right), \tag{A4}$$

where the u_i are the roots of the Legendre polynomial of order k and the w_i are appropriate weights such that a polynomial of degree k is integrated exactly. We use a fixed order quadrature for which Equation (A4) is straightforward to vectorize to multiple a and b. We determine the order of the quadrature iteratively by comparison with an adaptive quadrature rule; usually, a small order $k = O(10)$ already yields very good results for an $\mathrm{erfcx}(s)$ integrand.

Inhibitory Regime

First, we consider the case where lower and upper bound of the integral are positive, $0 < \widetilde{V}_0 < \widetilde{V}_{th}$. This corresponds to strongly inhibitory mean input. Expressing the integrand in terms of $\mathrm{erfcx}(s)$ and using the Dawson function, we get

$$I_{inh}(\widetilde{V}_0, \widetilde{V}_{th}) = 2e^{\widetilde{V}_{th}^2}D(\widetilde{V}_{th}) - 2e^{\widetilde{V}_0^2}D(\widetilde{V}_0) - \int_{\widetilde{V}_0}^{\widetilde{V}_{th}}\mathrm{erfcx}(s)\mathrm{d}s.$$

The remaining integral is evaluated using Gauss–Legendre quadrature, Equation (A4). We extract the leading contribution $e^{\widetilde{V}_{th}^2}$ from the denominator in Equation (4) and arrive at

$$\phi(\mu,\sigma) = \frac{e^{-\widetilde{V}_{th}^2}}{\tau_r e^{-\widetilde{V}_{th}^2} + \tau_m\sqrt{\pi}\left(e^{-\widetilde{V}_{th}^2}I_{inh}(\widetilde{V}_0, \widetilde{V}_{th})\right)}. \tag{A5}$$

Extracting $e^{\widetilde{V}_{th}^2}$ from the denominator reduces the latter to $2\tau_m\sqrt{\pi}D(\widetilde{V}_{th})$ and exponentially small correction terms (remember $0 < \widetilde{V}_0 < \widetilde{V}_{th}$ because $V_0 < V_{th}$), thereby preventing overflow.

Excitatory Regime

Second, we consider the case where lower and upper bound of the integral are negative, $\widetilde{V}_0 < \widetilde{V}_{th} < 0$. This corresponds to strongly excitatory mean input. In this regime, we change variables $s \to -s$ to make the domain of integration positive. Using $\mathrm{erf}(-s) = -\mathrm{erf}(s)$ as well as $\mathrm{erfcx}(s)$, we get

$$I_{exc}(\widetilde{V}_0, \widetilde{V}_{th}) = \int_{|\widetilde{V}_{th}|}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s.$$

Thus, we evaluate Equation (4) as

$$\phi(\mu,\sigma) = \frac{1}{\tau_r + \tau_m\sqrt{\pi}\int_{|\widetilde{V}_{th}|}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s}. \tag{A6}$$

In particular, there is no exponential contribution involved in this regime.

Intermediate Regime

Last, we consider the remaining case $\widetilde{V}_0 \le 0 \le \widetilde{V}_{th}$. We split the integral at zero and use the previous steps for the respective parts to get

$$I_{interm}(\widetilde{V}_0, \widetilde{V}_{th}) = 2e^{\widetilde{V}_{th}^2}D(\widetilde{V}_{th}) + \int_{\widetilde{V}_{th}}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s.$$

Note that the sign of the second integral depends on whether $|\widetilde{V}_0| > \widetilde{V}_{th}$ (+) or not (−). Again, we extract the leading contribution $e^{\widetilde{V}_{th}^2}$ from the denominator in Equation (4) and arrive at

$$\phi(\mu,\sigma) = \frac{e^{-\widetilde{V}_{th}^2}}{\tau_r e^{-\widetilde{V}_{th}^2} + \tau_m\sqrt{\pi}\left(e^{-\widetilde{V}_{th}^2}I_{interm}(\widetilde{V}_0, \widetilde{V}_{th})\right)}. \tag{A7}$$

As before, extracting $e^{\widetilde{V}_{th}^2}$ from the denominator prevents overflow.

Deterministic Limit

The deterministic limit $\sigma \to 0$ corresponds to $|\widetilde{V}_0|, |\widetilde{V}_{\mathrm{th}}| \to \infty$ for both Equation (5) and Equation (10). In the inhibitory and the intermediate regime, we see immediately that $\phi(\mu, \sigma \to 0) \to 0$ due to the dominant contribution $e^{-\widetilde{V}_{\mathrm{th}}^2}$. In the excitatory regime, we use the asymptotics $\mathrm{erfcx}(s) \sim 1/(\sqrt{\pi}s)$ to get

$$I(\widetilde{V}_0, \widetilde{V}_{\mathrm{th}}) \to \int_{|\widetilde{V}_{\mathrm{th}}|}^{|\widetilde{V}_0|} \frac{1}{\sqrt{\pi}s} \mathrm{d}s = \frac{1}{\sqrt{\pi}} \ln \frac{|\widetilde{V}_0|}{|\widetilde{V}_{\mathrm{th}}|}.$$

Inserting this into Equation (4) yields

$$\phi(\mu, \sigma) \to \begin{cases} \frac{1}{\tau_{\mathrm{r}} + \tau_{\mathrm{m}} \ln \frac{\mu - V_0}{\mu - V_{\mathrm{th}}}} & \text{if } \mu > V_{\mathrm{th}} \\ 0 & \text{otherwise} \end{cases}, \tag{A8}$$

which is the firing rate of a leaky integrate-and-fire neuron driven by a constant input (Gerstner et al., 2014). Thus, this implementation also tolerates the deterministic limit of a very small noise intensity σ.

TABLE A1 | Microcircuit Parameters.

Symbol	Value (Potjans and Diesmann, 2014)	Value (Bos et al., 2016)	Description
$K_{4E,4I}$	795	675	In-degree from 4I to 4E
$K_{4E,ext}$	2100	1780	External in-degree to 4E
$D(\omega)$	none	truncated Gaussian	Delay distribution
$d_e \pm \delta d_e$	1.5 ± 0.75 ms	1.5 ± 1.5 ms	Mean and standard deviation of excitatory delay
$d_i \pm \delta d_i$	0.75 ± 0.375 ms	0.75 ± 0.75 ms	Mean and standard deviation of inhibitory delay

Parameter adaptions used here are introduced by Bos et al. (2016) compared to original microcircuit model. K_{ij} denotes the in-degrees from population j to population i. The delays in the simulated networks were drawn from a truncated Gaussian distribution with the given mean and standard deviation. The mean-field approximation of the microcircuit by Potjans and Diesmann (2014) assumes the delay to be fixed at the mean value, which is specified in the toolbox by setting the parameter `delay_dist` *to* `none`.

A.2. Transfer Function Notations

In Section 3.3.1 we introduce the analytical form of the transfer function implemented in the toolbox. Schuecker et al. (2015), derive a more general form of the transfer function, which includes a modulation of the variance of the input. Here we compare the notation used in Equation (11) to the notation used in Schuecker et al. (2015, Eq. 29).

Schuecker et al. (2015) define the modulations of input mean and variance as

$$\begin{aligned} \mu(t) &= \mu + \epsilon\mu\, \mathrm{e}^{\mathrm{i}\omega t}, \\ \sigma^2(t) &= \sigma^2 + H\sigma^2\, \mathrm{e}^{\mathrm{i}\omega t}, \end{aligned} \tag{A9}$$

and introduce the transfer function in terms of its influence on the firing rate

$$\nu(t)/\nu_0 = 1 + n(\omega)\, \mathrm{e}^{\mathrm{i}\omega t},$$

where ν_0 is the stationary firing rate. Here the transfer function $n(\omega)$ includes contributions of both the modulation of the mean $n_G(\omega) \propto \epsilon$ and the modulation of the variance $n_H(\omega) \propto H$. We write the modulation of the mean as

$$\mu(t) = \mu + \delta\mu\, \mathrm{e}^{\mathrm{i}\omega t},$$

implying that $\delta\mu$ corresponds to $\epsilon\mu$ in Equation (A9). As we only consider the modulation of the mean, the firing rate can be rewritten as

$$\nu(t) = \nu + N(\omega)\, \delta\mu\, \mathrm{e}^{\mathrm{i}\omega t},$$

where we moved the stationary firing rate ν to the right hand side and included it in the definition of the transfer function $N(\omega)$. In the main text we emphasize that $\mu(t)$ and $\nu(t)$ are physical quantities by only considering the real part of complex contributions. Additionally, we swap the voltage boundaries in Equation (11), introducing a canceling sign change in both the numerator and the denominator. This reformulation was chosen to align the presented formula with the implementation in the toolbox.

frontiers | Frontiers in Neuroinformatics

TECHNOLOGY AND CODE
published: 14 June 2022
doi: 10.3389/fninf.2022.883223

A Robust Modular Automated Neuroimaging Pipeline for Model Inputs to TheVirtualBrain

*Noah Frazier-Logue[1,2†], Justin Wang[1†], Zheng Wang[1], Devin Sodums[1,3], Anisha Khosla[1,4], Alexandria D. Samson[1,4], Anthony R. McIntosh[1,2,4,5] and Kelly Shen[1,2]**

[1] Rotman Research Institute, Baycrest, Toronto, ON, Canada, [2] Institute for Neuroscience and Neurotechnology, Simon Fraser University, Burnaby, BC, Canada, [3] Kunin-Lunenfeld Centre for Applied Research and Innovation, Baycrest, Toronto, ON, Canada, [4] Department of Psychology, University of Toronto, Toronto, ON, Canada, [5] Department of Biomedical Physiology and Kinesiology, Simon Fraser University, Burnaby, BC, Canada

TheVirtualBrain, an open-source platform for large-scale network modeling, can be personalized to an individual using a wide range of neuroimaging modalities. With the growing number and scale of neuroimaging data sharing initiatives of both healthy and clinical populations comes an opportunity to create large and heterogeneous sets of dynamic network models to better understand individual differences in network dynamics and their impact on brain health. Here we present TheVirtualBrain-UK Biobank pipeline, a robust, automated and open-source brain image processing solution to address the expanding scope of TheVirtualBrain project. Our pipeline generates connectome-based modeling inputs compatible for use with TheVirtualBrain. We leverage the existing multimodal MRI processing pipeline from the UK Biobank made for use with a variety of brain imaging modalities. We add various features and changes to the original UK Biobank implementation specifically for informing large-scale network models, including user-defined parcellations for the construction of matching whole-brain functional and structural connectomes. Changes also include detailed reports for quality control of all modalities, a streamlined installation process, modular software packaging, updated software versions, and support for various publicly available datasets. The pipeline has been tested on various datasets from both healthy and clinical populations and is robust to the morphological changes observed in aging and dementia. In this paper, we describe these and other pipeline additions and modifications in detail, as well as how this pipeline fits into the TheVirtualBrain ecosystem.

Keywords: magnetic resonance imaging, structural connectivity, functional connectivity, connectome-based modelling, large-scale networks

OPEN ACCESS

Edited by:
Mike Hawrylycz, Allen Institute for Brain Science, United States

Reviewed by:
Seok Jun Hong, Sungkyunkwan University, South Korea
Lester Melie-Garcia, University of Basel, Switzerland

***Correspondence:**
Kelly Shen
kelly_shen@sfu.ca

† These authors have contributed equally to this work and share first authorship

Received: *24 February 2022*
Accepted: *26 May 2022*
Published: *14 June 2022*

Citation:
Frazier-Logue N, Wang J, Wang Z, Sodums D, Khosla A, Samson AD, McIntosh AR and Shen K (2022) A Robust Modular Automated Neuroimaging Pipeline for Model Inputs to TheVirtualBrain. Front. Neuroinform. 16:883223. doi: 10.3389/fninf.2022.883223

INTRODUCTION

Neuroimaging data sharing initiatives have expanded substantially in the last decade. Multimodal data collection initiatives like the Human Connectome Project (HCP; Van Essen et al., 2013), UK Biobank (Sudlow et al., 2015), and Alzheimer's Disease Neuroimaging Initiative (ADNI; Mueller et al., 2005), among others, allow for promising new avenues of neuroscientific research that

connect different scales of measurement across large samples. While many efforts are being made to analyze these large datasets to better understand the inner workings of the brain and, specific to neurological disorders, identify effective biomarkers of disease, their potential for creating large and heterogeneous sets of personalized generative models is not yet fully realized. TheVirtualBrain (TVB) is an open source software platform for large-scale network modeling (Sanz Leon et al., 2013; Sanz-Leon et al., 2015), where models can be personalized to an individual using a wide range of neuroimaging modalities. Creating personalized models in TVB from large multimodal neuroimaging datasets will allow us to not only better understand individual differences in network dynamics but also allow for the interrogation of mechanisms of disease across large and heterogeneous samples.

For modeling large-scale brain networks, TVB requires, as input, a structural connectivity matrix that represents the anatomical wiring of the brain. In humans, this is often derived from anatomical (T1w) and diffusion-weighted magnetic resonance imaging (dMRI) tractography and specified as the long-range connections between brain regions of interest (ROIs). Optional inputs for TVB models include the cortical surface for surface-based models (e.g., Spiegler et al., 2016), and functional data (e.g., BOLD-fMRI responses, M/EEG activity, functional connectivity) for model input (e.g., Schirner et al., 2018) or parameter fitting (e.g., Shen et al., 2019a), parcellated into the same ROIs as the structural connectivity. A software pipeline for processing large datasets for TVB, then, would ideally be automated and able to preprocess multiple imaging modalities into a set of matching parcellated inputs for TVB. Existing popular MRI processing pipelines include fMRIPrep for anatomical and fMRI data (Esteban et al., 2019), and HCP's Minimal Preprocessing Pipeline for anatomical, fMRI and dMRI data (Glasser et al., 2013). HCP's pipeline is especially well-suited for higher resolution images and relies on the FreeSurfer software package (Fischl, 2012) for working with the cortical surface. An existing empirical data processing pipeline already exists for processing anatomical, fMRI and dMRI data for TVB inputs, and also relies on FreeSurfer-generated surfaces (Schirner et al., 2015).

Data re-use of publicly-available datasets offers great promise for improving both accessibility and replicability. Within the scope of connectome-based modeling, these data also present the opportunity to generate models that capture a population-level understanding that no single empirical dataset can offer. However, considerations for data processing and analysis of data acquired using older protocols and in special populations are warranted. For example, a user may wish to avoid the projection of lower resolution data (e.g., fMRI) to cortical surface vertices (Alfaro-Almagro et al., 2018). With data from aging and clinical populations, FreeSurfer tissue-class segmentations can also be inaccurate and may require manual intervention (McCarthy et al., 2015; Henschel et al., 2020; Srinivasan et al., 2020), something that is not reasonably feasible with large samples. Moreover, with automated processing, a quality control (QC) workflow that detects processing inaccuracies is also needed. This is especially important for aging and clinical datasets where inaccuracies in preprocessing MRI data are common due to differences in brain morphology and image contrast. The HCP pipeline can be used with an fMRI QC pipeline that computes summary statistics to capture signal quality and subject motion of fMRI scans (Marcus et al., 2013). QC of other imaging modalities (e.g., T1w, dMRI) processed with the HCP pipeline relies on extensive manual inspection of raw and processed images. MRIQC (Esteban et al., 2017) is an fMRIPrep-compatible software package that computes image-based metrics for raw or minimally-processed T1w and fMRI data. It outputs a set of HTML-based reports of the individual and group-wise summary metrics to allow identification of outlier images. MRIQC also offers an automated pass-fail classification of T1w images. These existing tools, however, do not allow for identification for common preprocessing errors such as poor tissue-class segmentation, and poor registrations to templates and across modalities. Often, these errors are detected *via* detailed manual QC but the visual inspection of hundreds to thousands of subject's processed multi-modal data derivatives is unfeasible and a streamlined QC workflow at the scale of such large datasets is needed.

The UK Biobank offers an alternative multi-modal MRI (anatomical, fMRI, dMRI, susceptibility-weighted MRI) processing pipeline that mostly relies on tools from the FMRIB Software Library (FSL; Jenkinson et al., 2012) and maintains images in volumetric space. The pipeline is fully automated, built to process the very large and longitudinal UK Biobank sample of aging individuals. It generates a number of image-based metrics of raw and processed intermediates, mostly from their structural preprocessing sub-pipeline. Referred to as "Imaging-Derived Phenotypes," these metrics were used for automated QC of the large UK Biobank aging sample. Here, we describe an extension of the UK Biobank pipeline that addresses the expanding scope of TheVirtualBrain project. The extension includes the generation of matched structural and functional connectivity data based on a user-defined brain parcellation, expanded capability for additional MRI modalities and manufacturers, additional preprocessing considerations for aging data (e.g., age-specific templates), an expanded number of image-based metrics for fMRI and dMRI, and the addition of new metrics for structural and functional connectivity. We have also developed an extensive new HTML-based QC report for quick assessment of raw, intermediate and processed outputs, and containerized the pipeline to maximize portability and ease of installation. The pipeline supports data from aging and neurodegenerative populations, and has been tested on a number of different datasets including multi-modal MRI data from the Cambridge Centre for Ageing and Neuroscience study (Cam-CAN; Taylor et al., 2017) as well as the ADNI3 study (Weiner et al., 2016). Finally, in keeping with TheVirtualBrain's commitment to the FAIR guiding principles (Wilkinson et al., 2016) and open science practices, our pipeline is open source and compliant with the Brain Imaging Data Structure (BIDS) standard (Gorgolewski et al., 2016). Below, we describe the software and methodological modifications and additions we made to the original UK Biobank pipeline, highlight the new QC pipeline and HTML report, show

some usage examples, and discuss future work and integrations with TheVirtualBrain.

METHOD

We refer to our pipeline as TheVirtualBrain-UK Biobank (or TVB-UKBB) pipeline. It is built from a fork of the UK Biobank pipeline,[1] which has been previously described (Alfaro-Almagro et al., 2018). The UK Biobank pipeline processes a variety of MRI modalities but, for the purposes of creating TVB inputs, we focused on modifying and extending the existing structural (T1w, T2 FLAIR), functional (resting-state, task), and diffusion-weighted MRI sub-pipelines. The processing of other MRI modalities (e.g., susceptibility-weighted imaging) in the TVB-UKBB pipeline remain unaltered and untested.

Figure 1 shows the general workflow of the whole pipeline, its sub-pipelines, and their outputs. The pipeline accepts MRI data in both raw DICOM and reconstructed NIfTI formats, and data may be organized into any directory structure, including BIDS. The major output of the structural MRI pipeline is the user-defined parcellation registered to the subject's T1w image. The registered parcellation is used by both the functional and diffusion MRI sub-pipelines to define ROIs for computing average regional timeseries and connectivity measures for TVB inputs. Following the completion of the functional and diffusion MRI sub-pipelines, an "IDP" pipeline computes image-based metrics for all modalities. Finally, our newly developed QC pipeline generates a comprehensive HTML-based report for manual quality assurance procedures.

Structural Sub-Pipeline

Our pipeline largely retains the structural (T1w, T2 FLAIR) preprocessing steps from the UK Biobank pipeline (Alfaro-Almagro et al., 2018). These include brain extraction and non-linear registration to the MNI152 standard-space T1 template, defacing, bias correction, and tissue-class segmentation (**Figure 2**). Processing of T2* images (brain extraction, registration to MNI152 and T1w, bias correction) has been added. Other major modifications and additions to the structural sub-pipeline are outlined below.

Parcellation

To support connectome-based modeling in TVB, our additions to the structural sub-pipeline allow users to create connectomes from T1w, dMRI, and resting-state fMRI data by specifying a brain parcellation of their choice. Currently, our pipeline supports parcellations defined on the MNI152 1mm template. For ease, we include three different parcellations in our repository. Two are combinations of the Schaefer cortical (Schaefer et al., 2018) with either the Tian subcortical (Tian et al., 2020) or Harvard-Oxford subcortical (Frazier et al., 2005) parcellation and the third is the Regional Map parcellation (Bezgin et al., 2017). The Schaefer-Tian parcellation is offered at three different scales of granularity and, if the user wishes, other scales can be created from the parcellations shared on the respective GitHub repositories. A tab-separated look-up table for the parcellation that specifies image labels and label names is required. The parcellation is registered to the T1w image using the warps from the non-linear registration of the template to T1w.

Segmentation

In both healthy older adult and neurodegenerative samples, accurate tissue classification using T1w images is hindered by decreasing image contrast with age (Bansal et al., 2013). Additional difficulties in T1w tissue classification arise from the presence of white matter pathology, where white matter lesions become misclassified as gray matter (Levy-Cooperman et al., 2008). Since tissue classification is a vital part to defining accurate ROIs for both structural and functional connectivity, we have implemented a number of modifications to the segmentation procedure to improve ROI assignments. We derive an initial image segmentation following the UK Biobank's procedure using FSL's FAST toolbox. We then refine the gray matter subcortical segmentation by adding the outputs of FSL's FIRST toolbox (an object model-based segmentation and registration tool) to the gray matter mask.

To address inaccuracies in the gray matter mask due to the presence of WM pathology, we have implemented two alternative methods that may be used depending on available image modalities. The first method, if T2 FLAIR images are available, uses the outputs of the WM lesion classification (FSL's BIANCA) to exclude any misclassified voxels from the gray matter mask and add them back to the white matter mask. The second method is an option for when T2 FLAIR images are not available. In these cases, we use age-specific image classes (Fillmore et al., 2015) as tissue priors. T1w images from adults aged 40 or over are registered to the template for their age decile (e.g., 40–49 years, 50–59 years, etc.) while subjects aged under 40 are registered to the FSL-distributed tissue priors. These template space-registered T1w images are then segmented using the set of matching age-specific priors. Segmented images are registered back to T1w space. Age-specific templates are provided up to the 80s age decile. Subjects older than 89 years are registered to the 80–84 years template.

Defining Regions of Interest for fMRI and dMRI Sub-Pipelines

The user-provided parcellation is registered to the T1w image and the gray matter mask is labeled with ROI indices. The labeled gray matter volume serves as input to the functional MRI sub-pipeline. The white and gray matter segmentations are both used to create the gray matter–white matter interface for dMRI tractography. This interface consists of voxels of white matter that are adjacent to gray matter and, when labeled, will serve as the seed and target masks for tractography in the diffusion MRI sub-pipeline.

Functional Magnetic Resonance Imaging Sub-Pipeline

The fMRI sub-pipeline processes both resting-state- and task-fMRI data (**Figure 3**). The processing of both data types by the UK Biobank pipeline relies on FSL's FEAT toolbox.

[1] https://git.fmrib.ox.ac.uk/falmagro/UK_biobank_pipeline_v_1

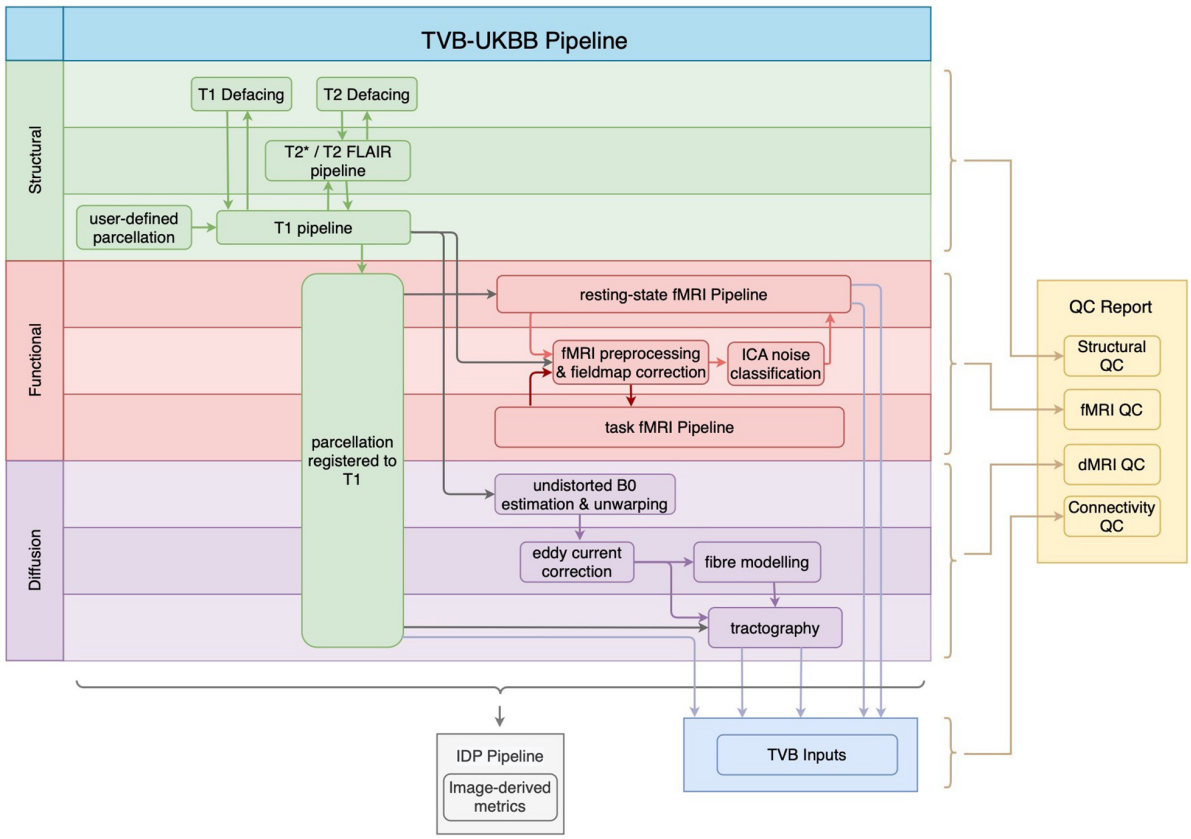


FIGURE 1 | General workflow of the TVB-UKBB pipeline. The main imaging sub-pipelines of interest for the current paper are shown (structural in green, functional in red, and diffusion in purple). A TVB-compatible .zip file (TVB Inputs) is created from the relevant outputs of the imaging sub-pipelines. The "IDP Pipeline" collects image-based metrics from raw, intermediate and processed outputs across imaging sub-pipelines and makes them available for analysis. The final step of the pipeline is the generation of the QC report.

As best practices for preprocessing of fMRI data are both dataset-dependent and constantly evolving (Uddin, 2017), the pipeline allows users flexibility on selecting the right preprocessing methods for their needs. Users may specify their preferences, which can include brain extraction, motion correction *via* realignment of fMRI images (MCFLIRT), slice timing correction, spatial smoothing, intensity normalization, and temporal filtering. Registration to the T1w image and MNI152 template is performed. For resting-state fMRI data, automated classification and removal of noise artifacts is performed using FMRIB's ICA-based Xnoiseifier (FIX) (Griffanti et al., 2014).

We have modified the UK Biobank pipeline to now accept an arbitrary number of fMRI sessions. Other major additions and modifications are described below.

Field Map Correction

The UK Biobank pipeline performs geometric distortion correction for the unwarping of EPI (e.g., fMRI and dMRI) images. This correction requires a reverse phase-encoded B0 dMRI image for estimating the field map, which is not always available. To support more "traditional" field map acquisitions for EPI distortion correction, such as those in the Cam-CAN dataset, we have implemented the option for dual echo-time gradient distortion correction using FSL's FUGUE toolbox.

Resting-State fMRI

We have updated the pipeline's FIX version from 1.063 to 1.06.15. Although FMRIB provides a default trained-weights file, and we provide trained-weights files for both the ADNI3 and Cam-CAN datasets, the classifier performs best when trained with the user's specific dataset. The most notable addition to resting-state fMRI processing is the replacement of group-ICA-based detection of resting-state networks with the parcellation of the resting-state fMRI data to accommodate connectome-based modeling. Following denoising, the parcellation output from the structural sub-pipeline (**Figure 2**) is registered to a reference resting-state fMRI volume and the average BOLD response across voxels is computed for all ROIs (i.e., ROI time series). The Pearson correlation coefficient between all ROI time series is also computed to obtain a measure of functional connectivity.

Task-Based fMRI

In our implementation of the fMRI sub-pipeline, task-based fMRI data are minimally preprocessed but not further analyzed. Users may choose to re-implement a GLM-based analysis using

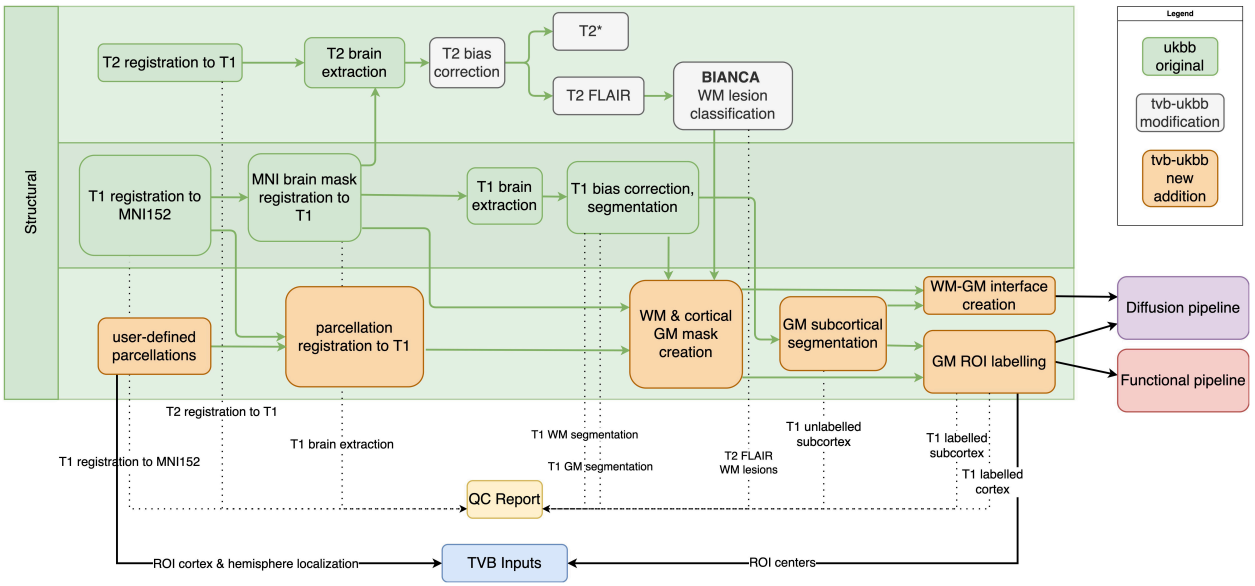


FIGURE 2 | Structural sub-pipeline workflow. Original components of the UK Biobank pipeline with few or no modifications are in green; pipeline components with major changes or additions are indicated in white; and new components are indicated in orange. Dotted lines indicate components that are included in the QC report. Black lines indicate components that are used downstream by other sub-pipelines or included in "TVB Inputs." GM, gray matter; WM, white matter.

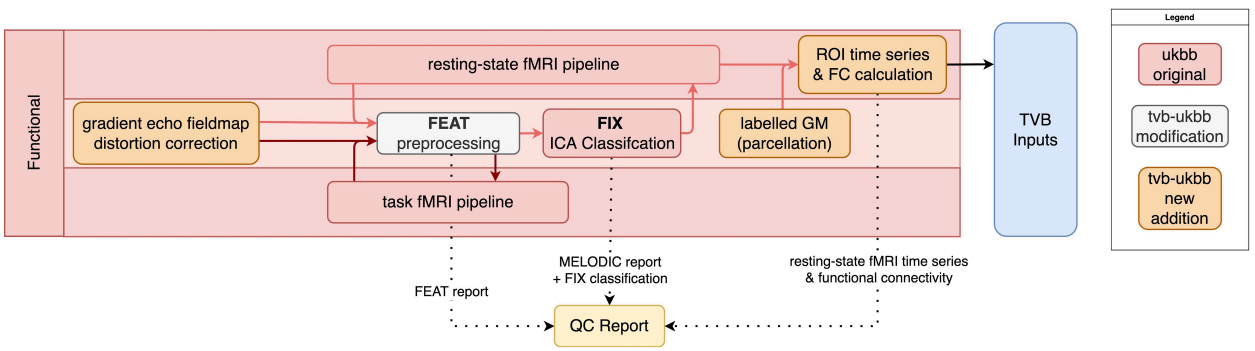


FIGURE 3 | fMRI sub-pipeline workflow. Original components of the UK Biobank pipeline with few or no modifications shown in red; pipeline components with major changes or additions shown in white; and new components shown in orange. Dotted lines indicate components that are included in the QC report. Black lines indicate components that are included in the TVB Inputs.

FEAT or, alternatively, they may take the preprocessed task-fMRI data and apply other analytic methods (e.g., Partial Least Squares; McIntosh and Lobaugh, 2004).

Diffusion Sub-Pipeline

Processing steps for diffusion imaging data that we have retained from the UK Biobank pipeline include correction of eddy currents and head motion (EDDY), diffusion tensor image fitting (DTIFIT) for tract-based analysis (TBSS), and multi-fiber orientation modeling (BEDPOSTX) (**Figure 4**). New features and additions to the diffusion sub-pipeline are described below.

Distortion Correction With Synthesized B0

Our first addition to the diffusion sub-pipeline was the integration of B0 field estimation for unwarping data that lack reverse phase-encoded images using the Synb0-DisCo tool (Schilling et al., 2019). This tool uses a deep learning approach to create a synthetic undistorted B0 image from a T1w image. The synthetic undistorted B0 is used as input to FSL's TOPUP toolbox for dMRI distortion correction. In our pipeline, users have the option to implement this tool to improve registrations between the T1w and dMRI images.

Tractography

The other major addition to the dMRI sub-pipeline was the replacement of the UK Biobank tractography approach with one that takes as input the user-defined parcellation for connectome construction. In our approach, the gray matter–white matter labeled interface is registered to the distortion-corrected B0 image. This interface is used to define seed and target ROI masks.

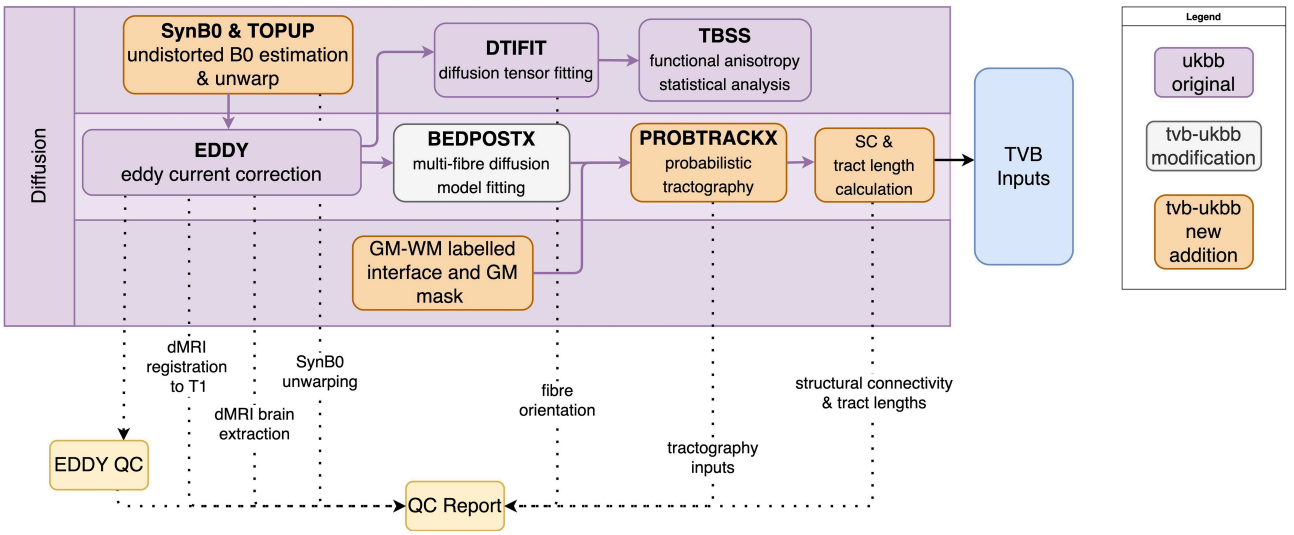


FIGURE 4 | Diffusion sub-pipeline workflow. Original components of the UK Biobank pipeline with few or no modifications shown in purple; pipeline components with major changes or additions shown in white; and new components shown in orange. Dotted lines indicate components that are included in the QC report. Black lines indicate components that are included in the TVB Inputs.

The gray matter mask is also registered to the B0 image and used as an exclusion mask. Probabilistic tractography is performed using FSL's PROBTRACKX toolbox to generate a matrix of the streamlines between all ROIs. The structural connectivity "weights" matrix is then computed by taking the streamlines matrix and dividing it by the total number of streamlines that were successfully sent from the seed ROIs. This weights matrix therefore encapsulates the probability of connection between all ROIs. "Distance" matrices (i.e., estimated tract lengths) are also obtained. Since directionality of fiber tracts cannot be inferred from dMRI tractography, both the weights and distance matrices are symmetrized. No other post-processing of structural connectivity, including thresholding, is performed.

Compatibility With TheVirtualBrain

Our pipeline generates inputs for connectome-based modeling, with file formats that are directly compatible with TheVirtualBrain (TVB[2]) (**Supplementary Figure 1**). These include the structural connectivity weights and tract lengths matrices, as well as the ROI time series and functional connectivity matrix from resting-state fMRI scans. ROI location information such as hemisphere or subcortical localization and centroid coordinates are also included. Toward the end of the pipeline, these TVB-input files are given the appropriate file names, placed in the correct folder structure, and compressed into a zip file that can be accepted by TVB without further processing. This zip file can be found in the top-level directory for each processed subject. The TVB website[3] has a variety of resources, including sample code, videos, and documentation, available for use with connectivity data such as those generated by the TVB-UKBB pipeline.

Imaging-Derived Phenotypes

The original UK Biobank pipeline generates various image-based metrics, or imaging-derived phenotypes (IDPs), for evaluating the characteristics of input images, pipeline processing outputs, and derivative files. These IDPs were intended to be a quantitative measure of the quality of processed subjects but mostly describe structural sub-pipeline processing and outputs. To better capture modalities of interest for connectome-based modeling, we have developed an additional 75 unique IDPs that describe fMRI and dMRI processing as well as connectivity outputs (**Supplementary Table 1**). Notable examples include IDPs for assessing the alignment of various modalities to T1 space, the temporal signal-to-noise ratio (tSNR) in resting-state fMRI, and summary statistics for functional and structural connectivity. In conjunction with the original IDPs, these new metrics were developed for the purpose of flagging subjects whose outputs' quality is poor, either due to acquisition errors, subject anomalies, or pipeline errors and insufficiencies.

We performed a manual QC of 140 (70 female, 70 male) Cam-CAN subjects using our QC reports (described below) to enable assessment of the utility of our newly developed IDPs for quantifying processing errors. The subjects were pseudorandomly selected, balanced for sex and 20 were chosen from each age decile to cover the entire age range of the dataset. Two experienced subject raters (DS, AK) scored the processing intermediates and outputs. These graders gave each subject a score along a 5-point scale for each modality (ranging from excellent [1] to poor [5]) and also gave each subject a pass/fail classification based on the integrity of

[2]http://thevirtualbrain.org/

[3]https://thevirtualbrain.org/tvb/zwei/brainsimulator-help

the TVB inputs as a whole. A fuller description of the QC procedure and example QC report usage for the Cam-CAN data is presented in the Section "Results." We used a multivariate statistical approach, partial least squares analysis (Krishnan et al., 2011), to identify a set of latent variables that represent the maximal covariance between the QC ratings and the image-based metrics outputted from the pipeline. First, the covariance between the two sets of variables was computed. Singular value decomposition on this cross-block covariance was then performed to produce latent variables, each containing three elements: (1) a set of weighted "saliences" that describe a pattern of IDPs; (2) a design contrast of QC ratings that express their relation to the saliences, and (3) a scalar singular value that expresses the strength of the covariance. The mutually orthogonal latent variables are extracted in order of magnitude, with the first latent variable explaining the most covariance between the IDPs and QC ratings, the second LV the second most, and so on. We report the relative percentage of total cross-block covariance explained by each latent variable, where the sum of this percentage across all latent variables is 100. The statistical significance of each latent variable was assessed with permutation testing: 1,000 permutations shuffled subjects' QC ratings without replacement while maintaining their IDP assignments. This resulted in 1,000 new covariance matrices which were each subjected to singular value decomposition to produce a null distribution of singular values. The reliability with which each IDP expressed the differences across QC ratings was determined with bootstrapping: 500 bootstrap samples were created by resampling subjects with replacement within each rating class. This resulted in 500 new covariance matrices which were, again, subjected to singular value decomposition. The 500 saliences from the bootstrapped dataset were used to build a sampling distribution of the saliences from the original dataset. The bootstrap ratio for a given IDP was calculated by taking the ratio of the salience to its boostrap-estimated standard error. With the assumption that the bootstrap distribution is normal, the bootstrap ratio is akin to a Z-score and corresponding saliences were considered to be reliable if the absolute value of their bootstrap ratio was ≥ 2.

Quality Control Report

Typical manual QC requires users to manually search for NIfTI files, load them into visualizer GUIs like FSLeyes, and adjust various parameters for each overlay. To streamline these procedures, our pipeline generates a Quality Control (QC) Report for each subject. The QC sub-pipeline runs at the end of the TVB-UKBB pipeline and leverages derivative data to generate brain image overlays, data visualization plots, and summary tables. These assets are wrapped in an offline HTML page that can be compressed into a portable, small, and standalone archive using a script included in the pipeline. This standalone report may be viewed on any browser and requires no access to the original subject's files.

Our QC Report allows users to view and interact with 17 preset key QC overlays immediately upon opening the HTML report. Our QC Report offers the ability to zoom, pan, switch between planes of view, inspect different analyses, and toggle visibility of layers in brain overlay images. These controls are also assigned to various hotkeys, allowing for browsing without a mouse and further expediting the QC process for more experienced users. Additionally, each brain overlay shows an array of 18 slices for each orientation, saving time typically spent seeking slices in visualization software. Especially when considering that multiple different overlays need to be generated for QC and certain overlays may need to be revisited more than once, our HTML Report can economize users' time and effort in the QC process.

The QC Report features a page for each sub-pipeline and multiple analyses on each page, corresponding to various key steps of the sub pipeline. For instance, brain image overlays, generated using FSL's FSLeyes and SLICER, are intended to offer users qualitative assessment of brain extraction, segmentation, registration, and labeling for multiple modalities (**Figure 5**). Data visualization plots are also included to simplify the verification of TVB-inputs. IDP tables offer a simple interface for accessing metrics and assessing the quality of a subject's processing. Within these tables, rows of IDPs are color-coded green or red (pass or fail) depending on their values relative to user-defined thresholds. A more detailed summary and explanation of QC analyses included in the report can be found in the **Supplementary Tables 2–4**. At the bottom of several QC Report pages, there are multiple file path links to the depicted overlay image as well as its source NIfTI image files. If more detailed investigation into a processed subject is required, then users have the option to load these files and perform QC with a NIfTI visualizer.

As part of the QC sub-pipeline development, we included FSL's EDDY QC toolbox for generating automated reports of within-(EDDY QUAD) and across-(EDDY SQUAD) subject QC assessments. Reports automatically generated by these tools, along with others from FEAT and MELODIC can be found in our QC Report. Notably, our QC Report reconstructs the existing MELODIC ICA report and combines it with classified ICA outputs from FIX into a single MELODIC page. This page groups signal and noise labeled components for quick assessment of FIX performance and allows immediate access to every component's analyses through a set of dropdown menus and optional hotkeys.

The QC Report is portable, at ∼180 MB for a compressed QC Report compared to ∼2 GB to ∼5 GB for a compressed full subject for the datasets we have tested. This enables faster and lower-overhead report sharing and collaboration without needing to share potentially sensitive raw or intermediate data. Furthermore, viewing the report requires no installations and it can be run on any operating system and modern browser. The lightweight and portable nature of our report is especially impactful for users who work on headless servers and may need to download files for visualization.

The Brain Imaging Data Structure

During processing, we retain and mimic the directory structure and file organization of the UK Biobank pipeline. We extend the UK Biobank's BIDS conversion script, which organizes pipeline output files in a manner outlined in a filename conversion dictionary. Our extension updates the conversion dictionary with

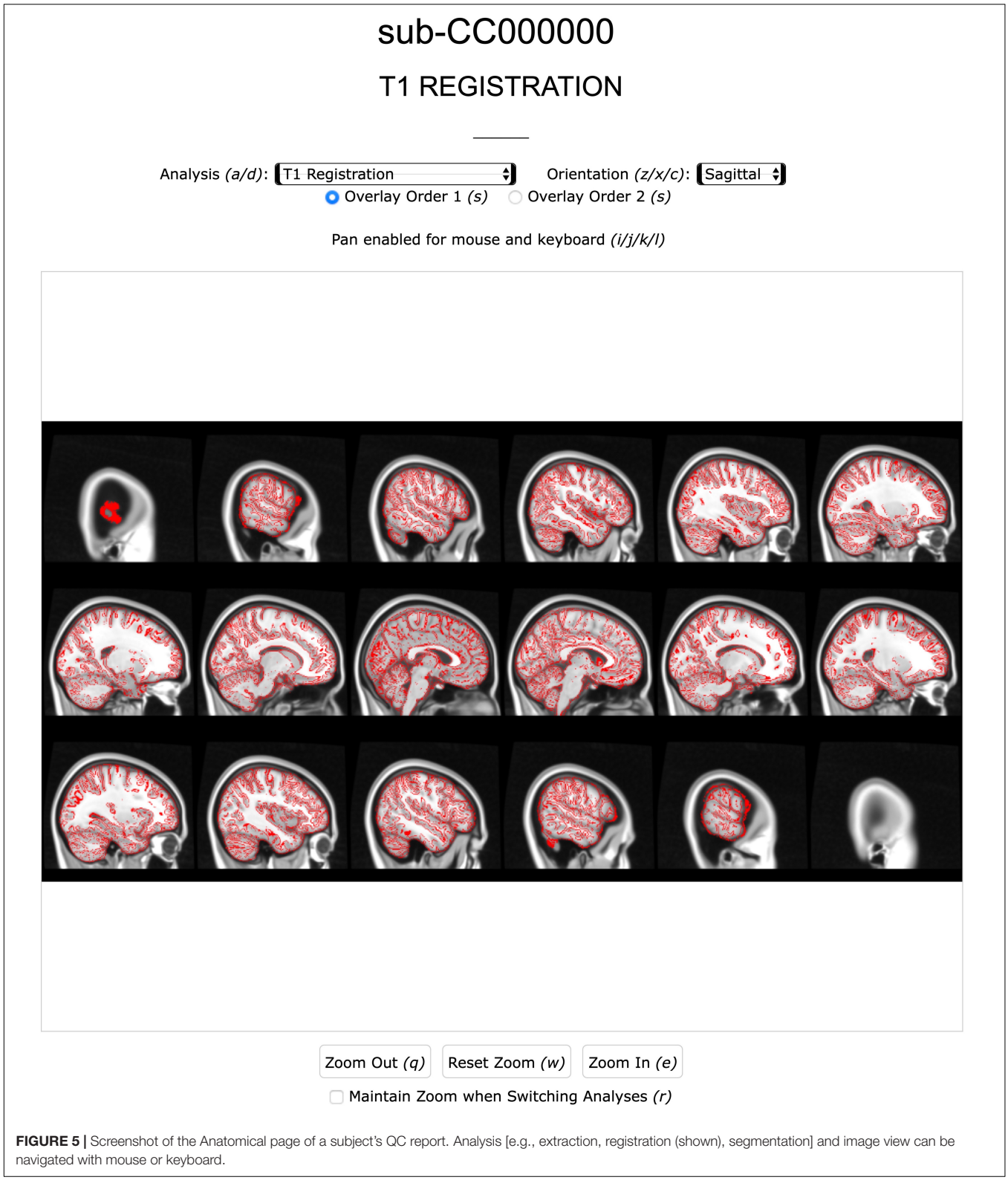


FIGURE 5 | Screenshot of the Anatomical page of a subject's QC report. Analysis [e.g., extraction, registration (shown), segmentation] and image view can be navigated with mouse or keyboard.

BIDS-compliant filenames for new TVB-UKBB intermediate and output files. This ensures interoperability of our pipeline's outputs, such that the derivative and raw data files for each subject are named, documented, and organized in a directory structure in accordance with BIDS v1.6.0. Additionally, we have introduced a reversal feature to the BIDS conversion script,

allowing BIDS-converted pipeline outputs to be reverted to the original TVB-UKBB file organization to facilitate reprocessing and reproducibility.

Developed Software

The pipeline has been constructed principally with Linux compatibility in mind. The software utilizes a Python backbone which brings together various BASH, MATLAB, and R scripts to process data moving through the pipeline. This software environment is encapsulated largely in a conda environment which can be used standalone or inside a supplied Singularity container (Kurtzer et al., 2017). The installation is straightforward and self-contained, with minimal dependencies on external applications after configuration. The Singularity container enables users to stage and run the pipeline in myriad high-performance computing environments and to leverage the batching capabilities of schedulers like SLURM and SGE.

GitHub Repository and Documentation

The source code for our pipeline is hosted on GitHub.[4] Several versions of the pipeline exist, each catering to different dataset needs and specifications. These versions are stored as separate branches on the repository. For example, branch Cam-CAN is available for pipeline users who want to process Cam-CAN subjects or datasets similar in specification to the Cam-CAN dataset using the Singularity container. ADNI3 is similar and is also the basis for the main branch as it is likely compatible with the widest range of datasets that the pipeline would be used with.

Extensive documentation on the TVB-UKBB pipeline is available on the Wiki page of our GitHub repository. This wiki includes information on the methodological components of the pipeline as well as installation, troubleshooting, QC interpretation, usage examples, etc.

A sample subject from the The Amsterdam Open MRI Collection (Snoek et al., 2021), containing inputs and processed outputs, is included in the repository so users may test and validate their own installations.

Installation and Singularity Container

Due to the high degree of complexity involved in the UK Biobank pipeline installation process, significant efforts were made to streamline installation and configuration. Singularity is a core component of these streamlining efforts due to its use in high performance computing environments as well as its ability to encapsulate complex and difficult-to-configure software stacks. Users may wish to install our pipeline with or without the Singularity container. All dependencies are included in the Singularity container, with the exception of FreeSurfer, AFNI, and ANTS. FSL and CUDA 9.1 were installed and configured in the container because GPU-enabled versions of BEDPOSTX, EDDY, and PROBTRACKX all require CUDA 9.1. MATLAB compatibility is packaged into the container using the MATLAB Compiled Runtime to eliminate the need for a MATLAB license.

[4]https://github.com/McIntosh-Lab/tvb-ukbb

Technical Features

The pipeline features CPU-only and CUDA-enabled versions. The CUDA-enabled version allows the FSL toolkit to take advantage of NVIDIA GPUs to drastically reduce runtimes of the BEDPOSTX, EDDY, and PROBTRACKX programs and cut the overall pipeline runtime significantly. If NVIDIA GPUs are not available, users can specify the CPU-only version which will run these FSL toolkits serially. To shorten the runtime and memory requirements of probabilistic tractography on CPU, we also include a parallelized implementation of PROBTRACKX.

Due to the variety of programming languages and heavy use of BASH, efforts were made to simplify configuration of pipeline parameters for end-users. The result is a single configuration file where the vast majority of environment variables for pipeline configuration and customization are specified. Parameters like the location of a FreeSurfer installation, specification of parcellation, etc. are set in this configuration file and is sourced prior to running the pipeline.

RESULTS

Usage

The pipeline currently supports several different datasets, including data from Cam-CAN and ADNI3, and can be customized with minimal effort to support novel datasets. Here we demonstrate usage of the TVB-UKBB pipeline using an example subject from the Cam-CAN dataset (Taylor et al., 2017), which includes T1w, T2*, resting-state and task-fMRI, field maps, and dMRI from ∼650 adults aged 18–99. In these examples, we used a Schaefer-Tian parcellation consisting of 400 cortical and 20 subcortical regions.

As we have not removed any features from the UK Biobank implementation, UK Biobank subjects should still work when processed with the TVB-UKBB pipeline. However, we were not able to validate this as we did not have access to the UK Biobank dataset at the time of this writing.

The key TVB inputs generated by the pipeline can be visualized and analyzed with ease. **Figure 6** shows the pipeline outputs of interest for connectome-based modeling for an example subject. These include the structural connectivity weights and tract lengths matrices, and the resting-state BOLD-fMRI responses and functional connectivity matrix.

Quality Control Procedures and Quality Control Report Usage

The QC reports allow users to quickly inspect pipeline intermediates and outputs. A detailed manual QC of a single subject without the QC report previously took our experienced raters (DS, KS) up to 30 min to complete, but a subject assessed with the QC report now takes an average of ∼5 min. Here we briefly outline our QC procedures for aging (Cam-CAN) and neurodegenerative (ADNI3) imaging data and provide some examples of common preprocessing errors detected using the QC reports. We describe the QC procedures in the order that the pipeline processes the data, but in practice we start QC

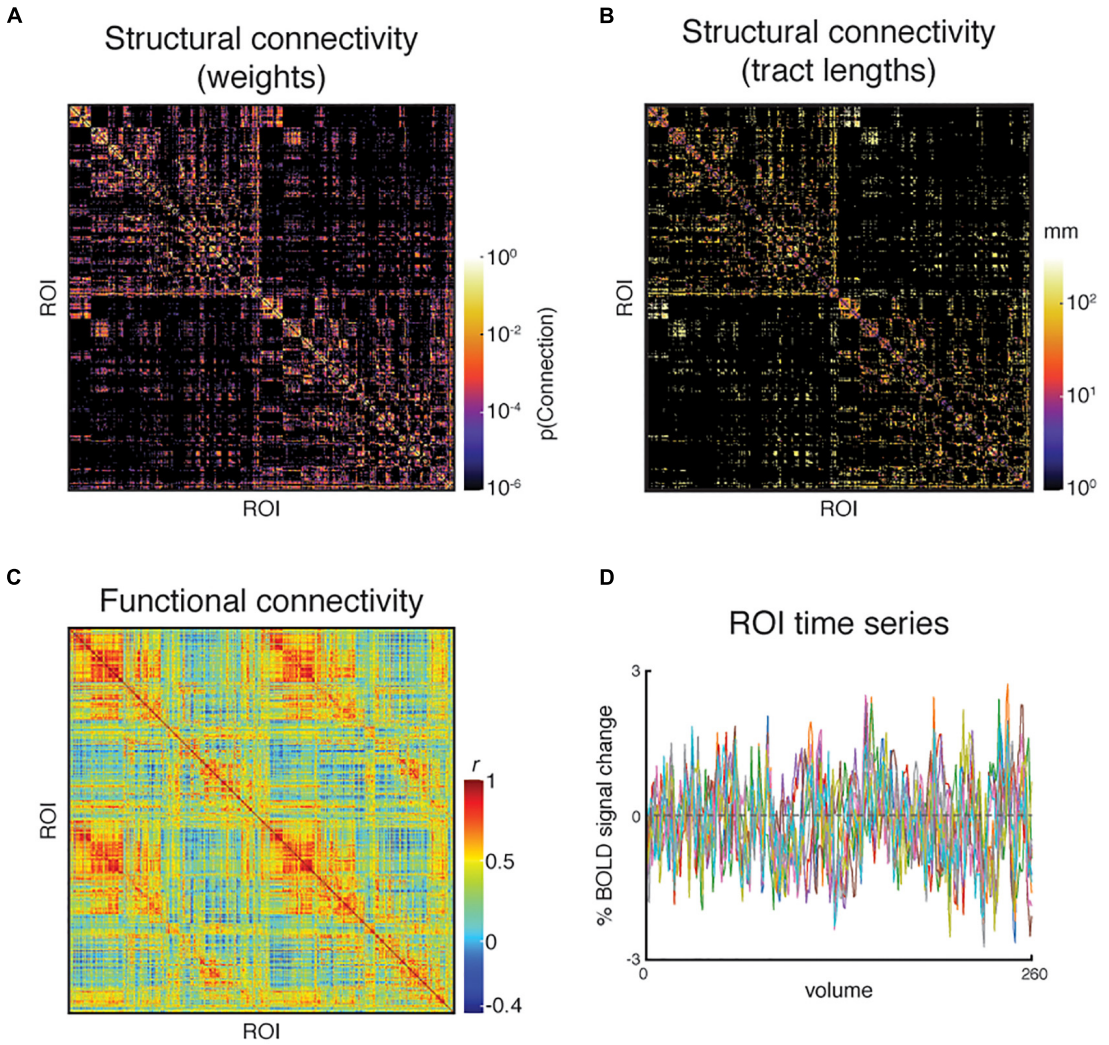


FIGURE 6 | An example subject's set of pipeline outputs for connectome-based modeling. These include **(A)** a weights matrix and **(B)** a tract lengths matrix from dMRI processing that capture the subject's structural connectivity; **(C)** a functional connectivity matrix of Pearon correlation coefficients, and **(D)** the region of interest (ROI) time series from resting-state fMRI processing. The structural connectivity matrices are presented on a log scale to enhance readability. Ten ROIs were chosen randomly for presentation in panel **(D)**.

investigations with the final outputs of the pipeline (structural and functional connectivity and functional responses) and work upstream through the QC report to quickly pinpoint the source of errors in processed subjects.

Structural Sub-Pipeline Quality Control

Examination of the structural pipeline includes the raw T1w image and the outputs of T1w brain extraction, segmentation, and registration to the MNI template. The reconstructed T1w image is checked for the presence of major motion or other visible artifacts. The T1w brain mask is then inspected and inclusion of dura along the lateral boundaries is noted.

The labeled and unlabeled segmentation outputs are also examined, and the accuracy of tissue classification (especially the delineation of gray and white matter) is assessed. Misclassification of non-brain tissue (i.e., inclusion in gray and white matter segmentations) is also noted. For older adults in the Cam-CAN sample (≥50 years), we also checked if white matter lesions were misclassified as gray matter during segmentation. This was supported by also inspecting the T2* image in conjunction with the T1w. **Figure 7** shows an example of white matter lesions being classified as gray matter. In cases with high WML loads, this will be impossible to avoid, and QC involves deciding to what extent the misclassification impacts tractography, namely the placement of seed and target ROIs, which will be covered below.

Finally, the registrations of the structural images to the MNI template are also inspected. Poor brain extraction and/or significant brain atrophy can affect the quality of the registration. Since the parcellation is defined on the MNI template, poor

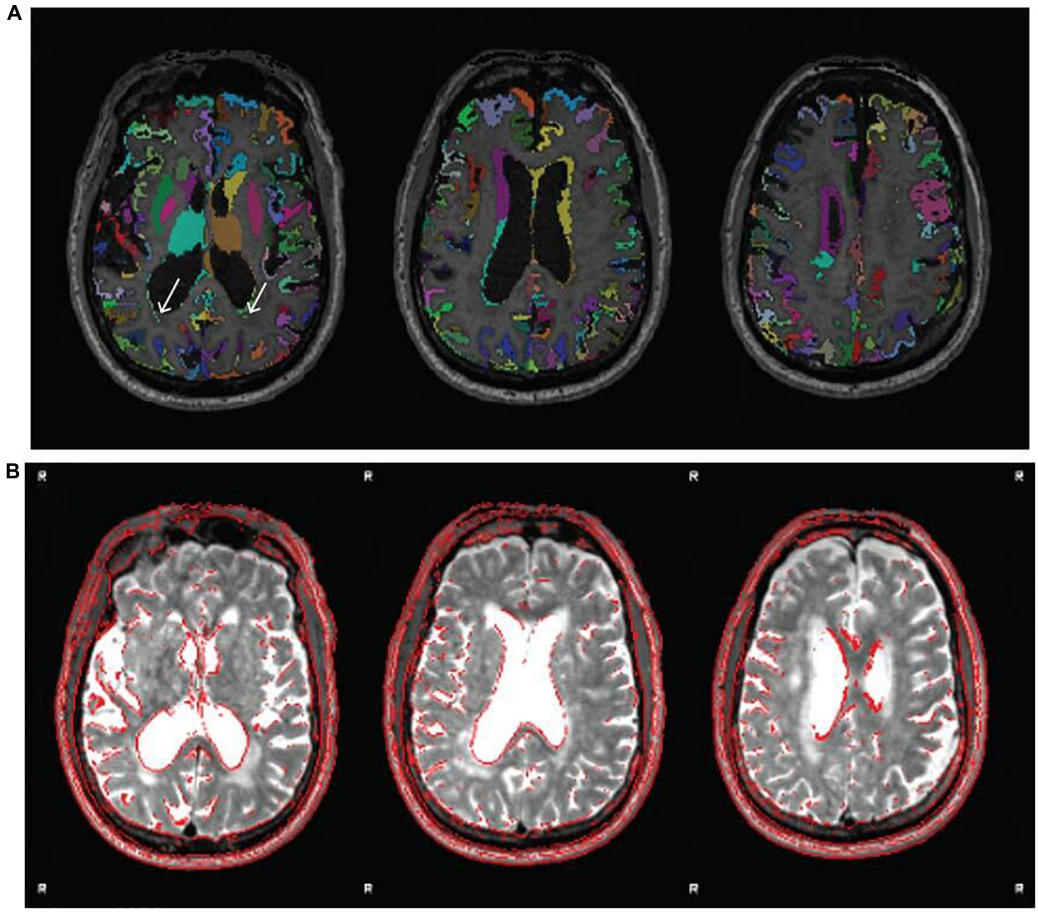


FIGURE 7 | Example of white matter lesion misclassification as gray matter. **(A)** The labeled gray matter image is shown on the T1w. **(B)** T2* image from the same older adult subject indicating a significant volume of white matter lesions that are also notable on the T1w. Although performing segmentation on the T1w image using age-specific tissue priors is largely successful despite the large white matter lesion volume, some misclassification remains [white arrows in panel **(A)**]. Images reproduced from the example subject's QC report.

registrations can substantially hinder the parcellated downstream outputs from both the functional and diffusion sub-pipelines.

Similar procedures are followed for examining T2* images. For T2 FLAIR images, like those in the ADNI3 dataset, lesion classification outputs from BIANCA are also examined.

Functional Sub-Pipeline Quality Control

For the purposes of creating modeling inputs for TVB, we focus here on QC of the processing of resting-state fMRI data. For these data, the hyperlinked FEAT report is used to check the field map registration and correction, the relative motion of the resting-state fMRI scans and their registrations to both the T1w and MNI152 template. Signal dropout in susceptible areas such as the temporal pole or orbitofrontal cortex, if substantial, is also noted. The MELODIC page of the QC report is used to examine the components classified as signal to determine whether substantial artifactual components were included post-processing.

The functional connectivity matrix is visually inspected in the QC report and is checked for the presence of strong homotopic connectivity, clear delineation of intra- and inter-hemispheric quadrants, a sensible range of correlation values and minimal "banding" which can reflect motion artifacts or misregistration of the parcellation. The QC report allows users to examine the matrix in conjunction with a carpet plot of the cleaned ROI time series and the MCFLIRT motion plots to determine whether residual motion artifacts impact the functional connectivity matrix. See **Figure 8** for an example of a bad resting-state fMRI processed outcome.

Diffusion Sub-Pipeline Quality Control

The QC procedure for the diffusion sub-pipeline starts with examining the undistorted B0 image to check the quality of distortion correction and the presence of major artifacts. The brain mask calculated from the distortion corrected B0 is also checked as it is used to exclude non-brain tissue from downstream diffusion processing. Brain masks that are too conservative are noted as they can impact registration and placement of ROIs for tractography. The principle orientations of the modeled fibers are also inspected to confirm that

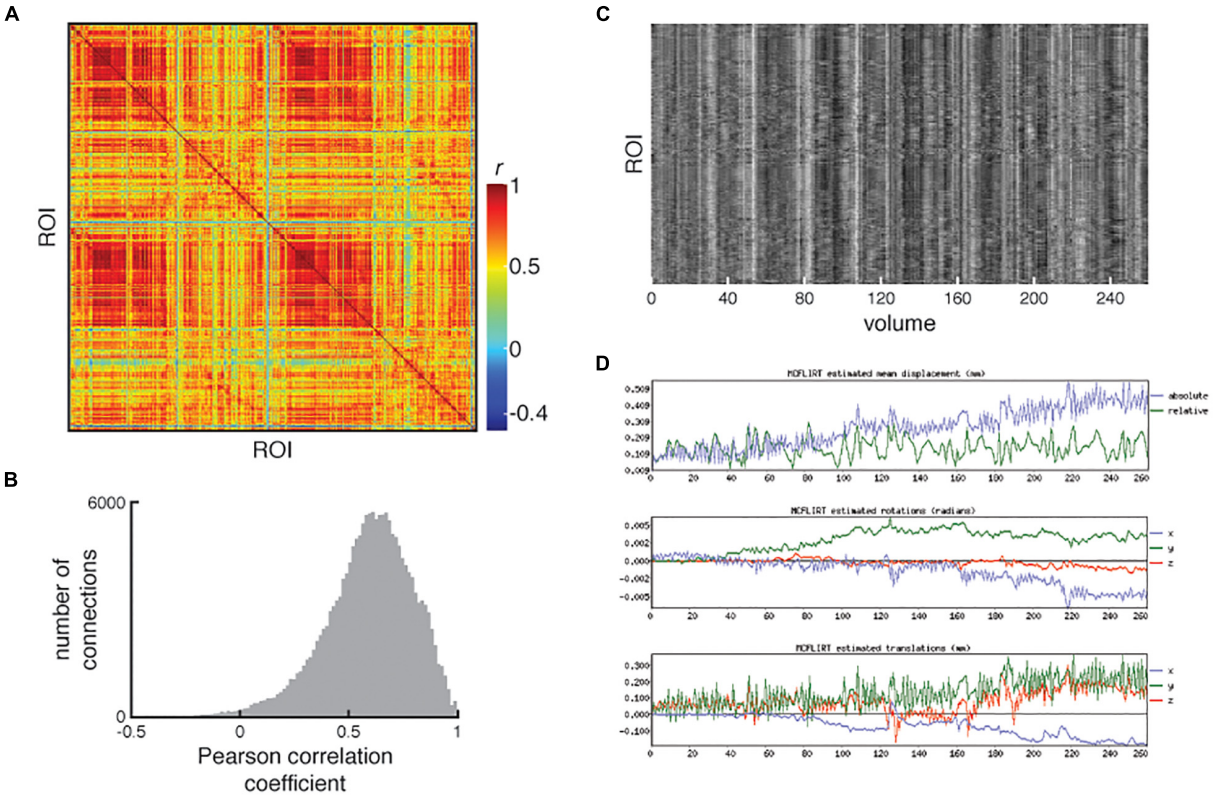


FIGURE 8 | An example of poorly processed resting-state fMRI. **(A)** Functional connectivity matrix and **(B)** distribution of functional connectivity show large number of strong positive correlations and a compressed range of correlations. **(C)** Examination of the carpet plot of region of interest (ROI) time series suggests artifacts remain in fMRI data after cleaning. **(D)** In the QC report, motion estimations from MCFLIRT are shown alongside the carpet plots for quick assessment. All images reproduced from the example subject's QC report.

the b-vectors have been specified appropriately. It is usually necessary to check the orientations for a single representative subject per study, but in the case of multi-site studies the user may wish to check representative subjects from each site. The registration between the reference B0 image and the T1w is also examined.

Next, the inputs for tractography are examined. These include the gray matter exclusion mask, and the seed and target ROIs that are overlaid on the FA image in the QC report. Each of these images are checked for accuracy of their placement. The border of the brain is also inspected and seeds that are mislocalized to dura or other non-brain tissue is noted (see **Figure 9** for example of poor quality tractography seed placement). With atrophic cases, poor T1-MNI template registration can impact the quality of the tractography within the brain and those with a large white matter lesion load will have lesions labeled as gray matter which can cause similar issues.

Finally, the structural connectivity matrices are examined. This includes the weights matrix, which is displayed with a logarithmic scale to improve visual assessment, and the tract lengths matrix. Visual inspection can be aided by the examination of the distributions of weights and tract lengths. Extreme sparsity of the connectome is easily detected and is often apparent in the interhemispheric quadrants of the matrices (**Figure 10**).

More examples of well-processed and poorly processed pipeline outputs can be found in **Supplementary Figures 2–9**.

Utility of New Imaging Derived Phenotypes and Other Summary Statistics

We performed manual QC of 140 Cam-CAN subjects to enable a preliminary assessment of the utility of existing and newly developed IDPs and summary statistics. This assessment was done using a partial least squares analysis of the IDPs with subjects grouped by the rater's scores. For the functional sub-pipeline, this analysis returned one significant latent variable (**Figure 11**) showing how IDPs related to head motion, temporal signal-to-noise ratio, the proportion of signal/noise components, and the distribution of functional connectivity values (e.g., center, range, shape) to be reliable indicators of resting-state fMRI processing quality (p = 0.001, 83.4% cross-block covariance).

A similar analysis of the diffusion sub-pipeline IDPs resulted in no significant latent variables. This was likely due to a lack of variability in the quality of the diffusion processing and structural

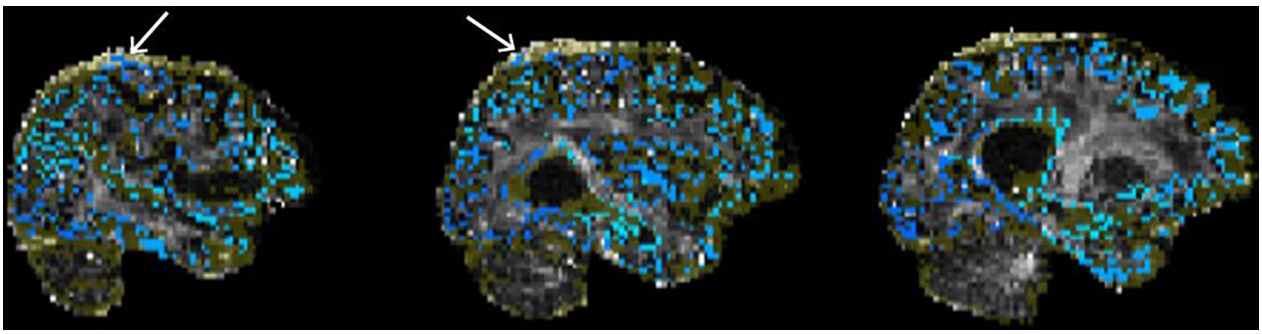


FIGURE 9 | Example of poor quality tractography seed/target placement. The seeds/targets image (blue) as well as the exclusion mask image (yellow) are overlaid on the FA image. White arrows indicate seeds/targets located in the dura.

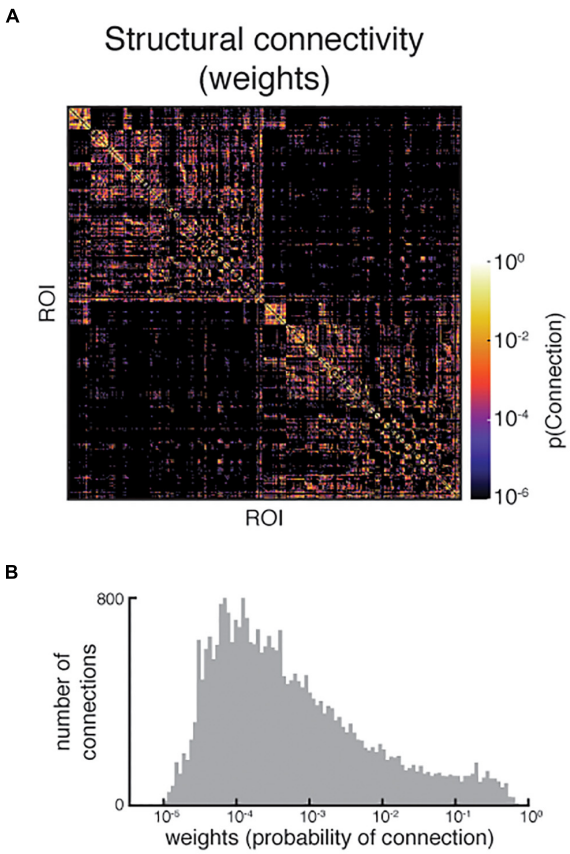


FIGURE 10 | (A) An example structural connectivity matrix of poorer quality. Note the sparsity, especially in the interhemispheric quadrants (top right and bottom left), which was confirmed by **(B)** the relatively small distribution of non-zero weights in the matrix. Upon further examination, the dMRI registration to T1w was poor, resulting in some tractography seeds and targets being placed in non-brain tissue. Both images shown are reproduced from the QC report.

connectivity, where nearly all subjects' (136/140) diffusion sub-pipeline outputs were judged by our raters to be either excellent (1) or very good (2).

DISCUSSION

We have described the development of the TVB-UKBB pipeline, an open-source, easy to install, automated multimodal MRI processing solution for generating inputs for connectome-based modeling that directly interface with TheVirtualBrain. We have expanded the original UK Biobank pipeline to accept additional MRI modalities and data from various manufacturers. Users may now provide their own parcellation of choice to generate complementary structural and functional connectivity outputs. We have also developed a QC report to support the assessment of pipeline outputs. The pipeline has been containerized and supports various job schedulers on high performance compute clusters. We have tested it on both healthy and clinical populations and added features to improve its robustness against the morphological changes observed in aging and dementia.

We developed the TVB-UKBB pipeline with the processing of aging and neurodegenerative data, such as those from ADNI (Mueller et al., 2005) and Cam-CAN (Taylor et al., 2017), in mind. These datasets present particular challenges such as significant changes in brain morphology with age and/or disease (i.e., brain atrophy) and decreased image contrast, which can greatly affect registrations to a standard template and the classification of tissue classes. We addressed inaccuracies in gray matter classification by either taking advantage of available T2 FLAIR images for classifying white matter lesions, or by using age-specific tissue priors when T2 FLAIR images are not available. Future developments will include a fuller implementation of age-specific or, more generally, study-specific templates to aid registrations.

Our pipeline offers an alternative for generating modeling inputs to pipelines that rely on working with cortical surfaces. This avoids the need to project lower resolution data to high resolution surfaces (Alfaro-Almagro et al., 2018), avoids manual interventions that might be needed for correcting tissue segmentations of aging and neurodegenerative data (McCarthy et al., 2015; Henschel et al., 2020; Srinivasan et al., 2020),

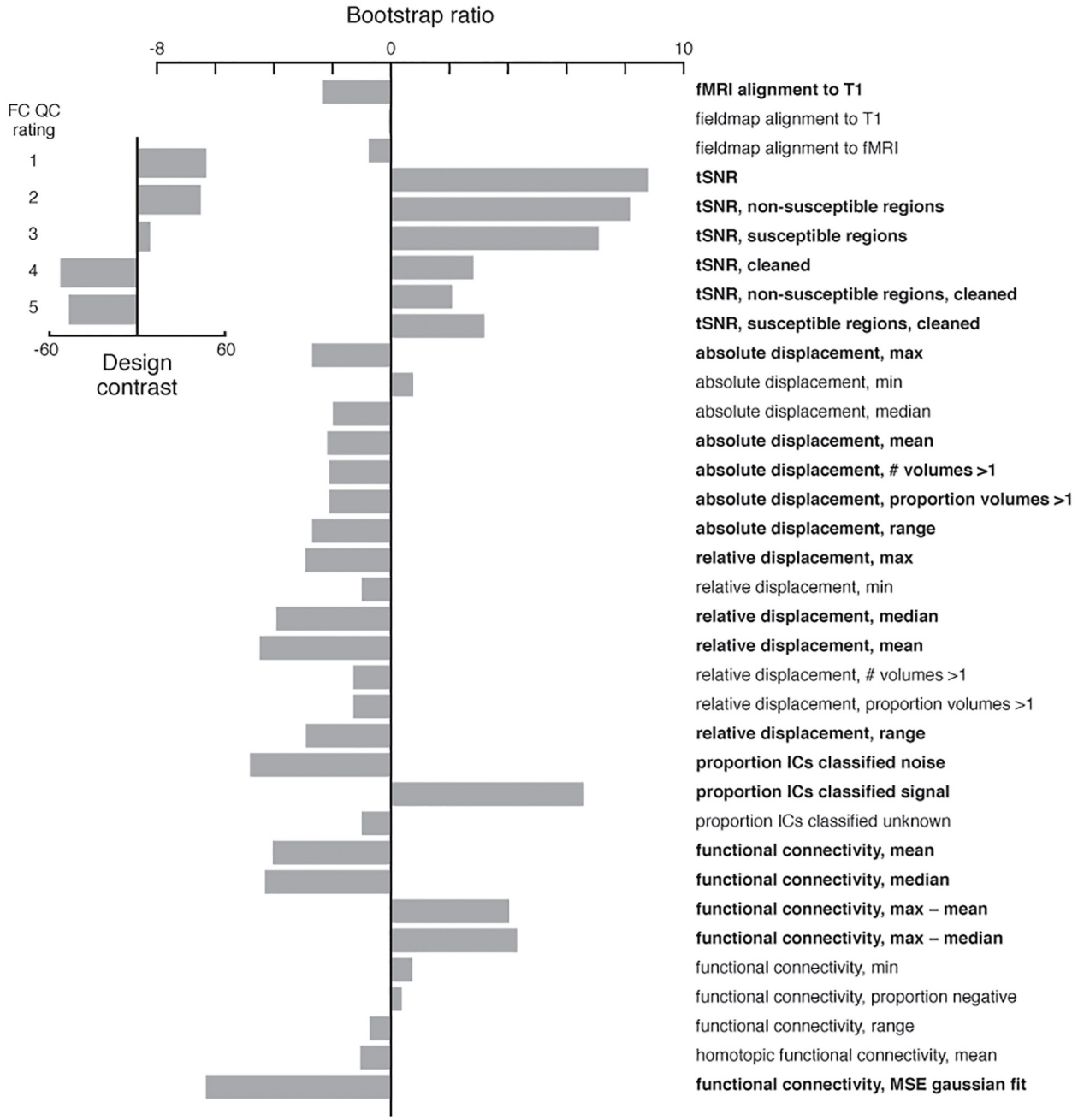


FIGURE 11 | Partial least squares analysis of functional sub-pipeline IDPs and summary statistics as a function of the functional connectivity QC rating. The analysis returned a contrast (inset) between good (1 and 2) and bad (4 and 5) scores. The most reliable indicators of a good QC rating included high temporal signal-to-noise ratios, low relative displacement, a higher proportion of ICA components classified as signal, and Gaussian-like functional connectivity distributions. IDPs and summary statistics with an absolute value bootstrap ratio > 2 were considered reliable (see Section "Method"), and are indicated in bold.

and avoids the long processing times needed for reconstructing the cortical surface. It also allows for easier integration of subcortical region parcels that, until very recently, were not available on the surface (see Lewis et al., 2022). We added the ability to perform distortion correction on dMRI data for datasets without reverse phase-encoded images by adopting a toolbox that generates a synthetic undistorted B0 image (Schilling et al., 2019). Tractography methodologies for our pipeline were chosen based on our previous validation work comparing probabilistic tractographic outputs to connectomes derived from anatomical tracer data in macaques (Shen et al., 2019b). We found this method to produce reasonable estimates of fiber tract capacities (or "weights") and fiber tract lengths. However, like many other reports of probabilistic

tractography (e.g., Thomas et al., 2014; Maier-Hein et al., 2017), we also found the method to be susceptible to false positives, generating connections where there ought not to be any. There are several thresholding methods to mitigate the effects of spurious connections (e.g., de Reus and van den Heuvel, 2013; Roberts et al., 2017; Shen et al., 2019b) and we leave it to users to decide the method that best suits their needs.

All of the above considerations were made so that a greater range of "legacy" datasets could be accommodated by our pipeline. Although these were all important, we recognize that cortical surface processing is considered state-of-the-art because it handles the problem of partial voluming effects and accommodates spatial smoothing to increase the signal-to-noise ratio (Brodoehl et al., 2020). Basic FreeSurfer support is already available as a part of the UK Biobank pipeline and future in-depth integrations with our pipeline are planned. GPU-enabled deep learning implementations, in particular, will be considered because they are attractive for creating more accurate cortical surface reconstructions quickly in aging and neurodegenerative data (Henschel et al., 2020). Given the increasing availability of GPU processing, this is in line with our efforts to develop a faster and more consistent pipeline. This type of cortical surface reconstruction will be especially important for our future development of M/EEG processing sub-pipelines where cortical surfaces are needed for computing the forward solution for source localization. Users may also wish to use other tractography approaches such as those that constrain tractography using anatomical priors (Smith et al., 2012). The modular implementation of our pipeline allows for these future adaptations to be implemented with relative ease.

A key component of our pipeline is the development of user-friendly HTML reports to facilitate QC assessment and faster subject scoring. With the introduction of hotkeys, fully navigable pre-generated image overlays, and re-compilation of FSL reports, our QC Reports make the novel and essential images generated by the QC sub-pipeline accessible. Existing reports are also consolidated with these images into a single, convenient point of access with an intuitive interface.

To further support QC efforts for large multimodal datasets, we developed a number of new image-based metrics and summary statistics for assessing resting-state fMRI and dMRI processing. The summary statistics, in particular, capture characteristics of processed data (i.e., connectivity matrices) that may still reflect residual artifacts that remain post-processing. For example, high motion indicated by simple motion related metrics may not warrant exclusion of a subject because some motion artifacts can be detected and removed. Post-processing summary metrics related to the FC can convey information about the successful or unsuccessful removal of motion artifacts which cannot be derived from simple motion-related metrics that are typically available in other QC reports. Image-based metrics from the UK Biobank's structural sub-pipeline has proved useful for training a classifier to detect poorly processed data (Alfaro-Almagro et al., 2018). Our preliminary assessment with a partial least squares analysis of our newly developed metrics suggest that extending the machine learning approach to include our new downstream metrics could be useful for automated QC.

We developed our pipeline with the FAIR principles for data (Wilkinson et al., 2016) and software (Lamprecht et al., 2019; Katz et al., 2021) management in mind. We adopt the BIDS neuroimaging standard (Gorgolewski et al., 2016) for raw data file naming, directory organization and metadata and extend the standard to the derived data. The source code is publicly available under the Apache 2.0 License, version controlled and supported by wiki-style documentation and a discussion board. Its containerization improves both accessibility and interoperability and its customization options allow for reuse across different datasets and research applications. Future iterations of the Singularity container will include FreeSurfer, AFNI, and ANTS once a solution to circumvent cloud storage quotas has been implemented.

Our pipeline generates multi-modal outputs for connectome-based modeling that are directly compatible with TheVirtualBrain software package. The high throughput nature of the pipeline, its robustness against the challenges imposed by MRI imaging of aging and clinical populations, and its extended QC capability contribute to the expanding scope of TheVirtualBrain project. In combination with the growing availability of datasets that span large age ranges and different neurological disorders, our pipeline supports TheVirtualBrain project's endeavors to understanding large-scale network dynamics at the level of the individual.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: http://adni.loni.usc.edu/data-samples/access-data/ and https://www.cam-can.org/index.php?content=dataset.

ETHICS STATEMENT

The studies involving human participants were reviewed and approved by Rotman Research Institute Research Ethics Board and the Cambridgeshire 2 Research Ethics Committee. The patients/participants provided their written informed consent to participate in this study.

AUTHOR CONTRIBUTIONS

KS and AM conceptualized the project, supervised the research activities, and acquired the financial support for the project. KS, DS, and JW developed the methodology. NF-L, JW, KS, and ZW contributed to the software development, implementation, and testing. AS, NF-L, JW, and KS performed the data curation. DS,

AK, AS, and KS validated the research outputs. KS, AK, and JW performed the statistical analysis. KS, NF-L, JW, and DS wrote the initial draft of this manuscript. All authors reviewed and edited this manuscript and approved the submitted version.

FUNDING

This project was supported by grants from the Canadian Institutes of Health Research and the BrightFocus Foundation to AM and KS, as well as by a grant from the Natural Sciences and Engineering Research Council of Canada to AM.

ACKNOWLEDGMENTS

This research was enabled in part by support provided by Compute Ontario (www.computeontario.ca/) and Compute Canada (www.computecanada.ca).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.883223/full#supplementary-material

REFERENCES

Alfaro-Almagro, F., Jenkinson, M., Bangerter, N. K., Andersson, J. L. R., Griffanti, L., Douaud, G., et al. (2018). Image processing and quality control for the first 10,000 brain imaging datasets from UK Biobank. *Neuroimage* 166, 400–424. doi: 10.1016/j.neuroimage.2017.10.034

Bansal, R., Hao, X., Liu, F., Xu, D., Liu, J., and Peterson, B. S. (2013). The effects of changing water content, relaxation times, and tissue contrast on tissue segmentation and measures of cortical anatomy in MR images. *Magn. Reson. Imaging* 31, 1709–1730. doi: 10.1016/J.MRI.2013.07.017

Bezgin, G., Solodkin, A., Bakker, R., Ritter, P., and McIntosh, A. R. (2017). Mapping complementary features of cross-species structural connectivity to construct realistic "Virtual Brains.". *Hum. Brain Mapp.* 38, 2080–2093. doi: 10.1002/hbm.23506

Brodoehl, S., Gaser, C., Dahnke, R., Witte, O. W., and Klingner, C. M. (2020). Surface-based analysis increases the specificity of cortical activation patterns and connectivity results. *Sci. Rep.* 101:5737. doi: 10.1038/s41598-020-62832-z

de Reus, M. A., and van den Heuvel, M. P. (2013). Estimating false positives and negatives in brain networks. *Neuroimage* 70, 402–409. doi: 10.1016/j.neuroimage.2012.12.066

Esteban, O., Birman, D., Schaer, M., Koyejo, O. O., Poldrack, R. A., and Gorgolewski, K. J. (2017). MRIQC: advancing the automatic prediction of image quality in MRI from unseen sites. *PLoS One* 12:e0184661. doi: 10.1371/journal.pone.0184661

Esteban, O., Markiewicz, C. J., Blair, R. W., Moodie, C. A., Isik, A. I., Erramuzpe, A., et al. (2019). FMRIPrep: a robust preprocessing pipeline for functional MRI. *Nat. Methods* 16, 111–116. doi: 10.1038/s41592-018-0235-4

Fillmore, P. T., Phillips-Meek, M. C., and Richards, J. E. (2015). Age-specific MRI brain and head templates for healthy adults from 20 through 89 years of age. *Front. Aging Neurosci.* 7:44. doi: 10.3389/fnagi.2015.00044

Fischl, B. (2012). FreeSurfer. *Neuroimage* 62:774. doi: 10.1016/J.NEUROIMAGE.2012.01.021

Frazier, J. A., Chiu, S., Breeze, J. L., Makris, N., Lange, N., Kennedy, D. N., et al. (2005). Structural brain magnetic resonance imaging of limbic and thalamic volumes in pediatric bipolar disorder. *Am. J. Psychiatry* 162, 1256–1265. doi: 10.1176/appi.ajp.162.7.1256

Glasser, M. F., Sotiropoulos, S. N., Wilson, J. A., Coalson, T. S., Fischl, B., Andersson, J. L., et al. (2013). The minimal preprocessing pipelines for the human connectome project. *Neuroimage* 80, 105–124. doi: 10.1016/j.neuroimage.2013.04.127

Gorgolewski, K. J., Auer, T., Calhoun, V. D., Craddock, R. C., Das, S., Duff, E. P., et al. (2016). The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Sci. Data* 3:160044. doi: 10.1038/sdata.2016.44

Griffanti, L., Salimi-Khorshidi, G., Beckmann, C. F., Auerbach, E. J., Douaud, G., Sexton, C. E., et al. (2014). ICA-based artefact removal and accelerated fMRI acquisition for improved resting state network imaging. *Neuroimage* 95, 232–247. doi: 10.1016/j.neuroimage.2014.03.034

Henschel, L., Conjeti, S., Estrada, S., Diers, K., Fischl, B., and Reuter, M. (2020). FastSurfer - A fast and accurate deep learning based neuroimaging pipeline. *Neuroimage* 219:117012. doi: 10.1016/j.neuroimage.2020.117012

Jenkinson, M., Beckmann, C., Behrens, T., Woolrich, M., and Smith, S. (2012). FSL. *Neuroimage* 62, 782–790. doi: 10.1016/J.NEUROIMAGE.2011.09.015

Katz, D. S., Gruenpeter, M., and Honeyman, T. (2021). Taking a fresh look at FAIR for research software. *Patterns* 2:100222. doi: 10.1016/j.patter.2021.100222

Krishnan, A., Williams, L. J., McIntosh, A. R., and Abdi, H. (2011). Partial Least Squares (PLS) methods for neuroimaging: a tutorial and review. *Neuroimage* 56, 455–475. doi: 10.1016/j.neuroimage.2010.07.034

Kurtzer, G. M., Sochat, V., and Bauer, M. W. (2017). Singularity: scientific containers for mobility of compute. *PLoS One* 12:e0177459. doi: 10.1371/JOURNAL.PONE.0177459

Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., et al. (2019). Towards FAIR principles for research software. *Data Sci.* 3, 37–59. doi: 10.3233/ds-190026

Levy-Cooperman, N., Ramirez, J., Lobaugh, N. J., and Black, S. E. (2008). Misclassified tissue volumes in Alzheimer disease patients with white matter hyperintensities: importance of lesion segmentation procedures for volumetric analysis. *Stroke* 39, 1134–1141. doi: 10.1161/STROKEAHA.107.498196

Lewis, J. D., Bezgin, G., Fonov, V. S., Collins, D. L., and Evans, A. C. (2022). A sub+cortical fMRI-based surface parcellation. *Hum. Brain Mapp.* 43, 616–632. doi: 10.1002/hbm.25675

Maier-Hein, K. H., Neher, P. F., Houde, J.-C., Côté, M.-A., Garyfallidis, E., Zhong, J., et al. (2017). The challenge of mapping the human connectome based on diffusion tractography. *Nat. Commun.* 8:1349. doi: 10.1038/s41467-017-01285-x

Marcus, D. S., Harms, M. P., Snyder, A. Z., Jenkinson, M., Wilson, J. A., Glasser, M. F., et al. (2013). Human connectome project informatics: quality control, database services, and data visualization. *Neuroimage* 80, 202–219. doi: 10.1016/j.neuroimage.2013.05.077

McCarthy, C. S., Ramprashad, A., Thompson, C., Botti, J. A., Coman, I. L., and Kates, W. R. (2015). A comparison of FreeSurfer-generated data with and without manual intervention. *Front. Neurosci.* 9:379. doi: 10.3389/fnins.2015.00379

McIntosh, A. R., and Lobaugh, N. J. (2004). Partial least squares analysis of neuroimaging data: applications and advances. *Neuroimage* 23(Suppl. 1), S250–S263. doi: 10.1016/j.neuroimage.2004.07.020

Mueller, S. G., Weiner, M. W., Thal, L. J., Petersen, R. C., Jack, C., Jagust, W., et al. (2005). Alzheimer's disease neuroimaging initiative. *Neuroimaging Clin. N. Am.* 15, 869–877. doi: 10.1016/j.nic.2005.09.008

Roberts, J. A., Perry, A., Roberts, G., Mitchell, P. B., and Breakspear, M. (2017). Consistency-based thresholding of the human connectome. *Neuroimage* 145, 118–129. doi: 10.1016/j.neuroimage.2016.09.053

Sanz Leon, P., Knock, S. A., Woodman, M. M., Domide, L., Mersmann, J., McIntosh, A. R., et al. (2013). The Virtual Brain: a simulator of primate brain network dynamics. *Front. Neuroinform.* 7:10. doi: 10.3389/fninf.2013.00010

Sanz-Leon, P., Knock, S. A., Spiegler, A., and Jirsa, V. K. (2015). Mathematical framework for large-scale brain network modeling in The Virtual Brain. *Neuroimage* 111, 385–430. doi: 10.1016/j.neuroimage.2015.01.002

Schaefer, A., Kong, R., Gordon, E. M., Laumann, T. O., Zuo, X.-N., Holmes, A. J., et al. (2018). Local-Global parcellation of the human cerebral cortex from intrinsic functional connectivity MRI. *Cereb. Cortex* 28:3095. doi: 10.1093/CERCOR/BHX179

Schilling, K. G., Blaber, J., Huo, Y., Newton, A., Hansen, C., Nath, V., et al. (2019). Synthesized b0 for diffusion distortion correction (Synb0-DisCo). *Magn. Reson. Imaging* 64, 62–70. doi: 10.1016/j.mri.2019.05.008

Schirner, M., McIntosh, A. R., Jirsa, V., Deco, G., and Ritter, P. (2018). Inferring multi-scale neural mechanisms with brain network modelling. *Elife* 7:e28927. doi: 10.7554/eLife.28927

Schirner, M., Rothmeier, S., Jirsa, V. K., McIntosh, A. R., and Ritter, P. (2015). An automated pipeline for constructing personalised virtual brains from multimodal neuroimaging data. *Neuroimage* 117, 343–357. doi: 10.1016/j.neuroimage.2015.03.055

Shen, K., Bezgin, G., Schirner, M., Ritter, P., Everling, S., and McIntosh, A. R. (2019a). A macaque connectome for large-scale network stimulations in TheVirtualBrain. *Sci. Data* 6:123. doi: 10.1038/s41597-019-0129-z

Shen, K., Goulas, A., Grayson, D. S., Eusebio, J., Gati, J. S., Menon, R. S., et al. (2019b). Exploring the limits of network topology estimation using diffusion-based tractography and tracer studies in the macaque cortex. *Neuroimage* 191, 81–92. doi: 10.1016/j.neuroimage.2019.02.018

Smith, R. E., Tournier, J. D., Calamante, F., and Connelly, A. (2012). Anatomically-constrained tractography: improved diffusion MRI streamlines tractography through effective use of anatomical information. *Neuroimage* 62, 1924–1938. doi: 10.1016/j.neuroimage.2012.06.005

Snoek, L., van der Miesen, M. M., Beemsterboer, T., van der Leij, A., Eigenhuis, A., and Steven Scholte, H. (2021). The amsterdam open MRI collection, a set of multimodal MRI datasets for individual difference analyses. *Sci. Data* 81:85. doi: 10.1038/s41597-021-00870-6

Spiegler, A., Hansen, E. C. A., Bernard, C., McIntosh, A. R., and Jirsa, V. K. (2016). Selective activation of resting-state networks following focal stimulation in a connectome-based network model of the human brain. *eNeuro* 3:ENEURO.0068-16.2016.

Srinivasan, D., Erus, G., Doshi, J., Wolk, D. A., Shou, H., Habes, M., et al. (2020). A comparison of Freesurfer and multi-atlas MUSE for brain anatomy segmentation: findings about size and age bias, and inter-scanner stability in multi-site aging studies. *Neuroimage* 223:117248. doi: 10.1016/j.neuroimage.2020.117248

Sudlow, C., Gallacher, J., Allen, N., Beral, V., Burton, P., Danesh, J., et al. (2015). UK Biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS Med.* 12:e1001779. doi: 10.1371/JOURNAL.PMED.1001779

Taylor, J. R., Williams, N., Cusack, R., Auer, T., Shafto, M. A., Dixon, M., et al. (2017). The cambridge centre for ageing and neuroscience (Cam-CAN) data repository: structural and functional MRI, MEG, and cognitive data from a cross-sectional adult lifespan sample. *Neuroimage* 144, 262–269. doi: 10.1016/j.neuroimage.2015.09.018

Thomas, C., Ye, F. Q., Irfanoglu, M. O., Modi, P., Saleem, K. S., Leopold, D. A., et al. (2014). Anatomical accuracy of brain connections derived from diffusion MRI tractography is inherently limited. *Proc. Natl. Acad. Sci. U.S.A.* 111, 16574–16579. doi: 10.1073/pnas.1405672111

Tian, Y., Margulies, D. S., Breakspear, M., and Zalesky, A. (2020). Topographic organization of the human subcortex unveiled with functional connectivity gradients. *Nat. Neurosci.* 23, 1421–1432. doi: 10.1038/s41593-020-00711-6

Uddin, L. Q. (2017). Mixed signals: on separating brain signal from noise. *Trends Cogn. Sci.* 21, 405–406. doi: 10.1016/j.tics.2017.04.002

Van Essen, D. C., Smith, S. M., Barch, D. M., Behrens, T. E. J., Yacoub, E., Ugurbil, K., et al. (2013). The WU-Minn human connectome project: an overview. *Neuroimage* 80, 62–79. doi: 10.1016/j.neuroimage.2013.05.041

Weiner, M. W., Aisen, P., Petersen, R., Rafii, M., Chow, T., Shaw, L. M., et al. (2016). *Alzheimer's Disease Neuroimaging Initiative 3 (ADNI3) Protocol. 3, 1*. Available online at: https://clinicaltrials.gov/ct2/show/NCT02854033 (accessed February 12, 2022).

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., et al. (2016). Comment: the FAIR guiding principles for scientific data management and stewardship. *Sci. Data* 3:160018. doi: 10.1038/sdata.2016.18

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroinformatics

ORIGINAL RESEARCH
published: 20 June 2022
doi: 10.3389/fninf.2022.853098

A Programmable Ontology Encompassing the Functional Logic of the *Drosophila* Brain

Aurel A. Lazar†, Mehmet Kerem Turkcan*† and Yiyin Zhou†

Department of Electrical Engineering, Columbia University, New York, NY, United States

The *Drosophila* brain has only a fraction of the number of neurons of higher organisms such as mice and humans. Yet the sheer complexity of its neural circuits recently revealed by large connectomics datasets suggests that computationally modeling the function of fruit fly brain circuits at this scale poses significant challenges. To address these challenges, we present here a programmable ontology that expands the scope of the current *Drosophila* brain anatomy ontologies to encompass the functional logic of the fly brain. The programmable ontology provides a language not only for modeling circuit motifs but also for programmatically exploring their functional logic. To achieve this goal, we tightly integrated the programmable ontology with the workflow of the interactive FlyBrainLab computing platform. As part of the programmable ontology, we developed NeuroNLP++, a web application that supports free-form English queries for constructing functional brain circuits fully anchored on the available connectome/synaptome datasets, and the published worldwide literature. In addition, we present a methodology for including a model of the space of odorants into the programmable ontology, and for modeling olfactory sensory circuits of the antenna of the fruit fly brain that detect odorant sources. Furthermore, we describe a methodology for modeling the functional logic of the antennal lobe circuit consisting of a massive number of local feedback loops, a characteristic feature observed across *Drosophila* brain regions. Finally, using a circuit library, we demonstrate the power of our methodology for interactively exploring the functional logic of the massive number of feedback loops in the antennal lobe.

Keywords: *Drosophila melanogaster*, ontology, connectome/synaptome, feedback loops, functional logic, early olfactory system, *in silico* execution, cell type

OPEN ACCESS

Edited by:
Padraig Gleeson,
University College London,
United Kingdom

Reviewed by:
Louis K. Scheffer,
Howard Hughes Medical Institute (HHMI), United States
Max Turner,
Stanford University, United States

***Correspondence:**
Aurel A. Lazar
aurel@ee.columbia.edu

†*The authors' names are listed in alphabetical order*

Received: *12 January 2022*
Accepted: *19 April 2022*
Published: *20 June 2022*

Citation:
Lazar AA, Turkcan MK and Zhou Y (2022) A Programmable Ontology Encompassing the Functional Logic of the Drosophila Brain. Front. Neuroinform. 16:853098. doi: 10.3389/fninf.2022.853098

1. INTRODUCTION

1.1. Challenges in Discovering the Functional Logic of Brain Circuits in the Connectomic/Synaptomic Era

Large scale foundational surveys of the anatomical, physiological and genomic architecture of brains of mice, primates and humans have shown the enormous variety of cell types (Tasic et al., 2018; Grünert and Martin, 2020; Bakken et al., 2021), diverse connectivity patterns with fan-ins and fan-outs in the tens of thousands and extensive feedback that vary both within and between brain regions (Harris et al., 2019). The last decade also saw an exponential growth in neuroscience data gathering, collection and availability, starting with the cubic millimeter brain tissue in mice and humans (Shapson-Coe et al., 2021). However, due to the sheer magnitude and complexity of brains

of higher organisms, even with such data at hand, we are far behind in our understanding of the principles of neural computation in the brain.

Prior studies have highlighted the need for developing means of formally specifying and generating executable models of circuits that incorporate various types of brain data, including the heterogeneity and connectivity of different cells types and brain circuits, neurophysiology recordings as well as gene expression data. In principle, a whole brain simulation can be instantiated by modeling all the neurons and synapses of the connectome/synaptome with simple dynamics such as integrate-and-fire neurons and α-synapses, with parameters tuned according to certain criteria (Huang et al., 2019). Such an effort, however, may fall short of revealing the fundamental computational units required for understanding the functional logic of the brain, as the details of the units of computation are likely buried in the uniform treatment of the vast number of neurons and their connection patterns.

It is, therefore, imperative to develop a formal reasoning framework of the functional logic of brain circuits that goes beyond simple instantiations of flows on graphs generated from the connectome. A framework is needed for building a functional brain from components whose functional logic can be readily envisioned, and for exploring the computational principles underlying these components given the available data.

Recently released connectome, synaptome and transcriptome datasets of the *Drosophila* brain and ventral nerve cord (VNC) present a refreshing view of the study of neural computation (Zheng et al., 2018; Scheffer et al., 2020; Li et al., 2021). These datasets present challenges and opportunities for hypothesizing and uncovering the fundamental computational units and their interactions.

1.2. Modeling the Functional Logic of Fruit Fly Brain Circuits With Cell Types and Feedback Loops

The fruit fly brain can be subdivided into some 40 neuropils. The concept of the local processing unit (LPU) was introduced in the early works of the fly connectome to represent functional subdivisions of the fruit fly brain circuit architecture (Chiang et al., 2011). LPUs are characterized by unique populations of local neurons whose processes are restricted to specific neuropils.

It was not until the release of follow up electron microscopy (EM) connectome datasets that the minute details of the connectivity of these local neurons were revealed (Ohyama et al., 2015; Takemura et al., 2015; Zheng et al., 2018; Scheffer et al., 2020). Oftentimes, local neurons within each neuropil form intricate feedback circuits with a massive number of feedback loops.

For example, the antennal lobe of the early olfactory system, consists of the axons of olfactory sensory neurons (OSNs) as inputs (depicted in **Figure 1A** in darker colors), the antennal lobe projection neurons (PNs) as outputs (in **Figure 1A** in brighter colors), and a large collection of local neurons (in **Figure 1A** in transparent white). The adjacency matrix of the connectivity graph of the AL circuit is shown in **Figure 1B**.

The axons of the OSNs expressing the same olfactory receptor (OR) project into the same glomerulus where they provide inputs to uniglomerular PNs (uPNs) whose dendrites only extend within the same glomerulus. Such connections form the feedforward signaling path in the antennal lobe (see the magenta-colored block in **Figure 1B**).

While not all neuropils share such glomerular structure, three features in the AL connectivity patterns can be found in many other neuropils.

First, OSNs expressing the same OR exhibit strong axon-axonal connections but not with OSNs expressing other ORs (see the cyan-colored block corresponding to the OSN-to-OSN connectivity on the top left of **Figure 1B**). Similar axonal connections can be observed between Kenyon Cells (KCs) of the mushroom body (MB) (Zheng et al., 2018), between Lobular Columnar (LC) neurons in the optic glomeruli (OG) (Scheffer et al., 2020), and between the ring neurons of the ellipsoid body (EB) (Hulse et al., 2021).

Second, local neurons in the AL can be grouped into a large number of cell types. The diversity of the LN cell types and the complexity of their arborization suggest the key role that the LNs play in shaping the functional logic of the AL. Determining the role each of these cell types plays is essential in modeling the functional logic of the AL circuit.

Third, local neurons receive inputs from OSNs and PNs (see green and blue blocks, corresponding to OSN-to-LN and PN-to-LN connectivity, respectively, in **Figure 1B**). They also provide feedback to OSNs and PNs (see red and yellow blocks, respectively, in **Figure 1B**). In addition, LNs also synapse onto other LNs (white block in **Figure 1B**). Given the simplicity of the feedforward signaling path and the complex nature of feedback driven by LN connectivity, the massive number of feedback loops must underlie the functional logic of the AL circuit.

A massive number of feedback loops can be ubiquitously found across other brain regions, for example in the medulla (Takemura et al., 2015), lateral horn, mushroom body (Scheffer et al., 2020), central complex (Hulse et al., 2021), etc. The feedback loops considered here map the states at the output of a circuit into inputs. Since the AL has a connectivity structure that in many ways is representative, for simplicity and clarity in the rest of this work we will be mostly focused on characterizing the AL circuit.

Finally, note that in mammals, particularly in the visual system, feedback pathways have long been considered to be a key component of the architecture of brain circuits (Lamme et al., 1998). However, due to the lack of detailed brain circuit connectivity in these higher organisms there remains insufficient insight into the functional role played by the feedback circuits. The connectome/synaptome of the fruit fly opens new avenues for discovering the full complexity and computational principles underlying feedback circuits.

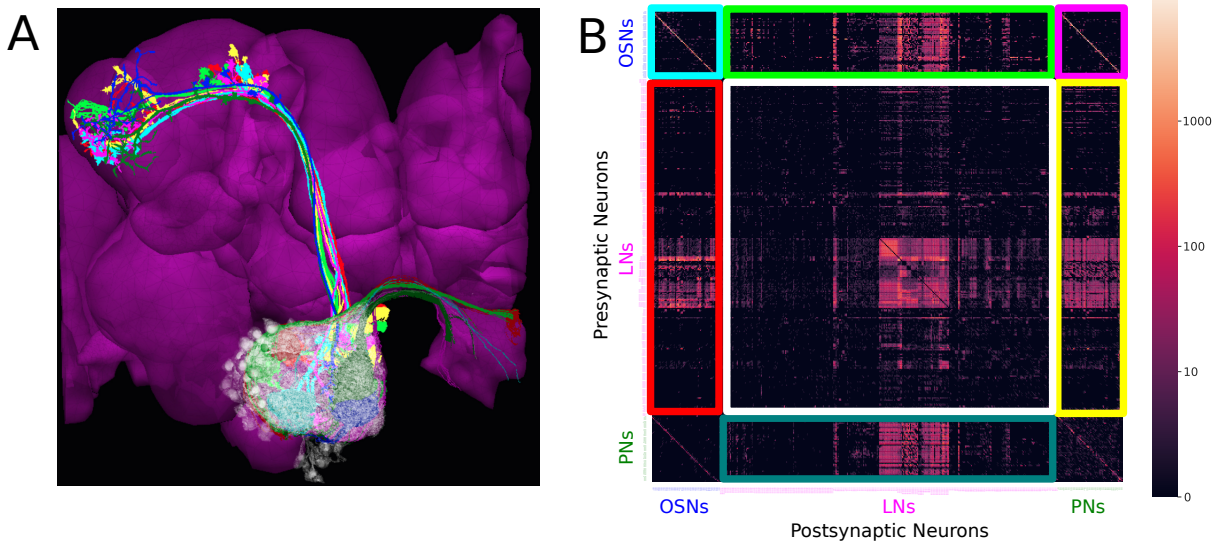


FIGURE 1 | Massive number of feedback loops in the Antennal Lobe. **(A)** Antennal Lobe circuit involving OSNs (darker colors), PNs (brighter colors) and LNs (transparent white). Select OSNs, PNs and LNs are shown. **(B)** The adjacency matrix of the connectivity graph of the neurons in the AL, with all OSNs expressing the same OR merged into a single neuron group node, and all PNs in the same glomerulus merged into a single neuron group node. Matrix elements indicate the number of synapses from a presynaptic neuron (or neuron group) to a postsynaptic neuron (or neuron group). Magenta block on top right: submatrix of the feedforward connectivity from OSNs to PNs in each glomerulus. Green block on the top: submatrix of the feedforward connectivity from OSNs to LNs. Blue block on the bottom: submatrix of the connectivity from PNs to LNs. Red block on the left: submatrix of the feedback connectivity from LNs to OSNs. Yellow block on the right: submatrix of the feedback connectivity from LNs to PNs. White block in the middle: submatrix of the connectivity between LNs.

1.3. A Programmable Ontology Encompassing the Functional Logic of the Fruit Fly Brain Circuits

Traditionally, ontologies formally define the classification of the anatomical structure of the *Drosophila* nervous system and the ownership relationships among anatomical entities (Costa et al., 2013; Lazar et al., 2021). However, existing ontologies lack computational primitives/motifs, such as feedback loops that can be more readily associated with the functional role of brain circuits.

Furthermore, characterizing the functional logic of sensory circuits calls for modeling the environment ("the input") the fruit flies live in. The object structure of the space of natural sensory stimuli that the fruit flies constantly sample has not been discussed in the formal ontology of the fly brain anatomy. Although natural stimuli have been widely used in sensory neuroscience (Egelhaaf et al., 2002; Tootoonian et al., 2012; Jeanne et al., 2018), the modeling of the object structure of the environment (Lazar et al., 2022) has often been neglected in the neuroscience literature at large. The aforementioned object structure is, however, essential in defining, characterizing and evaluating the functional logic of brain circuits.

The goal of the work presented here is to accelerate the discovery of the functional logic of the fruit fly brain circuits. Programmability, in the age of connectomics/synaptomics, is key. Expanding the scope of the classical ontology to encompass the natural sensory stimuli and the functional logic of the *Drosophila* brain circuits bridges the gap between the two fields and greatly benefits both. To boot, a *programmable ontology* will provide a language not only for describing but also for executing the functional modules of, for example, the large number of cell types, the massive number of feedback loops observed in brain circuits, which contribute to making brain function more transparent. We believe that this programmable ontology will provide the foundation for exploring the functional logic of the brain.

The proposed programmable ontology is tightly integrated with the workflow of the interactive FlyBrainLab (Lazar et al., 2021) computing platform, as elaborated in **Figure 2**.

The workflow in **Figure 2** consists of 3 steps. First, 3D visualization (see Section 6) of fly brain morphology data is explored and candidate anatomical structures defining functional units and modules (**Figure 2** left) identified. Second, the candidate biological circuits are mapped into executable circuits that provide an abstract representation of the circuit in machine language (**Figure 2** middle). Third, the devised executable circuits are instantiated for the interactive exploration of their functional logic with a highly intuitive graphical interface for configuring, composing and executing neural circuit models (**Figure 2** right, see Section 6).

The main rationale for the tight integration of the programmable ontology into the FlyBrainLab workflow of discovery is to fully anchor it onto biological data and the worldwide literature that describes it. FlyBrainLab fully supports

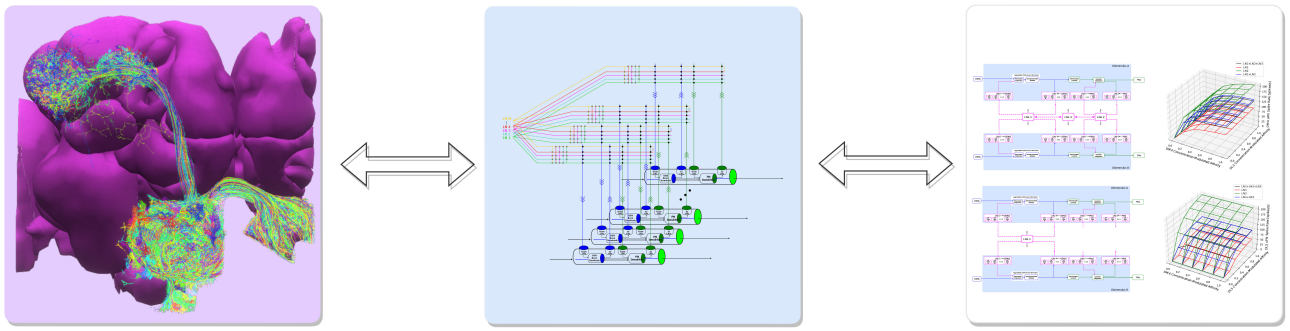


FIGURE 2 | The workflow of discovery of the functional logic of the fruit fly brain. Left: 3D visualization and exploration of fly brain data. Middle: Creation of executable circuits. Right: Interactive exploration of the functional logic of executable circuits.

the programmability of the ontology while easily supporting various computational schemes used for interrogating the functional logic of brain circuits.

2. EXPLORING THE MORPHOLOGY OF CELL TYPES AND FEEDBACK CIRCUITS

Recent releases of large-scale connectomic/synaptomic datasets have enabled experimental and computational neuroscientists to explore neural circuits in unprecedented detail. As **Figure 2** suggests, understanding the functional logic of fruit fly brain circuits starts with the exploration of fly brain connectome/synaptome datasets. To efficiently explore these datasets requires, however, knowledge of both the biological nomenclature and programming tools. These skills are often limited to members of their respective communities. For example, neurobiologists who design new experiments often lack in-depth programming skills to efficiently explore these datasets. Even computational neuroscientists who perform neural circuit simulations may find the need to learn more recent database query languages that go beyond simple operations such as retrieving neurons by name. Computer scientists, working on the next generation of artificial neural networks that are informed by biological neural circuits, need to set aside a significant amount of time to learn the biological nomenclature.

To close the programming gap, we developed the natural language query interface NeuroNLP (Ukani et al., 2019; Lazar et al., 2021) to support highly sophisticated English queries of *Drosophila* brain datasets, including morphology and position of neurons (cell type map), connectivity between neurons (connectome) and distribution and type of synapses (synaptome). Moreover, NeuroNLP provided the first open neurophysiology data service for the fruit fly brain (activity map). However, the NeuroNLP rule-based query engine could only map pre-designed sentence structures into database queries, thereby limiting its usage. In particular, users unfamiliar with the nomenclature used in a dataset may have found it difficult to query for particular cell types.

In what follows, we introduce NeuroNLP++, a substantially upgraded NeuroNLP web application, that alleviates these limitations and helps users to explore fruit fly brain datasets with *free-form English queries*. In Section 2.1, we introduce the capabilities of the NeuroNLP++ application. In Section 2.2, we demonstrate the use of NeuroNLP++ to explore the morphology and graph structure of the cell types in the AL. In Section 2.3, we demonstrate how to use NeuroNLP++ to query feedback loops.

2.1. Key Capabilities of NeuroNLP++

Expanding upon the NeuroNLP query interface (Ukani et al., 2019; Lazar et al., 2021), NeuroNLP++ provides two additional key advances. First, NeuroNLP++ interprets and answers free-form English queries that are well beyond the natural language capabilities of NeuroNLP. Second, NeuroNLP++ not only visualizes neuron/synapses but also links them to the worldwide fruit fly brain literature.

This is achieved by associating descriptive terms of neurons of the fruit fly brain available in the open literature with connectomic datasets. For example, NeuroNLP++ integrates cell types or lineages from the *Drosophila* Anatomy Ontology (DAO) (Costa et al., 2013) and matches them against neurons in the Hemibrain connectome dataset (Scheffer et al., 2020). Given a query, NeuroNLP++ then employs state-of-the-art document retrieval techniques (Karpukhin et al., 2020) to find cell types whose description match the description in the query (see also Section 6).

These descriptions are reflected in the query results of NeuroNLP++ in response to a question also mentioned in the caption of **Figures 3A,B**. Here, we started by asking "what neurons respond to carbon dioxide?". The query results, in the form of a list of the most relevant cell types, are displayed on the left of the NeuroNLP++ user interface, as shown in **Figure 3B**. Each entry lists the name of the cell type, a link to the DAO, as well as a description of the cell type. It also includes a UI button for adding to the workspace the neurons associated with the entry. The first query result, magnified in **Figure 3A**, includes names and synonyms of the V glomerulus projection neurons, as well as their ontological description along with specific entries to the relevant literature.

In **Figures 3C,D**, we present more examples of NeuroNLP++ queries. In **Figure 3C** we asked "which neurons are associated with water reward?," and in **Figure 3D** "what cell types are there

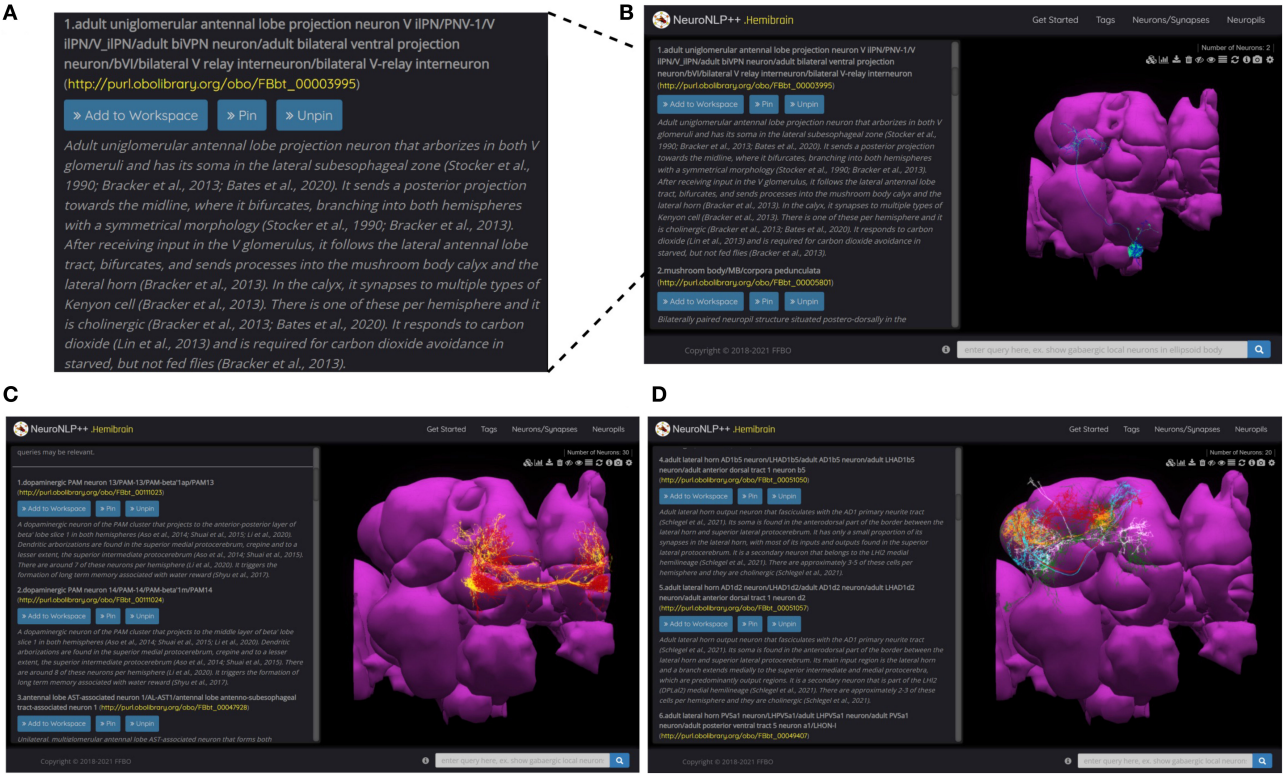


FIGURE 3 | NeuroNLP++ user interface and query results. **(A)** Typical results displayed in the Info Panel of the NeuroNLP++ application in response to the query "which neurons respond to carbon dioxide?". **(B)** The user interface of NeuroNLP++ with the Info Panel on the left and 3D visualization workspace on the right. **(C)** Results to the query "which neurons are associated with water reward?" with the first few relevant entries added to the workspace. (red) PAM13 neurons, (yellow) PAM14 neurons. **(D)** Results to the query "what cell types are there in lateral horn?" with the first few entries added to the workspace. (red) AD1b5 neurons, (green) LH centrifugal neurons, (cyan) AD1d2 neurons, (orange) PV5a1, (white) PD2a5 neurons.

in lateral horn." The query results revealed a variety of cell types. These may provide a starting point for exploring novel cell types associated with other neuropils and can guide additional rule-based queries. The two examples here can be found as live demos in the NeuroNLP++ application.

Compared with using NeuroNLP and other connectome-driven web services such as Neuprint (Clements et al., 2020), users benefit from employing NeuroNLP++ in several ways. First, with NeuroNLP++ neurons can be queried in ways that are not limited to the specific naming in a dataset. For example, a neuron may be named differently in different research papers, while a specific name is used in the Hemibrain dataset. Without knowing the specific name the neuron is called in the working dataset, the user may not be able to find with NeuroNLP the cell type by using a name mentioned in the literature. This knowledge of the nomenclature is not necessary when using NeuroNLP++. In addition, multiple matched results with descriptive answers alleviate the problem with naming ambiguity in English queries, when, for example, an abbreviation of the neuron name can refer to different cells in different brain regions.

Furthermore, NeuroNLP++ complements the specific sentence structure required by NeuroNLP. While the pre-designed sentence structure allows for querying neurons precisely by their properties, NeuroNLP++ allows questions to be more "open ended". For example, querying cell types is not limited to their names, but a user can ask questions such as "what types of local neurons are in the antennal lobe?" and "what are the ring neurons?."

Finally, NeuroNLP++ provides more context for the neurons searched in connectomic datasets by providing links to the worldwide literature associated with cell types. This will provide users, particularly those unfamiliar with the cell type literature, a convenient way of exploring prior knowledge.

2.2. Exploring the Morphology and Graph of Cell Types With NeuroNLP++

In addition to natural language querying capabilities, NeuroNLP++ provides an interactive Graph View application that displays the current neurons in the workspace at the neuronal or cell type level (see Section 6). While the morphology of neurons is often cluttered in the 3D visualization, Graph View helps sort out the connections between the neurons displayed.

Here we use NeuroNLP++ to explore the morphology and Graph View to visualize the circuit diagram of several glomeruli in the AL. We started by asking "what are the cell types of the DL5 glomerulus". The cell type Graph View of the neurons in the DL5 glomerulus is depicted on the left in **Figure 4A**. Here, the red and the yellow nodes represent the OSNs and PNs, respectively,

FIGURE 4 | Free-form English queries of the AL with NeuroNLP++. **(A)** Result to the query "what are the cell types of the DL5 glomerulus?", consisting of the OSNs with axons that arborize the DL5 glomerulus (red) and the PN with dendrites in the DL5 glomerulus (yellow). (left) Cell type connectivity graph of the visualized neurons. (right) Morphology of the retrieved neurons. **(B)** Result to the query "what are the cell types of the DM4 glomerulus?", in addition to **(A)**, consisting of the OSNs (orange with axons that arborize the DM4 glomerulus and the adPNs (cyan) and vPNs (blue) with dendrites in the DM4 glomerulus. (left) Cell type level connectivity graph of the resulting neurons. (right) Morphology of the retrieved neurons. **(C)** Results to the query "what are the patchy local neurons?", in addition to **(B)**. The resulting LNs are in pink. (left) Cell type connectivity graph of the visualized neurons. (right) Morphology of the retrieved neurons.

and the arrow from the red to the yellow node indicates that the OSNs provide inputs to the PNs. The colors in Graph View match those in the 3D morphology visualization. Graph View is also interactive, allowing users to highlight the corresponding neurons in the 3D visualization.

We then asked "what are the cell types in the DM4 glomerulus". The resulting neurons are added to the workspace and their cell type graph is depicted in **Figure 4B**. These include the OSNs (orange) that project into the DM4 glomerulus and two types of PNs, namely adPNs (cyan) that project to both the MB and LH, and vPNs (blue) that only project to the LH. The graph also confirms that the two glomeruli run in parallel.

Finally, we asked "what are the patchy local neurons?". Patchy local neurons, typically arborize in a large number of glomeruli of the AL. They were first discovered in a light microscopy study of local neurons; information about their connectivity with neurons inside the glomeruli is lacking (Chou et al., 2010). The neurons obtained in response to our query are shown in white in **Figure 4C**, together with the cell type graph of the entire circuit. The connectivity graph suggests the presence of strong feedback components within and between the two otherwise disjoint glomerular circuits.

To explore the diversity of LNs in the AL, we needed to classify cell types based on their morphology. We launched the query "what are the types of local neurons in the antennal lobe?." NeuroNLP++ provides a complete list of the currently known LN types in the AL. In **Supplementary Figure S1** in **Supplementary Material**, we list all these LN types, the number of neurons of each type, and an example morphology of neuron type. The morphology of the neurons is colored by their glomerular arborization. The graph structure of a typical LN of each cell type is summarized in the matrix depicted on the right.

2.3. Exploring the Morphology of Feedback Circuits With NeuroNLP++

As discussed above, feedback loops are major targets of the study of the functional logic of the fruit fly brain. Consequently, in addition to querying cell types, we also built into NeuroNLP++ capabilities to query for neurons that belong to specific feedback loops.

Different feedback loops can be described as entities in the DAO. This enables NeuroNLP++ to search for feedback loops with English queries. We identified different types of feedback loops for each glomerulus (see Section 6) and further identified a number of specific feedback loops consisting of local neurons. For example, a circuit consisting of LNs that receive inputs from, and provide feedback to, OSNs but has no interaction with PNs, is named here an OSN-LN-OSN feedback loop. Similarly, a circuit consisting of LNs that receive inputs from, and provide feedback to, PNs but has no interaction with OSNs is named a PN-LN-PN feedback loop. A circuit consisting of LNs that receive inputs from and provide feedback to both OSNs and PNs is named an OSN/PN-LN-OSN/PN feedback loop.

To query for feedback loops associated with OSNs and PNs in the DL5 glomerulus, i.e., starting from **Figure 4A**, we requested: "show available feedback loops." **Figure 5** depicts two different types of feedback loops in response to this query.

In **Figure 5A**, we added the 19 LNs (pink) that form the PN-LN-PN feedback loop. From Graph View, we confirm that these feedback loops are only associated with the DL5 PNs (yellow) but not OSNs (red) node. The arrows into the adPN node indicate the feedback pathway from LNs into the adPN. In **Figure 5B**, we added the 26 LNs that form feedback loops with both OSNs and PNs. From Graph View we note that these LNs (pink) form feedback loops with both OSNs (red) and PNs (yellow) nodes.

Concluding, by establishing the NeuroNLP++ natural language query interface for exploring the morphology of fruit fly brain circuits, we effectively created an ontology of the fruit fly brain consisting of the existing anatomical ontology, the connectome/synaptome datasets and the published worldwide literature. Moreover, we provided visualization tools for extracting what are thought to be functionally significant circuits. NeuroNLP++ represents a step toward a more intuitive and natural way of extracting information from large connectome/synaptome datasets that are relevant for the in-depth study of the functional logic of brain circuits. In addition, the capability to anchor the queried connectome/synaptome data onto the published worldwide literature provides much needed awareness of the prior existing knowledge regarding these circuits.

3. CREATING A PROGRAMMABLE ONTOLOGY OF THE FRUIT FLY BRAIN

Our goal in this section is to demonstrate how the framework of the ontology outlined in the previous section can be further enriched and extended to encompass the key stimulus and processing elements needed for exploring the functional logic of the fruit fly brain circuits. Overall, the resulting ontology will be programmable from the ground up. Programmability calls for i) employing spaces of stimuli whose basic objects can be computationally modeled and identified, and ii) constructing brain circuit models using simple executable building blocks that are composable based on rules built upon and informed by biological entities such as cell types and feedback loops.

The significance of modeling the space of stimuli for characterizing the I/O of functional circuits arises throughout the early sensory systems, e.g., in early olfaction (Jeanne et al., 2018), vision (Egelhaaf et al., 2002), audition (Tootoonian et al., 2012), mechanosensation (Tuthill and Wilson, 2016), etc. The odorant space and the visual field are examples that come to mind. See, for example, Lazar et al. (2015) and Lazar and Yeh (2020).

Given the current connetomic datasets, we shall describe here how some of the better characterized neuropils can be modeled and constructed through a process of composability. Due to space limitations, we will only present in what follows a methodology of a receptor-centric modeling of the space of odorant stimuli, as well as a methodology for devising the olfactory processing in the antenna and the antennal lobe of the fruit fly brain. How to apply the same general methodology to other neuropils of the fruit fly brain entails a set of challenges that will be addressed elsewhere.

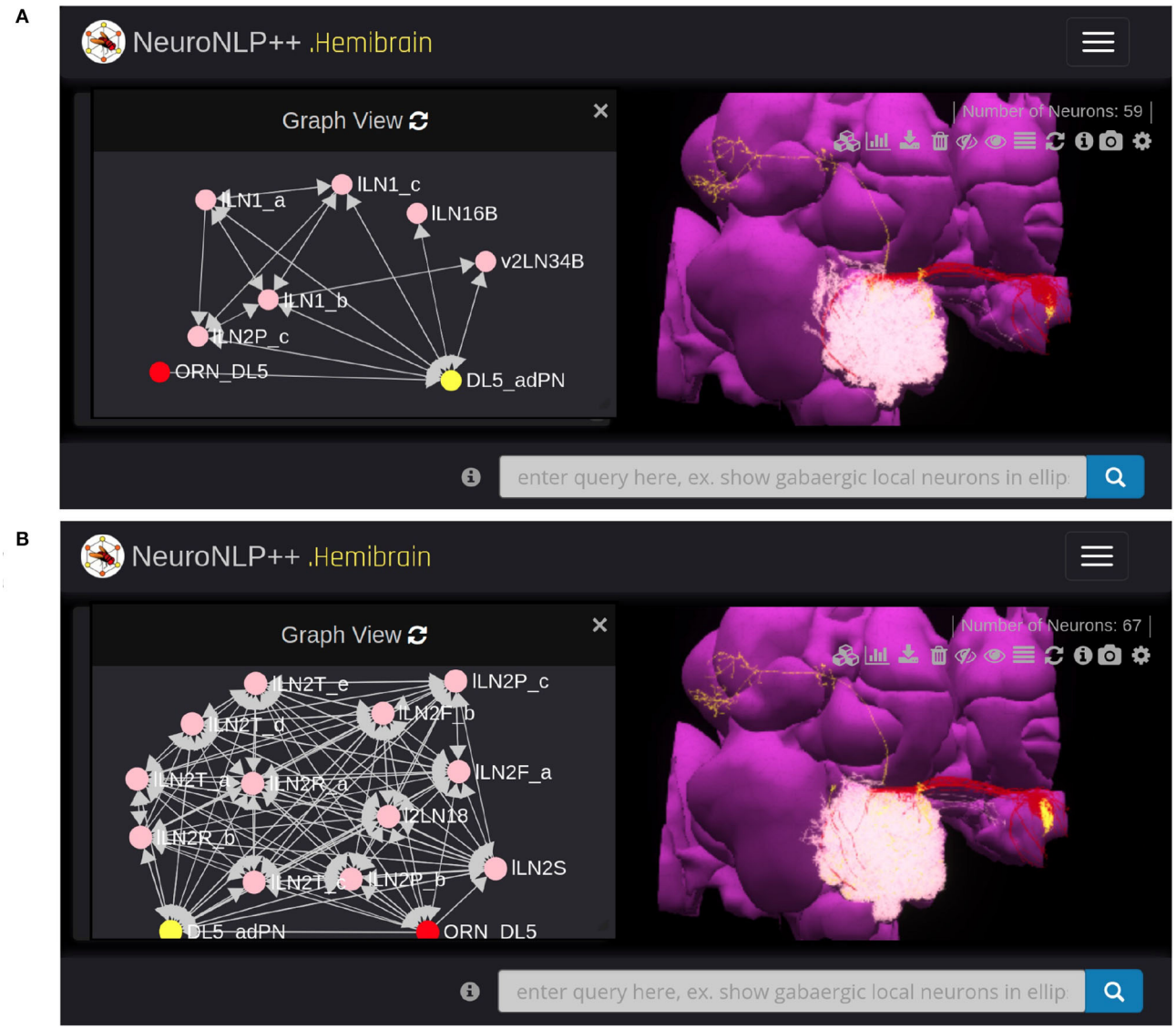


FIGURE 5 | Exploring feedback loops with NeuroNLP++. The result to the query "show available feedback loops" when starting from **Figure 4A** with OSNs with axons that arborize the DL5 glomerulus and PNs with dendrites in DL5 glomerulus. **(A)** PN-LN-PN feedback loop (receives inputs from and only feeds back to PNs). (red) OSNs. (yellow) PNs. (pink) LNs. **(B)** OSN/PN-LN-OSN/PN feedback loop (LNs receive inputs from both OSNs and PNs and provide, respectively, feedback to them). (red) OSNs. (yellow) PNs. (pink) LNs.

3.1. Receptor-Centric Modeling the Space of Odorant Stimuli

To fully characterize the functional logic of a sensory circuit calls for modeling the environment the studied organism lives in, a rather difficult undertaking. To model the environment, we first have to define the *space of odorant stimuli*. The space of odorant stimuli has never been discussed in the context of a formal ontology of the fruit fly brain anatomy. It is often neglected in the neuroscience literature, but essential in defining, characterizing and evaluating the functional logic of brain circuits involved in odor processing.

The Chemical Abstracts Service (CAS) registry has currently 156 million organic and inorganic substances registered (Morgan, 1965). Distinguishing between odorants in the CAS registry seems to be a problem of enormous complexity (Tran et al., 2019). How does the fly approach this problem? As a first step in the encoding process, the odorant receptors bind to the odorants present in the environment and that are of interest to the fly. The adult fruit fly has some 51 receptors whose binding and dissociation rates to/from odorant molecules characterize their identity. In addition to odorant identity, the odorant concentration amplitude is another key feature of the odorant space.

The odorant space considered here consists of pure and odorant mixtures. Pure odorants are mostly used in laboratory settings for studying the capabilities and the function of the early olfactory circuits. Odorant mixtures widely arise in the living environment. Following (Lazar and Yeh, 2020), the identity of an odorant can be modeled by a 3D tensor pair $(\mathbf{b}, \mathbf{d})$. The 3D tensor $\mathbf{b}$ with entries $[\mathbf{b}]_{ron}$ is called the odorant-receptor binding rate and models the association rate between an odorant o and a receptor of type r expressed by neuron n (see also **Figure 6**). The 3D tensor $\mathbf{d}$ with entries $[\mathbf{d}]_{ron}$ denotes the *odorant-receptor dissociation rate* and models the detachment rate between an odorant o and a receptor of type r expressed by neuron n (see also **Figure 6**). We denote the odorant concentration waveforms as the vector $\mathbf{u}(t)$, where $[\mathbf{u}]_o(t)$ denotes the concentration amplitude of odorant $o, o = 1, 2, \cdots, O$. The odorant concentration can be any arbitrary continuous waveform (see also **Figure 6**). For a pure odorant $\mathcal{O}$, $[\mathbf{u}]_o(t) = 0, o \neq \mathcal{O}$. A set of odorant waveforms modeled by the tensor trio $\left(\mathbf{b}, \mathbf{d}, \mathbf{u}(t)\right)$ is graphically depicted in **Figure 6**. Often, for simplicity, the binding rate $[\mathbf{b}]_{ron}$ and the dissociation rate $[\mathbf{d}]_{ron}$, for a given odorant o and a given receptor-type r, are assumed to take the same value for all neurons $n = 1, 2, ..., N$, expressing the same receptor-type r.

Note that the elements of the odorant space are not defined by the (largely intractable) detailed/precise chemical structure of the odorants. Rather, they are described by the rate of activation/deactivation between odorants and olfactory receptors. The tensor trio determines what types of sensors (olfactory receptors) will be activated by a certain odorant, and the level of activation will be jointly governed by the identity and the concentration waveform amplitude of the odorant. More precisely, for a single odorant, the overall activation of the sensors is determined by the value of the odorant-receptor binding rate modulated by the odorant concentration profile (Lazar and Yeh, 2020).

3.2. Building the Antenna Circuit With OSN Cell Types

The antenna circuit of the early olfactory system of the fruit fly consists of approximately 2,500 parallel Olfactory Sensory Neurons (OSNs) that are randomly distributed across the surface of the maxillary palps and antennae. In what follows, we will refer to the set of all OSNs on one side of the fruit fly brain as an antenna/maxillary palp (ANT) local processing unit (LPU).

The OSNs, depicted in **Figure 6** (right) in groups based on the olfactory receptors (ORs) that they express, form parallel circuits. For simplicity, we assumed that the number of OSNs expressing the same receptor-type is N. OSNs in the same group are said to be of the same cell type.

For each OSN, the odorants are first transduced by an olfactory transduction process (OTP) that depends on the receptor-type (Lazar and Yeh, 2020). Each of the generated transduction currents drive biophysical spike generators (BSGs) that produce spikes at the outputs of the antennae (see Section 6). Note that unlike the OTP whose I/O characterization depends on the receptor-type, the BSGs of OSNs expressing different receptor-types are assumed to be the same.

3.3. Composing the Antennal Lobe Circuits With Cell Types and Feedback Loops

The overall goal of this section is to develop a methodology for modeling and constructing circuits of arbitrary complexity of the Antennal Lobe. The methodology demonstrated here is generalizable to the other neuropils in the early olfactory system of the fruit fly brain, including the mushroom body and the lateral horn; due to space limitations, the application of this methodology to the other neuropils of the early olfactory system will be presented elsewhere.

3.3.1. Modeling Individual Glomeruli of the Antennal Lobe Circuit

As sketched in **Figure 4**, the AL exhibits a glomerular structure. Each glomerulus is primarily driven by the feedforward connections between the OSNs expressing the same OR and the corresponding PNs. As already mentioned, in addition to OSNs and PNs, LNs are the third cell type in each glomerulus. Although the modeling of glomeruli presented in this section is rather general, the concrete examples given below revolve around the DM4 and DL5 glomeruli.

To model a glomerulus, we closely followed the connectomic data provided by the Hemibrain dataset (Scheffer et al., 2020). In what follows the emphasis will be on showing how to extract/model the connectivity among cell types. We created a circuit diagram as depicted in **Figure 7E**. Here we abstracted the group of OSNs with axons that arborize the glomerulus as a single OSN (cell type). Similarly, we abstracted the group of PNs with dendrites in the same glomerulus as a single PN (cell type) (trivially obtained for DL5 since this glomerulus features a single PN). We only considered here the PNs that send their axons to both the MB and LH, and, thereby, primarily omit the vPNs with axons that only arborize the LH but not the MB. As shown in **Figure 4C** for the DM4 glomerulus, the omitted PNs typically receive inputs from other PNs rather than OSNs.

Since the OSN axon terminals and PN dendrites arborize the respective glomerulus, their synaptic connections with LNs must also occur within the same glomerulus. This is detailed in the examples in **Figures 7A–D** for the DL5 glomerulus. OSNs with axons arborizing in the DL5 glomerulus are shown in green, the PN with dendrites arborizing the same glomerulus is shown in blue, and 4 different LNs are shown in magenta, respectively. The locations of synapses between the OSNs and the LNs, and between the PN and the LNs are respectively shown in colored circles. The LN in **Figure 7A** receives inputs from the OSNs (cyan dots), provides inputs to the OSNs (red dots), receives inputs from the PN (yellow dots) and provides inputs to the PN (white dots). The LN in **Figure 7B** does not provide inputs to the OSNs; the LN in **Figure 7C** lacks PN inputs; the LN in **Figure 7D** does not synapse onto and receives no inputs from the OSNs shown.

Therefore, in the circuit diagram of the glomerulus in **Figure 7E**, we included 4 types of connections between OSNs and LNs and between PNs and LNs, including i) LNs presynaptically linked with OSN axon terminals, ii) LNs receiving inputs from

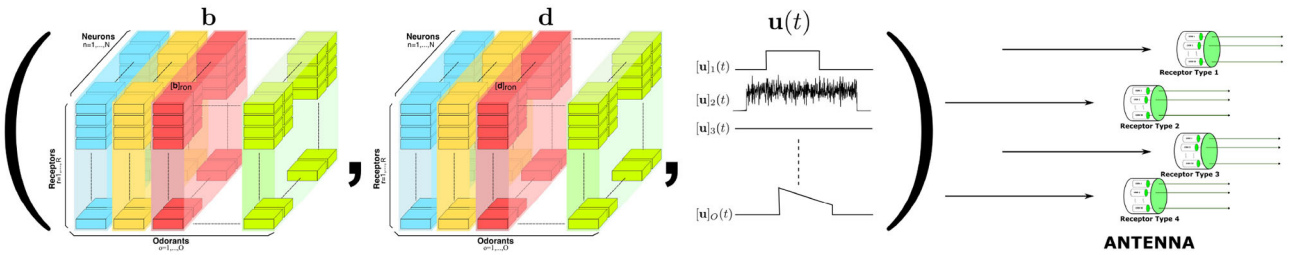


FIGURE 6 | Elements of the odorant space are defined by the odorant-receptor binding rate, dissociation rate and concentration amplitude tensor trio $[b, d, u(t)]$. For a given neuron $n = 1, 2, ..., N$, the binding rate and dissociation rate values are, respectively, denoted by $[b]_{ron}$ and $[d]_{ron}$, for all $r = 1, 2, ..., R$, and $o = 1, 2, ..., O$. Single and/or odorant mixtures interact with the receptors expressed by the Olfactory Sensory Neurons in the Antenna (right).

OSN axon terminals, iii) LNs providing inputs to PNs, and iv) LNs receiving input from PN dendrites. Within the glomerulus, however, we do not specify the exact LNs that carry out these interactions. Rather, we define 4 ports (see the magenta blocks in **Figure 7E**): i) LNs (→OSNs), ii) LNs (←OSNs), iii) LNs (→PNs), and iv) LNs (←PNs), corresponding to, respectively, the 4 types of connections mentioned above. The connections from/to the specific LNs will be defined through these ports. All the LNs that connect to each port carry out the specific port connectivity pattern within the glomerulus.

The innervation of an LN within a glomerulus can then be graphically composed using the 4 ports. There are 15 different patterns of port connectivity within a glomerulus, that we call LN port connectivity patterns. We use a 4 digit binary code to represent this connectivity, according to the left-right order of the ports in **Figure 7E**. For example, if an LN receives inputs from OSNs and provides feedback to the same OSNs, but has no input from or output to PNs, then we call this port connectivity pattern "1100." The number of occurrences of each port connectivity pattern according to the Hemibrain dataset can be found in **Supplementary Table S1** in **Supplementary Material**. Note that a single LN can engage into different port connectivity patterns with different glomeruli.

If the LN innervation in a glomerulus follows the port connectivity pattern 11xx, xx11 or 1xx1, then the said LN is considered to form a feedback loop within the glomerulus. Eight out of 15 port connectivity patterns are associated with feedback, and they are shown in **Figure 7E**. The rest of the port connectivity patterns are only involved in feedback across glomeruli (see below).

3.3.2. Modeling and Constructing Interconnected Glomeruli of the Antennal Lobe Circuit

For individual glomeruli, we have introduced 4 ports whose composability in the form of port connectivity patterns allow us to construct local feedback circuits. Here, we introduce composition rules of interconnected glomeruli that are also based on port connectivity patterns. The composability of port connectivity patterns enable scaling to multiple glomeruli. Furthermore, their programmability strengthens the reach of exploration of the functional logic of models of brain circuits.

To compose the "wiring diagram" of glomeruli, we define feedback motifs using the port connectivity patterns that an LN links and that belong to distinct glomeruli. Here, we provide a number of example feedback motifs based on two interconnected glomeruli, as depicted in **Figure 8C**.

The first example feedback motif is based on LNs that link the port connectivity pattern 0011 of each of the 2 glomeruli. This is denoted LN2 in **Figure 8C**. An instance of such an LN is depicted in **Figure 8A**. The second example feedback motif is an LN that links the port connectivity pattern 1111 of each of the 2 glomeruli. An instance of such LN is depicted in **Figure 8B** (omitted in **Figure 8C**). Continuing to do so, we can create a collection of such feedback motifs based on combinations of port connectivity patterns. Note that the patterns are not necessarily the same on both glomeruli.

Composability also allows us to create feedback motifs that are not present in the connectome but that can still be of interest in studying the computational role of certain feedback loops. For example, the feedback motif presented as LN1 in **Figure 8C** has the port connectivity pattern 1100 of each of the 2 glomeruli.

Finally, we define a feedback motif LN3 that models the port connectivity between LNs. The LN3s do not receive or feedback to either OSNs or PNs. Rather, they connect only with the other feedback motifs.

4. EXPLORING THE FUNCTIONAL LOGIC OF FEEDBACK CIRCUITS IN THE ANTENNAL LOBE

In this section, we present an approach for exploring the functional logic of feedback circuits of the fruit fly brain. This pertains to the third column of the workflow diagram of **Figure 2**. Specifically, we present here the interactive exploration of the AL following the previous sections where the morphology of the AL feedback circuits has been explored (Section 2) and single as well as pairwise interconnected glomeruli modeled and constructed (Section 3) (see also the second column of **Figure 2**).

We describe a programming library for instantiating examples of Antennal Lobe cell types and feedback circuit motifs abstracted from connectome data with customizable

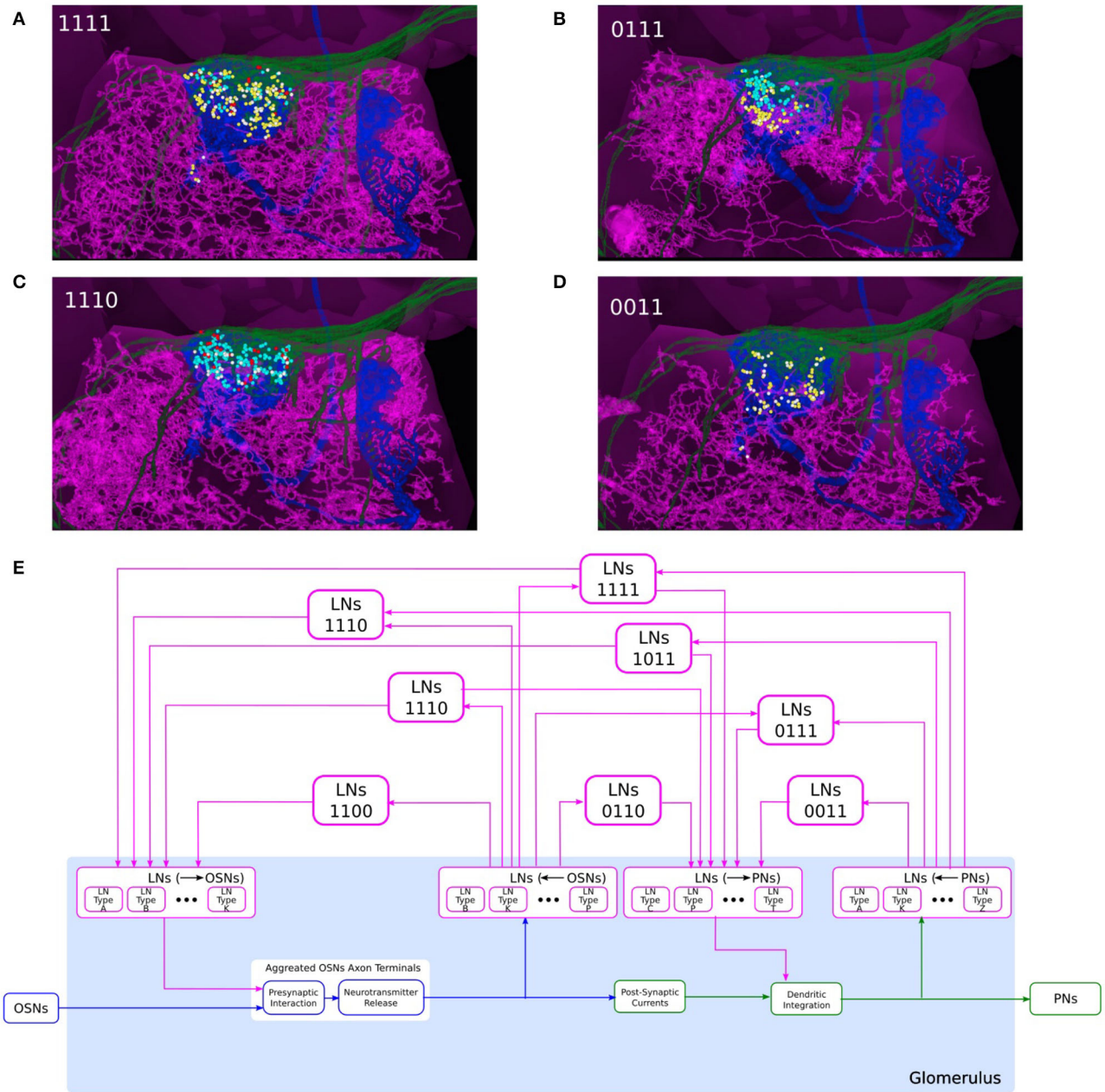


FIGURE 7 | Composability of the port connectivity of LNs within a glomerulus. **(A–D)** Examples of LNs with different port connectivity patterns in the DL5 glomerulus. **(A)** 1111, **(B)** 0111, **(C)** 1110, and **(D)** 0011 (see also (E)). (magenta) LN, (green) the OSNs that project to DL5 glomerulus, (blue) the PN that innervates the DL5 glomerulus, (cyan dots) locations of OSN-to-LN synapses, (red dots) locations of LN-to-OSN synapses, (yellow dots) locations of PN-to-LN synapses, (white dots) locations of LN-to-PN synapses. **(E)** Schematic diagram of a single glomerulus circuit. (bottom) The blocks at the very bottom of the glomerulus represent the feedforward circuits. (top) LNs with different port connectivity patterns. Port connectivity patterns that have less than 50 occurrences in the Hemibrain dataset are omitted (see also **Supplementary Table S1** in **Supplementary Material**).

parameters of neurons and synapses. We demonstrate in Section 4.1 the use of this circuit library in exploring the I/O of a single glomerulus and in Section 4.2 for a pair of interconnected glomeruli. In Section 4.3, we then provide an example of scaling the methodology presented here for exploring the functional logic of the entire AL circuit.

4.1. Exploring the Functional Logic of Feedback Circuits of a Single Glomerulus in Isolation

For clarity in the presentation and simplicity in evaluation, in this section we build upon the composability of LNs within a glomerulus detailed in Section 3.3 and shown in **Figure 7**.

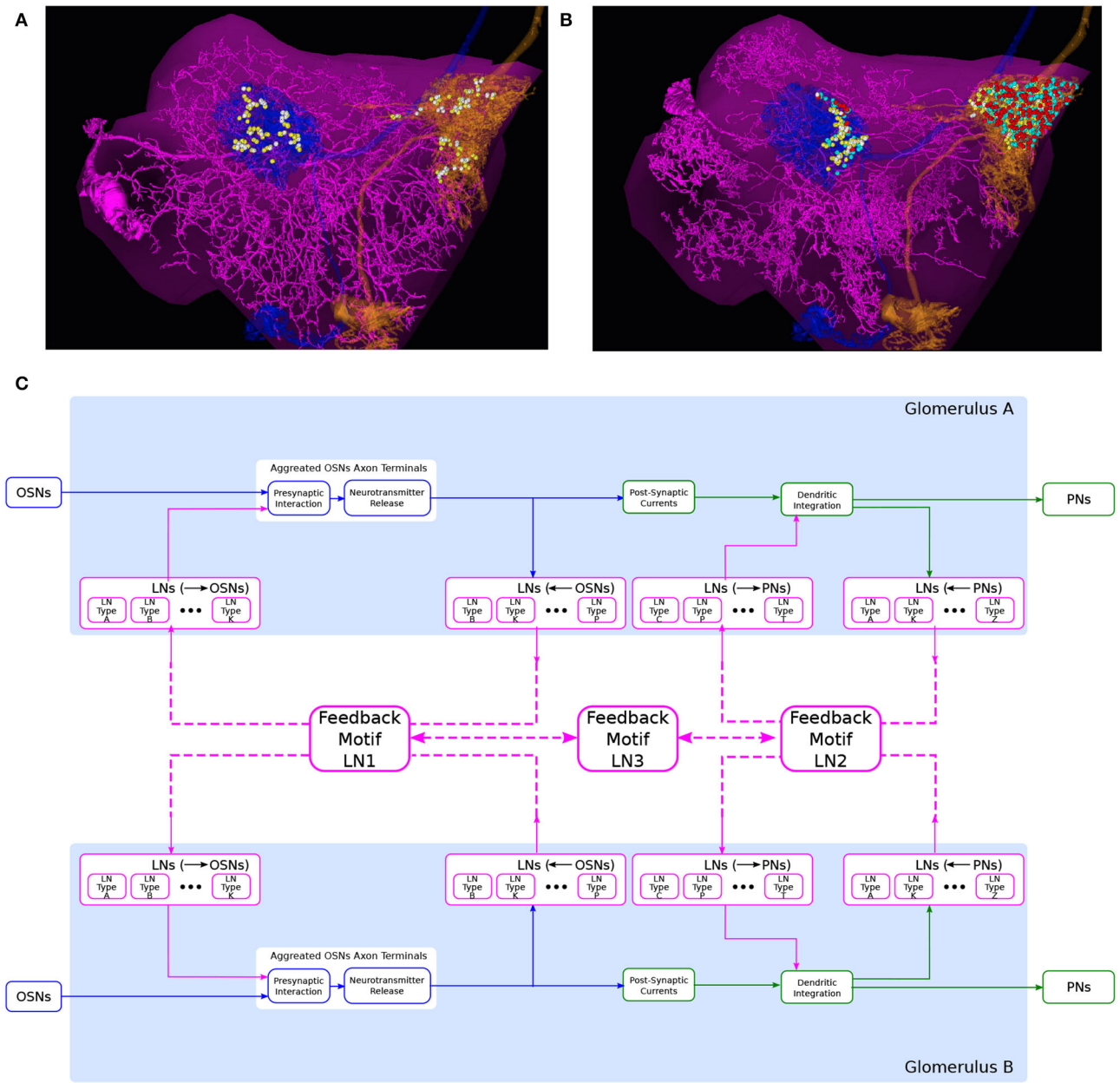


FIGURE 8 | An abstraction of glomerular feedback circuits with 3 feedback motifs. **(A)** An LN that is presynaptic and postsynaptic to the PNs, but not to OSNs, in both DL5 and DM4 glomeruli. (magenta) the LN, (blue) a PN with dendrites arborizing the DL5 glomerulus, (orange) a PN with dendrites arborizing DM4 glomerulus, (yellow dots) locations of PN-to-LN synapses, (white dots) locations of LN-to-PN synapses. **(B)** An LN that is presynaptic and postsynaptic to the OSNs and PNs in both DL5 and DM4 glomeruli. Color is the same as in **(A)** and additionally, (cyan dots) locations of OSN-to-LN synapses, (red dots) locations of LN-to-OSN synapses. **(C)** A circuit diagram of two glomeruli with 3 feedback motifs. Feedback motif LN1 corresponds to the port connectivity patterns 1100 of each of the two glomeruli. Feedback motif LN2 corresponds to the port connectivity patterns 0011 of each of the two glomeruli, e.g., **(A)**. Feedback motif LN3 models the bidirectional connectivity between LN1 and LN2.

In **Figure 9**, we evaluate the I/O behavior of the DM4 and DL5 glomeruli separately and in isolation for different compositions of feedback motifs. We outline how the presence of different feedback motifs can jointly or individually alter the PN output of the glomeruli. In our experimental setup, the number of OSNs and PNs, as well as the number of synapses between a given OSN and PN are configured using the data from the Hemibrain connectome dataset.

In **Figures 9A,C**, we evaluate and compare the DM4 glomerulus response due to different compositions of feedback motifs. We added to the single DM4 glomerular circuit composed of the OSNs, PN and three feedback motifs: LN1, LN2, and LN3

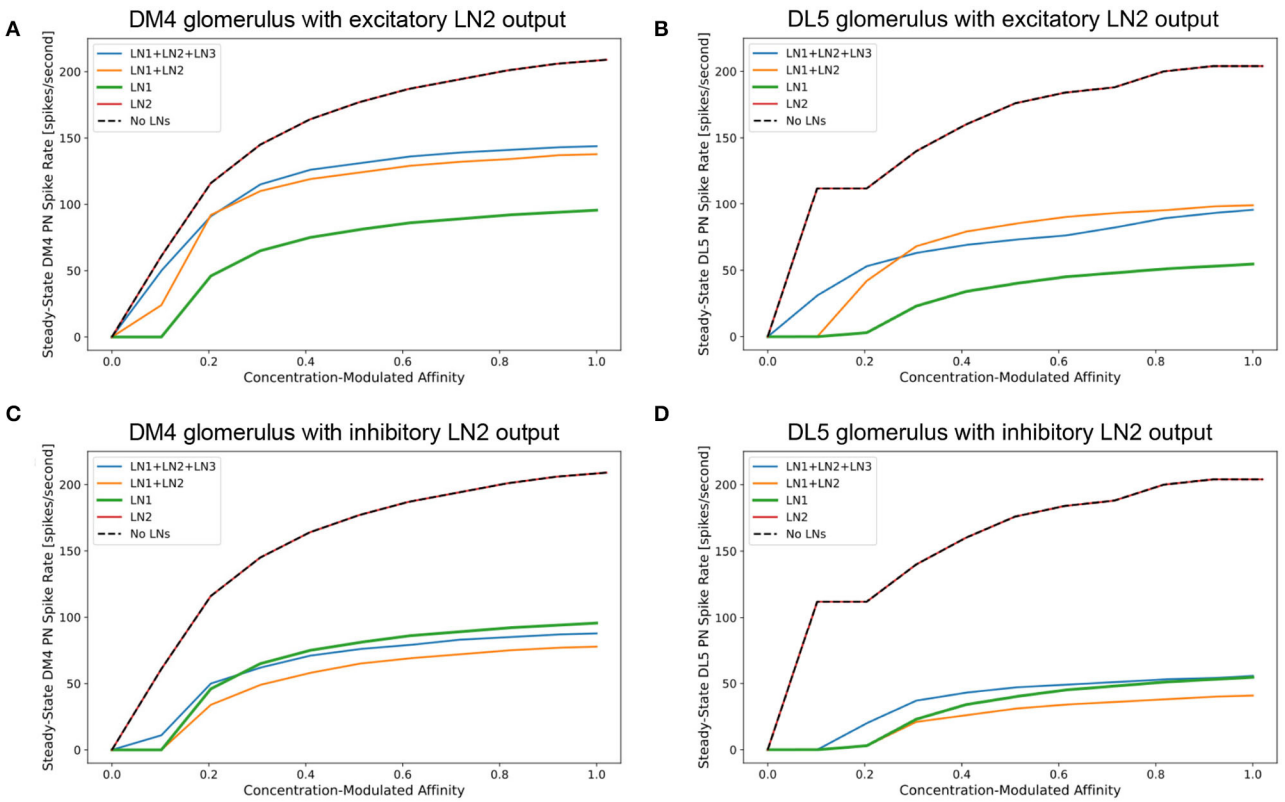


FIGURE 9 | Characterization of PN responses of the single (isolated) DM4 and DL5 glomerular circuits. **(A)** DM4 PN steady-state firing rate across different constant odorant concentration levels. (dashed black) No LN is present. (green) Only LN1 feedback motif is present. (red) Only LN2 feedback motif is present. (orange) LN1 and LN2 feedback motifs are present. (blue) LN1, LN2 and LN3 feedback motifs are present. **(B)** DL5 PN steady-state firing rate across different constant odorant concentration levels. LN2 feedback motifs in both **(A,B)** are assumed to be excitatory. **(C)** DM4 PN steady-state firing rate when LN2 feedback motifs are assumed to be inhibitory. **(D)** DL5 PN steady-state firing rate when LN2 feedback motifs are assumed to be inhibitory.

(see also Section 6). Results shown in **Figure 9A** were obtained assuming that the output of the feedback motif LN2 is excitatory, while for those in **Figure 9C** the output of the feedback motif LN2 is inhibitory.

For constant odorant concentration waveforms, the steady-state firing rates of the corresponding PNs are displayed in **Figure 9A** (see also Section 6). For the range of modulated affinity values, if the glomerulus is configured without any feedback loops, then the PN is driven immediately to saturation (**Figure 9A** dashed black curve). The addition of the feedback motif LN1, that presynaptically inhibits OSNs, results in a sigmoidal PN spiking rate for the tested range of odorant concentration values (**Figure 9A** green curve). We also note that the addition of the feedback motif LN2 alone, either with excitatory or inhibitory output, does not directly contribute to regulating the PN response, and results in a saturation level similar to the circuit without feedback (**Figures 9A,C** red curves).

We evaluated next the effect of the composition of feedback motifs. The orange curves in **Figures 9A,C** depict the responses of DM4 PNs when feedback motif LN2 with, respectively, excitatory and inhibitory outputs is added to the circuit with the feedback motif LN1 already present. Finally, we added feedback motif LN3 to the previous setup and stimulated it externally with a 20 nA current source. Injecting an external current enabled us to explore how activation of this feedback motif affects the responses of the DM4 PNs; this results in regular spiking in LN3 and suppression of both feedback motifs LN1 and LN2. However, the suppression of LN2 causes a larger effect and thus a net decrease in the PN spiking rate. These simple explorations demonstrate the effect different feedback loop compositions might have on the responses of the one-glomerulus circuit.

Evaluations with the same set of compositions were performed on the feedback circuit of the DL5 glomerulus (**Figure 9B** where output of LN2 is excitatory and **Figure 9D** where output of LN2 is inhibitory). Here, we observed similar contributions from different compositions of the feedback motifs as in the case of the DM4 glomerulus. Results for DM4 and DL5 if the output of LN2 is inhibitory are shown in **Figures 9C,D**, respectively. As expected, in this scenario, removing feedback motif LN2 causes a higher spike rate of the PN. An indirect suppression of LN2 through stimulation of LN3 similarly raises the spike rate of the PN.

The comparison of the PN outputs using different feedback motifs (**Figures 9A–D**) shows that i) the feedback motif LN1

is essential for the circuit to be stable under a large range of concentration amplitude values, ii) the addition of the feedback motif LN2 amplifies the steady-state PN spike rate response given the presence of the feedback motif LN1, and feedback motif LN3 controls the contribution of LN1 and LN2 to the I/O behavior of the circuit.

4.2. Exploring the Functional Logic of Feedback Circuits of a Pair of Interconnected Glomeruli

For clarity in the presentation and simplicity in evaluation, we shall use in this section the feedback circuit motif examples of the pair of interconnected glomeruli presented in Section 3.3 and detailed in **Figure 8**.

In **Figure 10**, we evaluated a circuit consisting of a pair of interconnected DM4 and DL5 glomeruli. A circuit diagram of the interconnected DM4 and DL5 glomeruli is shown in **Figure 10A**. The number of OSNs and PNs in these two glomeruli, and the number of synapses between these two types of neurons are configured according to the Hemibrain dataset. Following the motifs we explored in **Figure 8**, we then added five feedback circuit motifs: 1 LN1 each connecting to only DM4 and DL5, 1 LN2 each connecting to only DM4 and DL5, and 1 LN3 bidirectionally connected to LN1 and LN2 (see also Section 6).

In **Figures 10B,C**, we show the average spike rate of, respectively, the DM4 PN and DL5 PN as a function of the input value. Mirroring **Figure 9**, we consider compositions of different feedback circuit motifs. As in **Figure 9**, we find that the addition of LN1 alone produces the lowest spiking rate (**Figures 10B,C** red mesh). Similarly, the addition of LN2 alone produces the highest spiking rate due to lack of presynaptic inhibition from LN1 (**Figures 10B,C** green mesh). When we added feedback motif LN2 in addition to feedback motif LN1, the excitatory nature of the LN2 loop resulted in a spike rate increase by a small amount when compared with the DM4 glomerulus with only the feedback motif LN1 (**Figures 10B,C** blue mesh). Finally, by adding the feedback motif LN3 to the previous setup, the spike rate for the DM4 or DL5 glomerulus increases with the affinity of the receptors expressed by the OSNs projecting into the respective glomerulus. Note that, an increase in the odorant amplitude waveform of the OSNs projecting to one of the glomeruli also affects the spike rate of the other through the feedback motif LN3 (**Figures 10B,C** black mesh).

4.3. Circuit Library for Exploring the Functional Logic of the Massive Number of Feedback Loops in the Antennal Lobe

With models of OSN and PN cell types, LN feedback motifs, single and pairwise interconnected glomeruli, a methodology to address the composibility of feedback circuits has emerged that can be generalized to the entire AL. In particular, the morphological LN types described in Section 2.3 provide a blueprint for connecting multiple glomeruli via the LNs. **Figure 11A** depicts one such LN that innervates more than 20 glomeruli. In **Supplementary Figure S1** in **Supplementary Material**, we list all these morphological LN types identified in the Hemibrain dataset, the number of neurons of each type, and an example neuron-type morphology. The morphology of the neurons is colored by their glomerular arborization.

To model the entire AL feedback circuit, we define glomeruli as parallel channels (Scott and Dahanukar, 2014) each exposing 4 ports that were depicted in **Figures 7E**, **8**. The ports and the LNs form a crossbar as depicted in **Figure 11B**, where the crossbar "intersections" are marked with black dots. The connections of each LN with the ports of glomeruli are determined by the matrix listed on the right of **Supplementary Figure S1**. LNs also form a second crossbar associated with each glomerulus, depicted in **Figure 11B**, where the crossbar "intersections" are marked with gray dots. The exact connectivity pattern can be constructed through compositions.

To formally address composability of the circuit models of the programmable ontology, we introduce here a circuit library, called FeedbackCircuits, for exploring the functional logic of the massive number of feedback loops (motifs) in the fruit fly brain. While the library is generic and can applied to any local processing unit of the fly brain, we highlight here its capabilities in constructing and exploring the AL feedback circuit models described in Section 3.

First, the FeedbackCircuits Library provides tools for interactively visualizing and exploring the feedback loops in the AL circuit operational on the FlyBrainLab computing platform (Lazar et al., 2021) (see Section 6 for details).

Second, the FeedbackCircuits Library enables users to instantiate an executable circuit of the feedback circuit model in two ways. An executable circuit can be instantiated according to a connectome dataset. For example, any circuit explored via NeuroNLP++ can be loaded into an executable circuit directly. It can also be instantiated according to the abstraction of feedback motif examples provided in Section 3.3 (see also Section 6).

While the connectivity pattern of neurons can be extracted from connectome datasets, users can also define higher level objects, such as glomeruli of the AL (see also Section 6). Within a chosen object, the characteristics of the executable models, such as the dynamics of neurons of different cell types, are specified by the user. For example, users can specify all OSN to PN connections to execute commonly-used models of synaptic dynamics. Every instance of such synapses, residing in the connectome dataset, will be automatically assigned the specified dynamics. Similarly, all LN to OSN connections can be specified to act presynaptically on OSN axon terminals (Lazar et al., 2020).

Finally, different LN feedback motifs can be flexibly configured, stimulated or ablated in the FeedbackCircuit Library and their individual and combined effect on the AL outputs can be evaluated.

The FeedbackCircuits Library provides easy-to-customize loader and visualization functions to explore the I/O behavior of the antennal lobe circuit. This process can be repurposed for a wide variety of neuropils, including the mushroom body and the lateral horn of the early olfactory system.

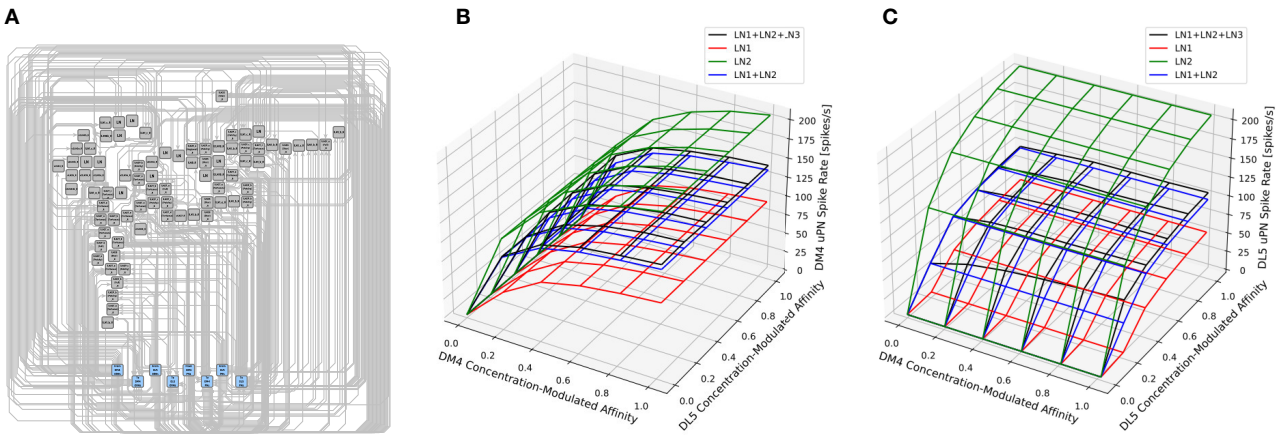


FIGURE 10 | Characterization of DM4 PN and DL5 PN responses of a circuit consisting of interconnected DM4 and DL5 glomeruli. **(A)** A circuit diagram of the interconnected DM4 and DL5 glomeruli. Toward the bottom, the blue nodes represent ports to and from DM4/DL5 OSNs/PNs. Each gray node corresponds to a specific LN in the Hemibrain dataset. Only connectivity between the gray nodes and the blue nodes are shown in the diagram. **(B)** DM4 PN steady-state firing rate and **(C)** DL5 PN steady-state firing rate as glomeruli are subject to different concentration modulated affinity values. For both **(B,C)**: (red) Only LN1 feedback motifs are present. (green) Only LN2 feedback motifs are present. (blue) LN1 and LN2 feedback motifs are present. (black) LN1, LN2 and LN3 feedback motifs are present.

5. DISCUSSION

5.1. A Programmable Ontology Encompassing the Functional Logic of the Brain

Existing neuroscience-centered ontologies, including those of the fruit fly (Costa et al., 2013), the rat/mouse (Paxions and Watson, 2013; Swanson, 2018; Wang et al., 2020) and the human (Sunkin et al., 2012) brain, mainly focus on neuroanatomical structures, hierarchies and nomenclature. Description of entities or relations that have functional significance is rare and is kept at behavioral or cognitive level (Poldrack and Yarkoni, 2016).

While describing the structure of the brain is certainly a first step in the quest of understanding brain function, it is far from being sufficient. Thus, an ontology of the brain can not end with the description of anatomical data. Rather, the anatomical entities and relations have to be augmented with insights characterizing the functional logic of brain circuits.

In this paper, we presented a programmable ontology that expands the scope of the current ontology of *Drosophila* brain anatomy (Costa et al., 2013; Lazar et al., 2021) to encompass the functional logic of the fly brain. The programmable ontology provides a language not only for modeling circuit motifs but also for programmatically exploring their functional logic. To achieve this goal, we tightly integrated the programmable ontology with the workflow of the interactive FlyBrainLab computing platform (Lazar et al., 2021). In effect, the programmable ontology, embedded into the FlyBrainLab, has grown into a programming environment operating with access to a plethora of datasets, containing models of sensory space, the connectome/synaptome including cell types/feedback loops and neuronal/synaptic dynamics. The programmable ontology has the "built in" capability for evaluating the functional logic of brain circuits and for comparing their behavior with the biological counterparts.

To provide a language for defining functional circuit motifs anchored onto biological datasets and the worldwide literature, we developed the NeuroNLP++ web application that supports free-form English query searches of ontological entities and references to these in the published literature worldwide. NeuroNLP++ enables circuits to be composed using connectomic/synaptomic data in support of the evaluation of their function *in silico*. To bridge the gap between the existing *Drosophila* Anatomy Ontology dataset and the Hemibrain connectome morphology dataset, we associated with each ontological entity the corresponding neurons in the morphology dataset. The DrosoBOT Engine, in conjunction with the rule-based NLP engine, represents a first step toward providing a unified and integrated view of connectomic/synaptomic datasets and of the fruit fly brain literature worldwide.

In our programmable ontology the modeling of the space of sensory stimuli is explicitly included. We note that, e.g., the space of odorants has not been discussed in formal ontologies of the fly brain anatomy, although it plays a key role in defining, characterizing, and evaluating the functional logic of brain circuits. Here, the odorant space is modeled by a 3D tensor trio that describes the interaction between odorants and olfactory receptors, rather than by the (largely intractable) detailed/precise chemical structure of the odorants. Defining odorants and odorant mixtures as well as their interactions with olfactory receptors is an important step of this program.

By augmenting the ontology with the space of odorant objects and by providing an English query web pipeline for exploring structural features of the architecture of the early olfactory brain circuits, we are now in a position to evaluate the functional logic of these circuits in their full generality. The program of research presented here, due to space limitations, only sets the stage to modeling and exploratory computational evaluation of the early olfactory system. Clearly, an extension to the other sensory

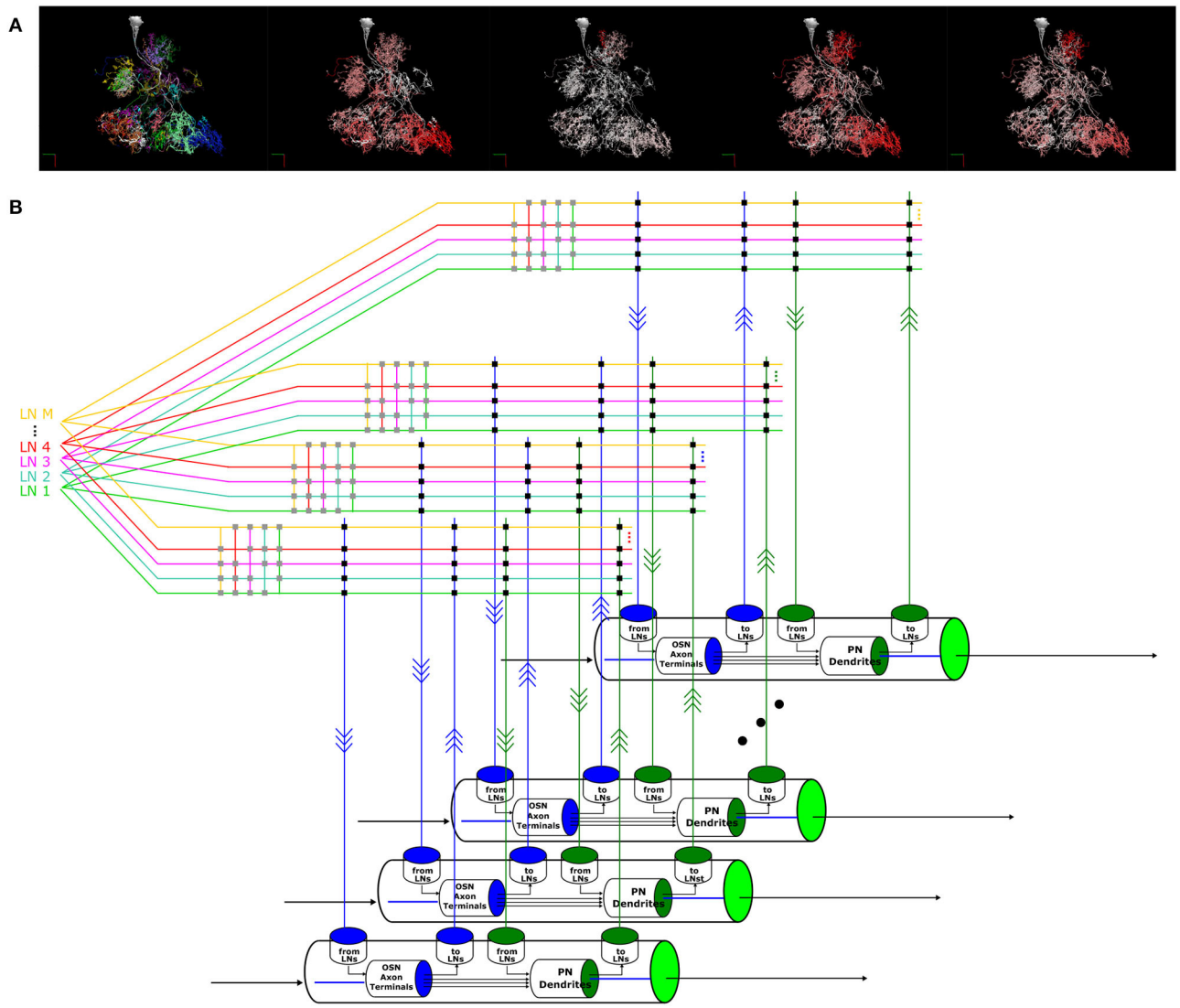


FIGURE 11 | A circuit diagram modeling the entire AL circuit. **(A)** Morphology of an LN, its innervation of glomeruli and its connectivity to the glomeruli it innervates. The neurites in each glomerulus are colored differently, and from left to right: 1) Each color indicates the glomerulus the LN arborizes. 2) The arbors of the LN in each glomerulus are colored in red with saturation proportional to the number of synapses with the OSNs with axons arborizing in that glomerulus, i.e., redder indicates higher number of synapses. 3) The arbors of the LN in each glomerulus are colored in red with saturation proportional to the number of synapses onto the OSNs with axons arborizing in that glomerulus. 4) The arbors of the LN in each glomerulus are colored in red with saturation proportional to the number of synapses with the PNs in that glomerulus. 5) The arbors of the LN in each glomerulus is colored in red with saturation proportional to the number of synapses onto the uPNs in that glomerulus. **(B)** Schematic diagram of the glomeruli and the massive number of feedback circuits in the AL.

modalities is in order. In particular, the early vision system (Lazar et al., 2015) and the central complex (Givon et al., 2017; Lazar et al., 2021) are our next candidates.

5.2. Construction of Circuit Motifs With the FeedbackCircuits Library

Detailed connectomic datasets, such as the Hemibrain dataset, reveal a massive number of nested feedback loops among different cell types. Dissecting the role of these feedback circuits is key to the understanding the model of computation underlying the Local Processing Units (LPUs) of the fruit fly brain. The methodology underlying the FeedbackCircuits Library we advanced here has wide reaching implications for studying the massive feedback loops that dominate all regions/neuropils of the fruit fly brain.

The FeedbackCircuits Library brings together the available *Drosophila* connectomic, synaptomic and cell type data, with tools for 1) querying connectome datasets that automatically find and incorporate feedback pathways, 2) generating interactive circuit diagrams of the feedback circuits, 3) automatic derivation of executable models based on the composition of feedback motifs anchored on actual connectomic data, 4) arbitrary

manipulation (and/or ablation) of feedback circuits featured by an executable interactive circuit diagram, and 5) systematic characterization and comparison of the effect of different feedback loops on the I/O behavior of arbitrary brain circuits.

We have demonstrated the capabilities of the FeedbackCircuits Library using circuits of the DM4 and DL5 glomeruli of the *Drosophila* antennal lobe constructed, based on the Hemibrain dataset, either in isolation or pairwise interconnected. We have also demonstrated a methodology for constructing, exploring and characterizing the contribution of individual feedback motifs as well as their compositions. The entire AL feedback circuit can be readily constructed with the approach we outlined.

The work presented here represents the beginning of an in-depth study of feedback motifs and their functional logic. By outlining the programmable ontology and demonstrating its workflow in exploring the functional logic of brain circuit from fly brain data, we advanced an accelerated path for the exploration and discovery of the functional logic of the fruit fly brain.

Studies of the functional logic of sensory processing neuropils such as the medulla and the mushroom body are currently limited to the feedforward pathways (Yang and Clandinin, 2018; Borst et al., 2020; Modi et al., 2020), although these circuits exhibit strong feedback components (Nern et al., 2015; Eschbach et al., 2020; Lazar et al., 2021). Our methodology paves the way for deeper investigations into the composition of such feedback circuits.

6. MATERIALS AND METHODS

In this section, we present the details of the methodology for the exploration of the morphology of massive feedback circuits, modeling odor signal processing in the early olfactory system, and the interactive exploration of the antennal lobe as a network of glomeruli.

For creating the tools underlying the programmable ontology we used extensive capabilities to query datasets and build executable circuits, query the antennal lobe circuitry using these as well as customized tools, constructing and evaluating the feedback circuits with the FeedbackCircuits Library, and mapping glomeruli and their compositions into executable circuits.

6.1. Exploring the Morphology of Cell Types and Feedback Circuits

6.1.1. The NeuroNLP++ Web Application

NeuroNLP is a web application that supports the exploration of fruit fly brain datasets with rule-based English queries (Ukani et al., 2019; Lazar et al., 2021). To enhance the user experience when asking questions that are well beyond the current capabilities of NeuroNLP, we devised the NeuronNLP++ brain explorer (https://plusplus.neuronlp.fruitflybrain.org). **Figure 12** depicts the software architecture of the NeuroNLP++ application. In addition to the backend servers supporting the NeuroNLP web application (NeuroArch Server and NeuroNLP Server with rule-based NLP Engine Ukani et al., 2019; Lazar et al., 2021), NeuroNLP++ is supported by the DrosoBOT Engine (see below), i.e., an additional backend of the NeuroNLP Server.

A free-form English query submitted through NeuroNLP++ that cannot be interpreted by the rule-based NLP engine is redirected to DrosoBOT. DrosoBOT responds with a list of ontological entities that are most pertinent. Each entry in the list includes the name of the cell type, a link to the *Drosophila* Anatomy Ontology containing references to the entities in question (Costa et al., 2013), and a description of the cell type as well as relevant entries to the worldwide literature (see also the NeuroNLP++ window in **Figure 3A**). Each entry also includes buttons to add, pin and unpin the neurons in the 3D visualization workspace. The morphology of the neurons is retrieved from the NeuroArch Database (Givon et al., 2015) via the NeuroArch Server and displayed using custom visualization in the browser (Ukani et al., 2019; Lazar et al., 2021) (see **Figure 12**).

In addition to using DrosoBOT to resolve free-form English queries, NeuroNLP++ includes an application called Graph View that allows users to visualize the graph representing the connectivity of neurons in their workspace at neuronal or cell type level. Once the Graph View button is pressed, NeuroNLP++ retrieves the connectivity of all the neurons in the workspace, with an additional capability to filter out the connections that have less than N synapses, where $N \geq 0$. A graph is then plotted in the workspace using the sigma.js library.

For neuronal level graph, each node represents a single neuron in the visualization workspace, and the edge between two nodes indicates a positive number of synaptic connections between the corresponding neurons. For a cell type level graph, each node represents a cell type that may abstract multiple neurons in the visualization workspace. An edge indicates that there exists at least 1 synaptic connection between neurons in the two cell types. In addition, the graph in Graph View is interactive. Hovering the mouse on a node highlights the corresponding neuron or all neurons of the same cell type in the 3D visualization. The graph can be further individually rearranged.

6.1.2. The DrosoBOT Engine

DrosoBOT is a natural language processing engine that 1) parses free-form English queries pertaining to entities available in an ontological dataset (Costa et al., 2013), and 2) provides morphological data from a connectome dataset (Scheffer et al., 2020) already associated with each ontological entity.

Given a free-form English query, DrosoBOT first uses DPR (see below) to retrieve relevant passages in the query as context candidates, and then uses PubMedBERT, fine-tuned on the Stanford Question Answering Dataset, to find possible answers to questions pertaining to a collection of *Drosophila*-specific ontology terms and their descriptions. Here DPR is the dense passage retriever trained on the Natural Questions dataset (Kwiatkowski et al., 2019) that uses real anonymized queries issued to Google and annotated answers from the top 5 Wikipedia articles. PubMedBERT is the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018)

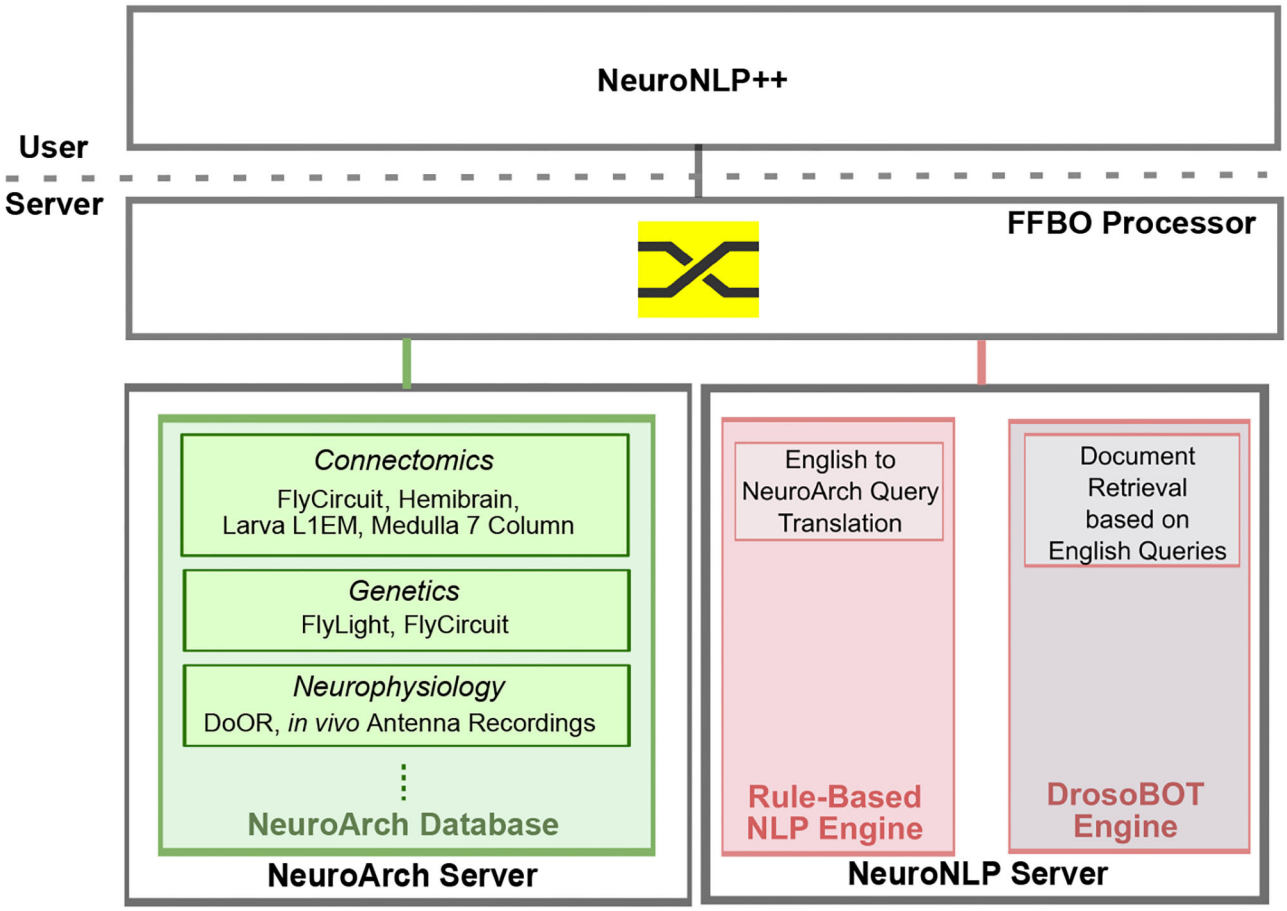


FIGURE 12 | Software architecture of the NeuroNLP++ web application. The frontend application communicates with three backend servers: a NeuroArch Server that hosts a NeuroArch Database of fruit fly brain datasets, a NeuroNLP Server that runs the rule-based NLP Engine, and, an additional NeuroNLP Server that runs the DrosoBOT Engine.

model with biomedical domain-specific pre-training (Gu et al., 2021) from abstracts on PubMed.

In addition, for specific cell types, DrosoBOT implements a modular lexical search subsystem that uses domain knowledge to improve search results when specific keywords of cell types are asked. We make use of this system to improve the search results for the Antennal Lobe, which requires biological nomenclatures such as "DM4" to be detected not as typos but as important structures.

To bridge the gap between existing ontology and connectome datasets, we associated with each ontological entity the corresponding neurons in the Hemibrain dataset based on the names of the entities and their synonyms after searching through all possible matches in the *Drosophila* Anatomy Ontology (DAO) dataset (Costa et al., 2013). We then created a graph with nodes consisting of both names of entities in the DAO and names of neurons. An edge is created between two nodes with a matching term. After finding the ontological term relevant to the English query from the first step, we then retrieved the names of the neurons that are the graph neighbors of the ontological entity, and finally retrieved the neurons from the database.

For the AL, starting with the terms for cell types and abstractions in Costa et al. (2013) and expanding these to include references to all cell types so that all common synonyms are accounted for (for example, PNs, OSNs, glomeruli and LNs), we facilitated the specification of Antennal Lobe circuits through queries. Here we provided the capability to add relevant groups of neurons such as new glomeruli and local neurons in only a few searches and button presses. We also added the names of all glomeruli as special "keywords" whose association with the Antennal Lobe is automatically detected if present in a search query.

6.1.3. Morphology and Graph Abstractions of Cell Types in the Antennal Lobe

The morphology of OSNs, PNs and LNs is retrieved, as shown in **Figure 4**, by using NeuroNLP++. Graph abstractions of OSNs and PNs are obtained by invoking the cell type level algorithm of Graph View.

The full list of morphological LN-types in the Antennal Lobe is presented in **Supplementary Figure S1** in **Supplementary Material**. Each row depicts an instance

of LN-type, the name of the LN-type as defined in the Hemibrain dataset, and the number of instances of LN-types. The connectivity of the shown instances of LNs with OSN and PNs arborizing in each of the 51 olfactory glomeruli is represented as a matrix on the right. The matrix entries are the number of synaptic connections of the LNs from/to OSNs and PNs. Here, we employed custom code to retrieve synaptome information directly from the NeuroArch Database.

6.1.4. Morphology and Graph Abstractions of Feedback Circuits in the Antennal Lobe

To identify feedback loops in the AL, we extracted from the Hemibrain dataset (stored in the NeuroArch Database) the number of synapses between each LN-type and each type of OSNs and PNs. Here, we only considered a synapse if both its presynaptic and postsynaptic sites are identified at a confidence level higher than 70%. An LN forms an OSN-LN-OSN feedback loop if it receives a total of more than 5 synapses from all the OSNs and provides a total of more than 5 synapses to all the OSNs, and it has less than 5 synapses with all the PNs. Similarly, we consider that an LN forms a PN-LN-PN feedback loop if it receives from and provides to all the PNs a total of more than 5 synapses and has less than 5 synapses with all the OSNs. The LNs identified with each type of feedback loop in a glomerulus are then associated with an ontological entity that is accessible by DrosoBOT for document retrieval.

6.2. Creating the Programmable Ontology of the *Drosophila* Brain

We tightly integrated the programmable ontology of the fruit fly brain into the workflow of the interactive FlyBrainLab computing platform depicted in **Figure 2** (Lazar et al., 2021).

To improve the support of the visualization of the fly brain morphology, we integrated using NeuroNLP++ the *Drosophila* anatomical ontology with the connectome/synaptome data of the Hemibrain dataset. In the FlyBrainLab workflow shown in **Figure 2** this integration factors into the first step of the left.

The programmable ontology provides a methodology to define abstractions of fly brain circuits and create executable circuit diagrams as described in the second step of the FlyBrainLab workflow shown in **Figure 2**. FlyBrainLab provides the support for the visualization and user interaction with executable circuit diagrams.

Finally, programmability of the ontology is supported by the FlyBrainLab in the third step of the workflow to configure, compose and execute neural circuit models and to evaluate their functional logic.

6.2.1. Receptor-Centric Modeling of the Space of Odorants

To construct odorant objects as elements of the space of odorants, we employed the receptor-centric Odorant Transduction Process (OTP) model developed in Lazar and Yeh (2020). In steady-state, the estimated affinity tensor $\mathbf{b}/\mathbf{d}$ with entries $[\mathbf{b}]_{ron}/[\mathbf{d}]_{ron}$, for all $r = 1, 2, ..., R$, $o = 1, 2, ..., O$ and $n = 1, 2, ..., N$, matches the spike rate response of the OTP model with the spike rate of the neurophysiology recordings (Hallem and Carlson, 2006) obtained in response to a constant amplitude waveform of 110 different odorants. Detailed data can be found in the olfactory transduction circuit library OlfTransCircuit available at https://github.com/FlyBrainLab/OlfTrans (Lazar et al., 2021).

6.2.2. Modeling/Constructing Individual Glomeruli in the Antennal Lobe Circuit

As a first step in modeling an individual glomerulus, we first extracted all OSNs with axons arborizing a glomerulus and all the PNs that innervate that glomerulus.

To abstract the connectivity patterns of the LNs leading to the circuit diagram in **Figure 7**, we inspected all 311 LNs in the Hemibrain dataset (Scheffer et al., 2020). Of these, 226 LNs have more than 10 synapses in the right hemisphere AL. We only considered a synapse if both its presynaptic and postsynaptic sites are identified at a confidence level higher than 70% in the Hemibrain dataset. For each of these LNs, we counted the number of synapses they make, presynaptically and postsynaptically, with partner OSNs as well as PNs in each glomerulus. If the total number of synapses within a glomerulus is less than 5, we deem the port connectivity pattern to be 0000, i.e., no connections. The first digit of the 4-digit binary code is 1 if the number of LN to OSNs synapses is larger than 5. Similarly, the second digit is 1 if the number of synapses the LN receives from OSNs is larger than 5. The third digit is 1 if the number of LN to PNs synapses is larger than 5. Similarly, the fourth digit is 1 if the number of synapses the LN receives from PNs is larger than 5.

After inspecting all 226 LNs that innervate the right AL in the hemibrain dataset (Scheffer et al., 2020), we listed in the 2nd column of **Supplementary Table S1** in **Supplementary Material** the number of instances each port connectivity pattern occurs across 51 olfactory glomeruli. For the DM4 and DL5 glomeruli, the number of occurrences of each port connectivity pattern is listed in the 3rd and 4th column, respectively.

6.2.3. Modeling/Constructing a Pair of Interconnected Glomeruli in the AL

The modeling of interconnected glomeruli starts with the initialization of isolated glomeruli as described above. LN1s and LN2s feedback motifs can then be easily composed across glomeruli. Composition of the resulting feedback loops with LN3 leads to an interconnect of two glomeruli. These are very simple composition rules that open new directions in exploring the functional logic of interconnected glomeruli.

6.3. Exploring the Functional Logic of the Feedback Circuits in the Antennal Lobe

6.3.1. Interactively Exploring Circuit Diagrams With the FeedbackCircuits Library

The FeedbackCircuits Library mentioned earlier in Section 4.3 was developed in Python and designed to be integrated into the FlyBrainLab ecosystem for constructing feedback circuits and exploring their function.

The FeedbackCircuits Library provides tools for interactively visualizing and exploring the feedback loops of the AL model circuits that are operational on the FlyBrainLab computing

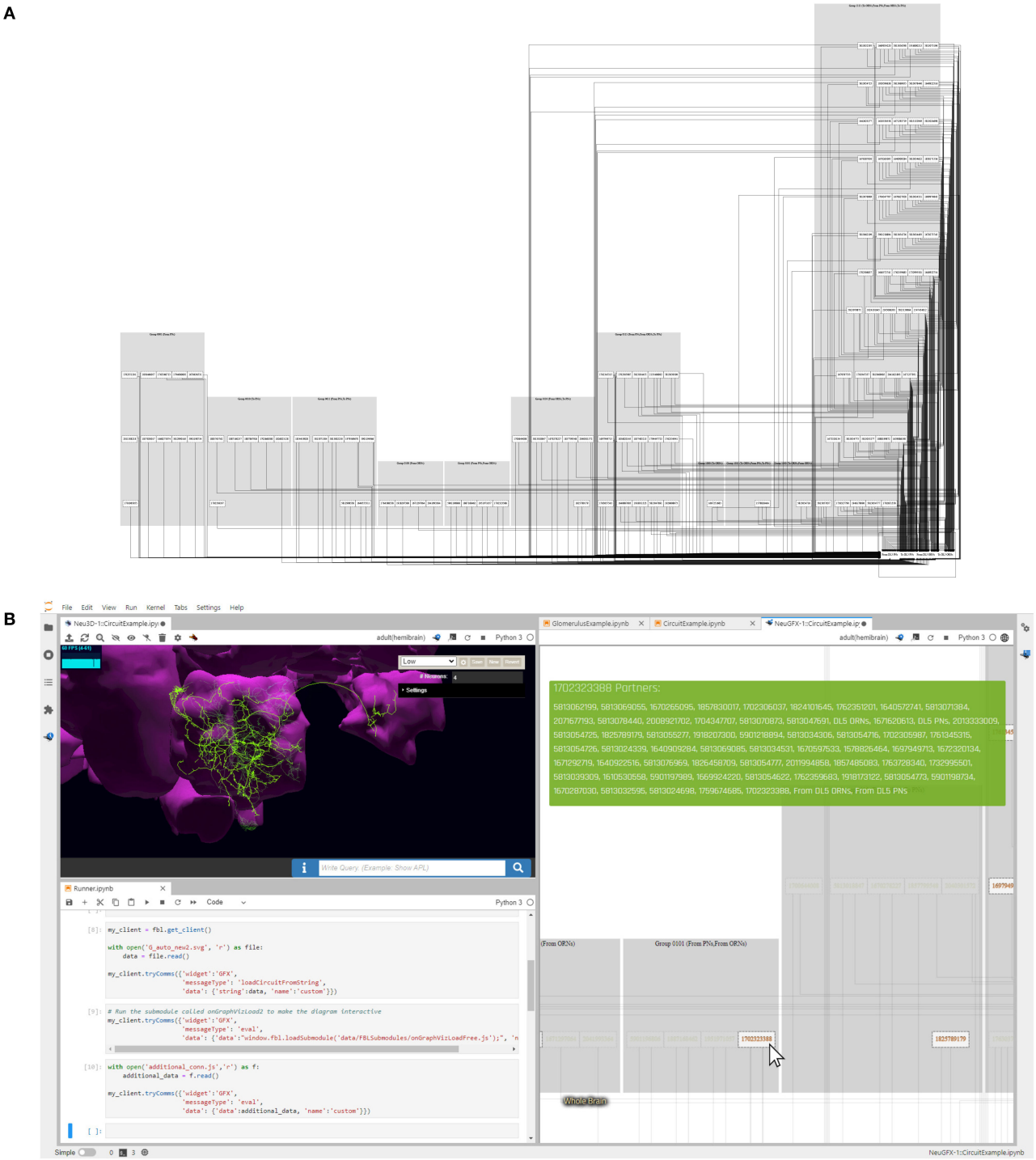


FIGURE 13 | Exploring the feedback loops between the DL5 and DM4 glomeruli using an interactive circuit diagram generated by the FeedbackCircuits Library. **(A)** Users can generate a circuit diagram for any glomerulus consisting of OSNs, PNs, and LNs grouped according to the feedback connectivity pattern described in **Figure 7E**, such as the diagram here for the DL5 glomerulus. **(B)** The generated diagrams are interactive in the FlyBrainLab platform. Hovering over the neurons on the diagram (right) shows their partners, and highlights them in the diagram and in the corresponding 3D morphology (top left); clicking disables/enables them for program execution (bottom left). Here, the user is currently hovering over the Hemibrain neuron with identifier 1702323388, which shows its partners in the green block on the top right window and highlights it on the morphology in the top left in green.

platform (Lazar et al., 2021). **Figure 13A** depicts an automatically generated circuit diagram of the DL5 glomerulus. This circuit diagram is a schematic of the glomerulus model shown in **Figure 7E** and is based, for OSNs and PNs, upon the Hemibrain connectome dataset (Scheffer et al., 2020). LNs are grouped into blocks according to their port connectivity patterns with respect to a given glomerulus (here, DL5). The circuit diagram is fully operational in the FlyBrainLab platform; the interactive user interface is shown in **Figure 13B**. On the top left is the NeuroNLP 3D visualization window for displaying the morphology of neurons. At the bottom left a Jupyter notebook for code execution is displayed. The circuit diagram allows users to inspect the morphology of neurons in the NeuroNLP window by clicking on the ones displayed. For example, in **Figure 13B** (right), we zoomed into the LNs that exhibit the port connectivity pattern 0101. By clicking on the neuron whose Hemibrain ID is 1702323388, the LN in green is highlighted in **Figure 13B** (top left). Hovering also highlights all connected neurons, such as the one with Hemibrain ID 1825789179 in **Figure 13B** (right). The interactive circuit diagram provides an intuitive means of constructing feedback motifs from connectome data.

6.3.2. Evaluating the Role of Feedback Circuits in a Single Glomerulus

As already briefly mentioned above, to construct the DM4 glomerulus feedback circuits with the FeedbackCircuits Library, we created a DM4 glomerulus object that includes all the OSNs and PNs and their connections according to the Hemibrain dataset. We then add feedback motifs LN1, LN2, and LN3 from **Figure 8C**. The construction of the DL5 glomerulus including the feedback motifs follows a similar procedure.

To model the odorant transduction process of the OSNs, we followed the OTP model in Lazar and Yeh (2020). The axon hillock of OSNs, PNs and LNs is given by the Connor- Stevens point neuron model (Connor and Stevens, 1971). All synapses are modeled as a variation of the α synapse. We also modeled the presynaptic effect of LNs onto the OSN axon terminal. A detailed description of the dynamics of the olfactory transduction process, neurons and synapses can be found in Section 2 of **Supplementary Material**. The configured model circuits were executed on the Neurokernel Execution Engine (Givon and Lazar, 2016). Neurokernel supports the execution of spiking and/or analog neuron models.

To evaluate the feedback motifs, we swept through all possible concentration-modulated affinity values, defined as $[\mathbf{b}]_{ron}/[\mathbf{d}]_{ron} \cdot [\mathbf{u}]_o$, where $[\mathbf{b}]_{ron}/[\mathbf{d}]_{ron}$ is the affinity odorant o with the receptor r expressed by OSN n, and $[\mathbf{u}]_o$ is the constant concentration waveform of odorant o presented to OSN n.

6.3.3. Evaluating the Role of Feedback Circuits in/Between a Pair of Glomeruli

To construct the feedback circuit interconnecting a pair of glomeruli (e.g., DM4 and DL5), we started with two independent circuits, with feedback motifs LN1 and LN2, that only innervate a single glomerulus. We then composed these two independent circuits across the two glomeruli, and added a feedback motif LN3 that connects to each LN1 and LN2 in both directions. Instead of stimulating LN3 externally, we assumed that synapses from LN1 and LN2 to LN3 are excitatory, and the output of LN3 is inhibitory.

The olfactory transduction, axon hillock and synaptic models of the interconnected pair of glomeruli are the same as the ones of the single glomerulus above, and their dynamics are described in detail in Section 2 of **Supplementary Material**.

To evaluate the feedback circuit of the pair of interconnected glomeruli, we swept through constant inputs on a grid of concentration-modulated affinity values associated with the odorant receptor of OSNs with axons that arborize the DM4 and DL5 glomeruli, respectively. PN responses to the inputs with values on lines crossing the origin can be used to characterize the responses to the odorants of interest.

6.3.4. Modeling and Constructing the Massive Feedback Circuits of the AL

Composition of the circuit diagram of the entire AL in **Figure 11** comes in three steps. First, OSNs and PNs and their ports from/to LNs are constructed for each glomerulus according to **Figure 7**, where the glomeruli are depicted as cylinders at the bottom of **Figure 11B**. Second, for each glomerulus, we also configure the connectivity patterns of the LNs, as described in Section 4.3. This forms the local crossbar between all LNs and the ports of a glomerulus and is depicted, e.g., on the top right of **Figure 11B**, as 4 vertical lines. Finally, by connecting the local crossbars from all glomeruli with the innervation pattern of each LN, we obtain a hierarchical crossbar between LNs and the ports of the glomeruli. The hierarchical crossbar provides the flexibility to configure the routing of interconnections across glomeruli, either by using the port connectivity patterns of LNs extracted from connectome data (see also **Supplementary Figure S1** in **Supplementary Material**), or by any variations/ablations thereof for testing and evaluating the functional logic of the AL circuit.

CODE AVAILABILITY STATEMENT

NeuroNLP++ web application is available at https://plusplus.neuronlp.fruitflybrain.org. It is also available as a Docker machine image for standalone installations. The FeedbackCircuits Library is available as a Python package at https://github.com/mkturkcan/FeedbackCircuits. The FeedbackCircuits Library includes a number of example Jupyter notebooks that help users explore its functionality. For installation of FlyBrainLab, refer to Lazar et al. (2021).

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: https://www.fruitflybrain.org. The NeuroArch Database hosting publicly available Hemibrain dataset that are used in the exploration and validation in this paper can be downloaded from https://github.com/FlyBrainLab/datasets.

AUTHOR CONTRIBUTIONS

AL contributed to conceptualization, developing research directions, investigation, methodology, writing—original draft, writing—review and editing, resources, supervision, funding acquisition, and project administration. MT contributed to conceptualization, software, validation, investigation, visualization, methodology, writing—original draft, writing—review and editing. Developed NeuroNLP++ and the FeedbackCircuits library. Performed evaluation of the AL feedback circuit. YZ contributed to conceptualization, developing research directions, validation, investigation, visualization, methodology, writing—original draft, writing—review and editing, funding acquisition, and project administration. All authors contributed to the article and approved the submitted version.

FUNDING

The research reported here was supported by AFOSR under grant #FA9550-16-1-0410, DARPA under contract #HR0011-19-9-0035 and NSF under grant #2024607.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments to improve the manuscript.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.853098/full#supplementary-material

REFERENCES

Bakken, T. E., Jorstad, N. L., Hu, Q., Lake, B. B., Tian, W., Kalmbach, B. E., et al. (2021). Comparative cellular analysis of motor cortex in human, marmoset and mouse. *Nature* 598, 111–119. doi: 10.1038/s41586-021-03465-8

Borst, A., Haag, J., and Mauss, A. S. (2020). How fly neurons compute the direction of visual motion. *J. Compar. Physiol. A* 206, 109–124. doi: 10.1007/s00359-019-01375-9

Chiang, A.-S., Lin, C.-Y., Chuang, C.-C., Chang, H.-M., Hsieh, C.-H., Yeh, C.-W., et al. (2011). Three-dimensional reconstruction of brain-wide wiring networks in *Drosophila* at single-cell resolution. *Curr. Biol.* 21, 1–11. doi: 10.1016/j.cub.2010.11.056

Chou, Y.-H., Spletter, M. L., Yaksi, E., Leong, J. C. S., Wilson, R. I., and Luo, L. (2010). Diversity and wiring variability of olfactory local interneurons in the *Drosophila* antennal lobe. *Nat. Neurosci.* 13, 439–449. doi: 10.1038/nn.2489

Clements, J., Dolafi, T., Umayam, L., Neubarth, N. L., Berg, S., Scheffer, L. K., et al. (2020). neuprint: analysis tools for em connectomics. *bioRxiv*. doi: 10.1101/2020.01.16.909465

Connor, J. A., and Stevens, C. F. (1971). Prediction of repetitive firing behavior from voltage clamp data on an isolated neurone soma. *J. Physiol.* 213, 31–53. doi: 10.1113/jphysiol.1971.sp009366

Costa, M., Reeve, S., Grumbling, G., and Osumi-Sutherland, D. (2013). The *Drosophila* anatomy ontology. J. Biomed. Semantics 4, 32. doi: 10.1186/2041-1480-4-32

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint* arXiv:1810.04805. doi: 10.48550/arXiv.1810.04805

Egelhaaf, M., Kern, R., Krapp, H. G., Kretzberg, J., Kurtz, R., and Warzecha, A.-K. (2002). Neural encoding of behaviorally relevant visual-motion information in the fly. *Trends Neurosci.* 25, 96–102. doi: 10.1016/S0166-2236(02)02063-5

Eschbach, C., Fushiki, A., Winding, M., Schneider-Mizell, C. M., Shao, M., Arruda, R., et al. (2020). Recurrent architecture for adaptive regulation of learning in the insect brain. *Nat. Neurosci.* 23, 544–555. doi: 10.1038/s41593-020-0607-9

Givon, L. E., and Lazar, A. A. (2016). Neurokernel: an open source platform for emulating the fruit fly brain. *PLoS ONE* 11, e0146581. doi: 10.1371/journal.pone.0146581

Givon, L. E., Lazar, A. A., and Ukani, N. H. (2015). Neuroarch: A graph db for querying and executing fruit fly brain circuits. *Zenodo*. doi: 10.5281/zenodo.44225

Givon, L. E., Lazar, A. A., and Yeh, C.-H. (2017). Generating executable models of the *Drosophila* central complex. *Front. Behav. Neurosci.* 11, 102. doi: 10.3389/fnbeh.2017.00102

Grünert, U., and Martin, P. R. (2020). Cell types and cell circuits in human and non-human primate retina. *Prog. Retin Eye Res.* 78, 100844. doi: 10.1016/j.preteyeres.2020.100844

Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., et al. (2021). Domain-specific language model pretraining for biomedical natural language processing. *ACM Trans. Comput. Healthcare* 3, 1–23. doi: 10.1145/3458754

Hallem, E. A., and Carlson, J. R. (2006). Coding of odors by a receptor repertoire. *Cell* 125, 143–160. doi: 10.1016/j.cell.2006.01.050

Harris, J. A., Mihalas, S., Hirokawa, K. E., Whitesell, J. D., Choi, H., Bernard, A., et al. (2019). Hierarchical organization of cortical and thalamic connectivity. *Nature* 575, 195–202. doi: 10.1038/s41586-019-1716-z

Huang, Y.-C., Wang, C.-T., Su, T.-S., Kao, K.-W., Lin, Y.-J., Chuang, C.-C., et al. (2019). A single-cell level and connectome-derived computational model of the *Drosophila* brain. *Front. Neuroinform.* 12, 99. doi: 10.3389/fninf.2018.00099

Hulse, B. K., Haberkern, H., Franconville, R., Turner-Evans, D. B., Takemura, S.-,y., Wolff, T., et al. (2021). A connectome of the *Drosophila* central complex reveals network motifs suitable for flexible navigation and context-dependent action selection. *Elife* 10, e66039. doi: 10.7554/eLife.66039

Jeanne, J. M., Fişek, M., and Wilson, R. I. (2018). The organization of projections from olfactory glomeruli onto higher-order neurons. *Neuron* 98, 1198.e6–1213.e6. doi: 10.1016/j.neuron.2018.05.011

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., et al. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint* arXiv:2004.04906. doi: 10.18653/v1/2020.emnlp-main.550

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., et al. (2019). Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguist.* 7, 453–466. doi: 10.1162/tacl_a_00276

Lamme, V. A., Supèr, H., and Spekreijse, H. (1998). Feedforward, horizontal, and feedback processing in the visual cortex. *Curr. Opin. Neurobiol.* 8, 529–535. doi: 10.1016/S0959-4388(98)80042-1

Lazar, A. A., Liu, T., Turkcan, M. K., and Zhou, Y. (2021). Accelerating with flybrainlab the discovery of the functional logic of the *Drosophila* brain in the connectomic and synaptomic era. *Elife* 10, e62362. doi: 10.7554/eLife.62362

Lazar, A. A., Liu, T., and Yeh, C.-H. (2020). "An odorant encoding machine for sampling, reconstruction and robust representation of odorant identity," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Barcelona: IEEE), 1743–1747.

Lazar, A. A., Liu, T., and Yeh, C.-H. (2022). The functional logic of odor information processing in the *Drosophila* antennal lobe. *bioRxiv*. doi: 10.1101/2021.12.27.474306

Lazar, A. A., Ukani, N. H., Psychas, K., and Zhou, Y. (2015). A parallel processing model of the *Drosophila* retina. *Zenodo*. doi: 10.5281/zenodo.30036

Lazar, A. A., and Yeh, C.-H. (2020). A molecular odorant transduction model and the complexity of spatio-temporal encoding in the *Drosophila* antenna. *PLoS Comput. Biol.* 16, e1007751. doi: 10.1371/journal.pcbi.1007751

Li, H., Janssens, J., De Waegeneer, M., Kolluru, S. S., Davie, K., Gardeux, V., et al. (2021). Fly cell atlas: a single-cell transcriptomic atlas of the adult fruit fly. *bioRxiv*. doi: 10.1101/2021.07.04.451050

Modi, M. N., Shuai, Y., and Turner, G. C. (2020). The *Drosophila* mushroom body: from architecture to algorithm in a learning circuit. *Ann. Rev. Neurosci.* 43, 465–484. doi: 10.1146/annurev-neuro-080317-0621333

Morgan, H. L. (1965). The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *J. Chem. Doc.* 5, 107–113. doi: 10.1021/c160017a018

Nern, A., Pfeiffer, B. D., and Rubin, G. M. (2015). Optimized tools for multicolor stochastic labeling reveal diverse stereotyped cell arrangements in the fly visual system. *Proc. Natl. Acad. Sci. U.S.A.* 112, E2967–E2976. doi: 10.1073/pnas.1506763112

Ohyama, T., Schneider-Mizell, C. M., Fetter, R. D., Aleman, J. V., Franconville, R., Rivera-Alba, M., et al. (2015). A multilevel multimodal circuit enhances action selection in *Drosophila*. *Nature* 520, 633–639. doi: 10.1038/nature14297

Paxions, G., and Watson, C. (2013). *The Rat Brain in Stereotaxic Coordinates*. Elsevier Academic Press. Available online at: https://www.elsevier.com/books/the-rat-brain-in-stereotaxic-coordinates/paxinos/978-0-12-391949-6

Poldrack, R. A., and Yarkoni, T. (2016). From brain maps to cognitive ontologies: informatics and the search for mental structure. *Ann. Rev. Psychol.* 67, 587–612. doi: 10.1146/annurev-psych-122414-033729

Scheffer, L. K., Xu, C. S., Januszewski, M., Lu, Z., Takemura, S.-,y., Hayworth, K. J., et al. (2020). A connectome and analysis of the adult *Drosophila* central brain. *Elife* 9, e57443. doi: 10.7554/eLife.57443

Scott, C. A., and Dahanukar, A. (2014). Sensory coding of olfaction and taste. *Behav. Genet. Fly* 2, 49. doi: 10.1017/CBO9780511920585.005

Shapson-Coe, A., Januszewski, M., Berger, D. R., Pope, A., Wu, Y., Blakely, T., et al. (2021). A connectomic study of a petascale fragment of human cerebral cortex. *bioRxiv*. doi: 10.1101/2021.05.29.446289

Sunkin, S. M., Ng, L., Lau, C., Dolbeare, T., Gilbert, T. L., Thompson, C. L., et al. (2012). Allen brain atlas: an integrated spatio-temporal portal for exploring the central nervous system. *Nucleic Acids Res.* 41, D996–D1008. doi: 10.1093/nar/gks1042

Swanson, L. W. (2018). Brain maps 4.0–structure of the rat brain: An open access atlas with global nervous system nomenclature ontology and flatmaps. *J. Compar. Neurol.* 526, 935–943. doi: 10.1002/cne.24381

Takemura, S.-y., Xu, C. S., Lu, Z., Rivlin, P. K., Parag, T., Olbris, D. J., et al. (2015). Synaptic circuits and their variations within different columns in the visual system of *Drosophila*. *Proc. Natl. Acad. Sci. U.S.A.* 112, 13711–13716. doi: 10.1073/pnas.1509820112

Tasic, B., Yao, Z., Graybuck, L. T., Smith, K. A., Nguyen, T. N., Bertagnolli, D., et al. (2018). Shared and distinct transcriptomic cell types across neocortical areas. *Nature* 563, 72–78. doi: 10.1038/s41586-018-0654-5

Tootoonian, S., Coen, P., Kawai, R., and Murthy, M. (2012). Neural representations of courtship song in the *Drosophila* brain. *J. Neurosci.* 32, 787–798. doi: 10.1523/JNEUROSCI.5104-11.2012

Tran, N., Kepple, D., Shuvaev, S., and Koulakov, A. (2019). "Deepnose: Using artificial neural networks to represent the space of odorants," in *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 (PMLR), 6305–6314. Available online at: http://proceedings.mlr.press/v97/tran19b/tran19b.pdf

Tuthill, J. C., and Wilson, R. I. (2016). Mechanosensation and adaptive motor control in insects. *Curr. Biol.* 26, R1022–R1038. doi: 10.1016/j.cub.2016.06.070

Ukani, N. H., Yeh, C.-H., Tomkins, A., Zhou, Y., Florescu, D., Ortiz, C. L., et al. (2019). The fruit fly brain observatory: from structure to function. *bioRxiv*. doi: 10.1101/580290

Wang, Q., Ding, S.-L., Li, Y., Royall, J., Feng, D., Lesnar, P., et al. (2020). The allen mouse brain common coordinate framework: a 3d reference atlas. *Cell* 181, 936.e20–953.e20. doi: 10.1016/j.cell.2020.04.007

Yang, H. H., and Clandinin, T. R. (2018). Elementary motion detection in *Drosophila*: algorithms and mechanisms. *Ann. Rev. Vis. Sci.* 4, 143–163. doi: 10.1146/annurev-vision-091517-034153

Zheng, Z., Lauritzen, J. S., Perlman, E., Robinson, C. G., Nichols, M., Milkie, D., et al. (2018). A complete electron microscopy volume of the brain of adult *Drosophila melanogaster*. *Cell* 174, 730.e22–743.e22. doi: 10.1016/j.cell.2018.06.019

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers | Frontiers in Neuroinformatics

ORIGINAL RESEARCH
published: 13 July 2022
doi: 10.3389/fninf.2022.855765

A Spiking Neural Network Builder for Systematic Data-to-Model Workflow

Carlos Enrique Gutierrez[1]*, *Henrik Skibbe*[2], *Hugo Musset*[1] and *Kenji Doya*[1]

[1] Neural Computation Unit, Okinawa Institute of Science and Technology Graduate University, Okinawa, Japan, [2] Brain Image Analysis Unit, RIKEN Center for Brain Science, Wako, Japan

In building biological neural network models, it is crucial to efficiently convert diverse anatomical and physiological data into parameters of neurons and synapses and to systematically estimate unknown parameters in reference to experimental observations. Web-based tools for systematic model building can improve the transparency and reproducibility of computational models and can facilitate collaborative model building, validation, and evolution. Here, we present a framework to support collaborative data-driven development of spiking neural network (SNN) models based on the Entity-Relationship (ER) data description commonly used in large-scale business software development. We organize all data attributes, including species, brain regions, neuron types, projections, neuron models, and references as tables and relations within a database management system (DBMS) and provide GUI interfaces for data registration and visualization. This allows a robust "business-oriented" data representation that supports collaborative model building and traceability of source information for every detail of a model. We tested this data-to-model framework in cortical and striatal network models by successfully combining data from papers with existing neuron and synapse models and by generating NEST simulation codes for various network sizes. Our framework also helps to check data integrity and consistency and data comparisons across species. The framework enables the modeling of any region of the brain and is being deployed to support the integration of anatomical and physiological datasets from the brain/MINDS project for systematic SNN modeling of the marmoset brain.

Keywords: spiking neural networks, computational brain modeling, neural simulation, web application, data-to-model workflow, collective intelligence

OPEN ACCESS

Edited by:
Padraig Gleeson,
University College London,
United Kingdom

Reviewed by:
Ján Antolík,
Charles University, Czechia
David Dahmen,
Helmholtz Association of German Research Centres (HZ), Germany

***Correspondence:**
Carlos Enrique Gutierrez
carlos.gutierrez@oist.jp

Received: *15 January 2022*
Accepted: *30 May 2022*
Published: *13 July 2022*

Citation:
Gutierrez CE, Skibbe H, Musset H and Doya K (2022) A Spiking Neural Network Builder for Systematic Data-to-Model Workflow. Front. Neuroinform. 16:855765. doi: 10.3389/fninf.2022.855765

1. INTRODUCTION

Large amounts of diverse brain data are being generated from multiple brain science projects around the world (Markram et al., 2011; Okano et al., 2016; Abbott, 2021). However, to understand the functions of the brain, it is necessary to integrate such diverse experimental data as neural network models and to analyze dynamics and information transfer through systematic simulations. As an approach to effectively utilize experimental data, projects are promoting development of tools for brain modeling, such as the virtual brain (Sanz Leon et al., 2013), NetPyNE (Dura-Bernal et al., 2019), the Brain modeling toolKit (Dai et al., 2020), NEST Desktop (Spreizer et al., 2021), or PhysioDesigner (Asai et al., 2012). Besides that, a range of tools has been proposed as well for facilitating model description and supporting workflow-related processes, such as NeuroML (Gleeson et al., 2010), Mozaik (http://neuralensemble.org/docs/mozaik), SNNtoolbox (Rueckauer et al., 2017), Nengo (Bekolay et al., 2014), pypet (Meyer and Obermayer, 2016), NeuroManager (Stockton and Santamaria, 2015), and others.

In building realistic brain models, it is necessary to systematically incorporate experimental data, published data in the literature, parameters from prior models, and theoretical or mechanistic assumptions (**Figure 1**). Because most models have uncertain parameters, tuning them by comparing simulated model behaviors and experimental observations and/or functional assumptions is also an essential process in modeling. Performing such model building and systematic verification by maintaining traceability of the bases for parameter settings is essential for accountability, reproducibility, and future revision (evolvability) of the model.

SNNbuilder (https://snnbuilder.riken.jp) is a web-based collaborative tool for data-driven modeling by spiking neural networks (SNN). It allows the collection and management of model parameters of any region of the brain for any species, by virtue of its generic data representation using tables, attributes, and relations in a common database.

SNNbuilder uses neuron and synapse models following the state-of-the-art neural network simulator NEST (Hahne et al., 2021) and manages the data-to-model passage using a set of transfer functions to generate neural parameters and connection rules. Partial data are completed automatically with default values, while alternative and multiple data items from different sources and users are combined as a collective estimation. Model parameters can be specified as "fixed" or "to-optimize" values, as well as assumptions or prior values. Every value is linked to references for traceability.

SNNbuilder creates a model description as a JSON (JavaScript Object Notation) file with full specifications and data modalities, including desired behaviors labeled as "objectives." The framework also generates simulation code in PyNEST (the python bindings of the NEST simulator) for building and simulating SNN models.

SNNbuilder allows an agile modeling workflow, with a primary focus on model specifications. Starting with the main parameters, a model can be created, systematically tested, and can gradually evolve with further data and collaborative contributions. The framework is designed as a web-based, multi-user application with an intuitive graphical user interface (GUI). Considering other tools, as far as we know, SNNbuilder constitutes the first attempt in offering a shared place where many users get together for building collaboratively common models. This is a straightforward way to organize users toward one of the most challenging and important tasks: modeling the complex network of the brain in a thoroughly sustainable manner.

Japan's Brain/MINDS project (Brain Mapping by Integrated Neurotechnologies for Disease Studies, https://brainminds.jp/en/; Okano et al., 2016) is building a multi-scale marmoset brain map with structural and functional imaging. Images are obtained from diffusion MRI, systematic tracer injections (Skibbe et al., 2019; Gutierrez et al., 2020; Watakabe et al., 2021), and many types of fluorescent calcium imaging. SNNbuilder seeks to integrate such diverse, large-scale data into computational modeling, and open data and tools from other brain projects.

2. DESIGN

SNNbuilder is designed as a "web-app," developed using .NET and C#, an open-source developer platform. Its database runs on MySQL, an open-source relational database management system

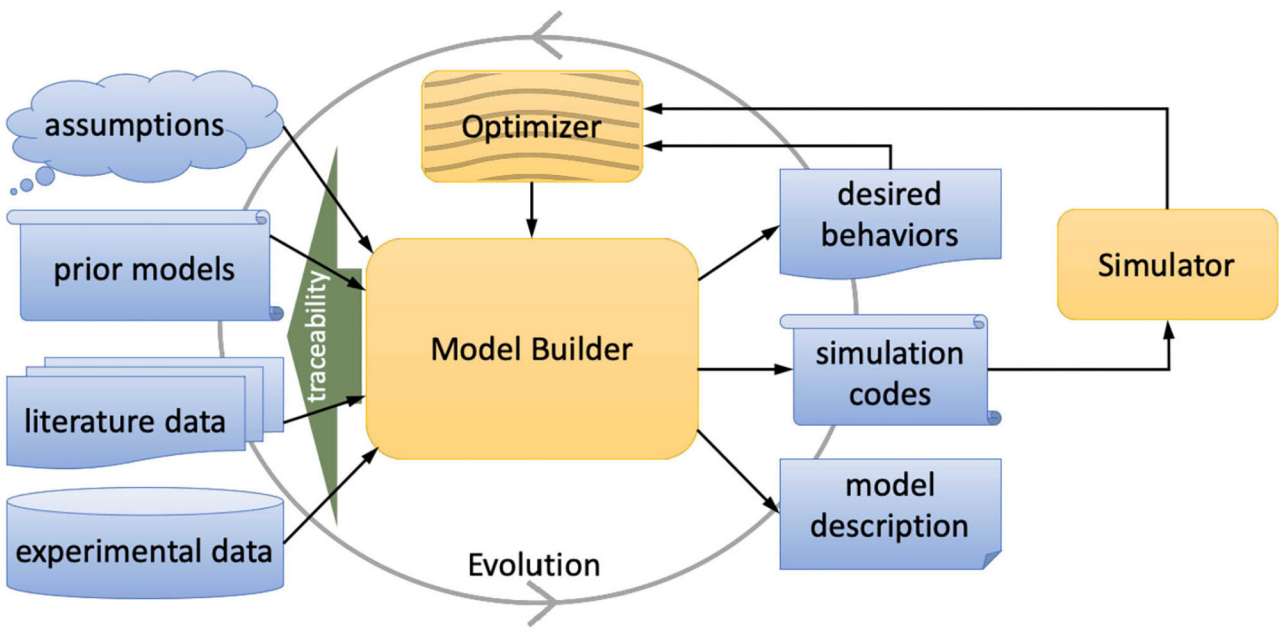


FIGURE 1 | SNNbuilder conceptualization. Different data modalities are loaded through web GUI. The model builder manages the data-to-model process, generating files for systematic simulation (optimization is not included in the current release). This agile process allows the evolution of models.

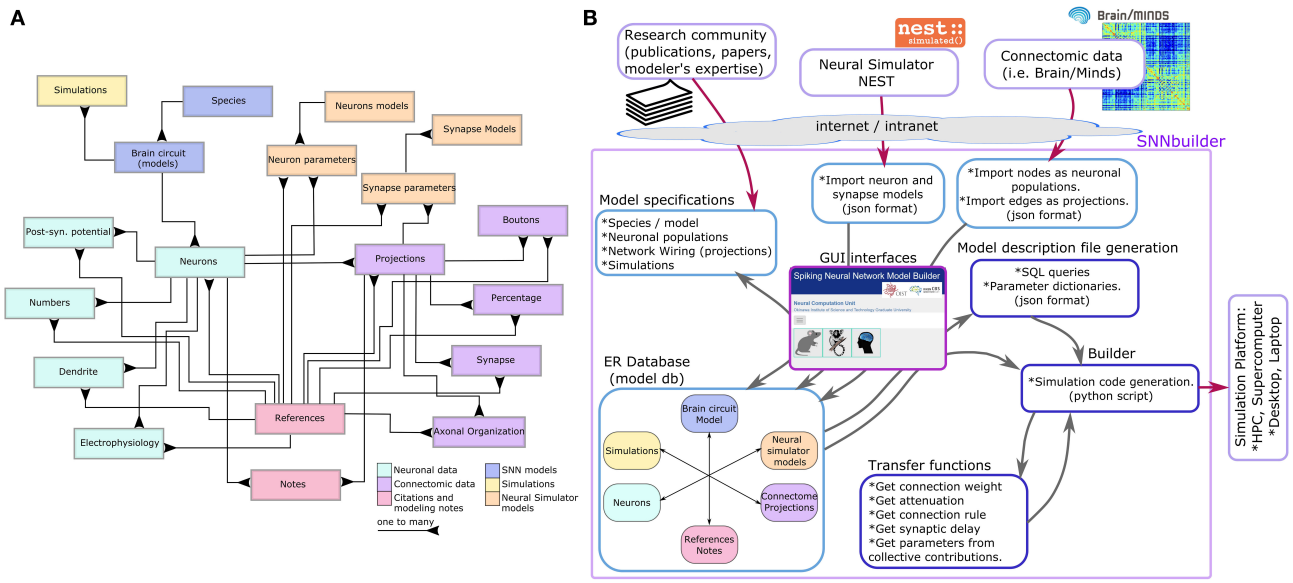


FIGURE 2 | The generic ER (entities and relations) data model **(A)** is able to capture specifications of any region of the brain for any species. Closely related entities are shown as linked groups (color code). The modeling workflow **(B)** runs on GUI. Inputs correspond to research publications, modeler expertise, connectomic data, and models from NEST simulator. Specifications are stored in the database. A builder function performs data mapping as NEST parameters using either transfer functions or direct assignment, creates a model description file, and generates code for simulation.

(DBMS). The selection of a web environment for brain modeling, responds to the importance of the internet as a common shared space that enables users to access from remote locations, perform modeling tasks transparently, and share up-to-date models. For straightforward online collaboration, a login system manages accesses and permissions (see section 3.10).

2.1. Design Principles

From its conceptualization, the framework takes into account modeling principles, as follows:

Fairness and transparency: our framework allows linking model parameters with experimental data, database entries, or scientific publications. References as DOIs (digital object identifier) or URLs can be recorded for every detail of a model. Model descriptions and simulation codes are open to the research community through the web app.

Reproducibility: the framework provides automatic generation of simulation code. Models can be re-built with different choices of source data, and results can be reproduced by simulation of generated codes.

Sustainability: upon the emergence of new papers or experimental data, SNNbuilder allows model updates, such as parameter additions, modifications, and deletions. Rather than building a model for just one point in time, our framework facilitates sustained model evolution.

Collective action: similar experimental studies may produce dissimilar data in different laboratories and at different times. Our framework allows the loading of several values for the same data attribute. In such a way, better parameter settings may be selected by collective contributions from various modelers.

2.2. From Brain Biology to Database Structure

Depending on the region of the brain, degrees of detail and scale, SNN models can incorporate various types of neurons and synapses, as thousands, millions, or billions of components. For that reason, we designed a generic database to support a diversity of models, species, scales, and growing data.

To set up a comprehensive database structure for any model of the brain, we first identified, from brain biology, the most important generic objects that "produce" data, similar to specifying the main components and features on software development projects. We described those "data provider" objects and their relations as entities with multiple data attributes and connections using Entity-Relationship (ER) modeling (Chen, 1976, 2002). ER modeling is commonly used in software engineering for the representation of business needs and processes and provides a business-recognized framework to define the information structure of a relational database.

From our analysis, six main entity groups were identified (**Figure 2A**):

- SNN models: the description of a brain circuit or region to be modeled for a certain species.
- Neuronal data: neuron types or neural populations, including relevant anatomical, morphological, and physiological characteristics.
- Connectomic data: projections between neural populations, along with anatomical and morphological details of network wiring.
- Citations and modeling notes: for reporting origins of data (references), such as DOIs or

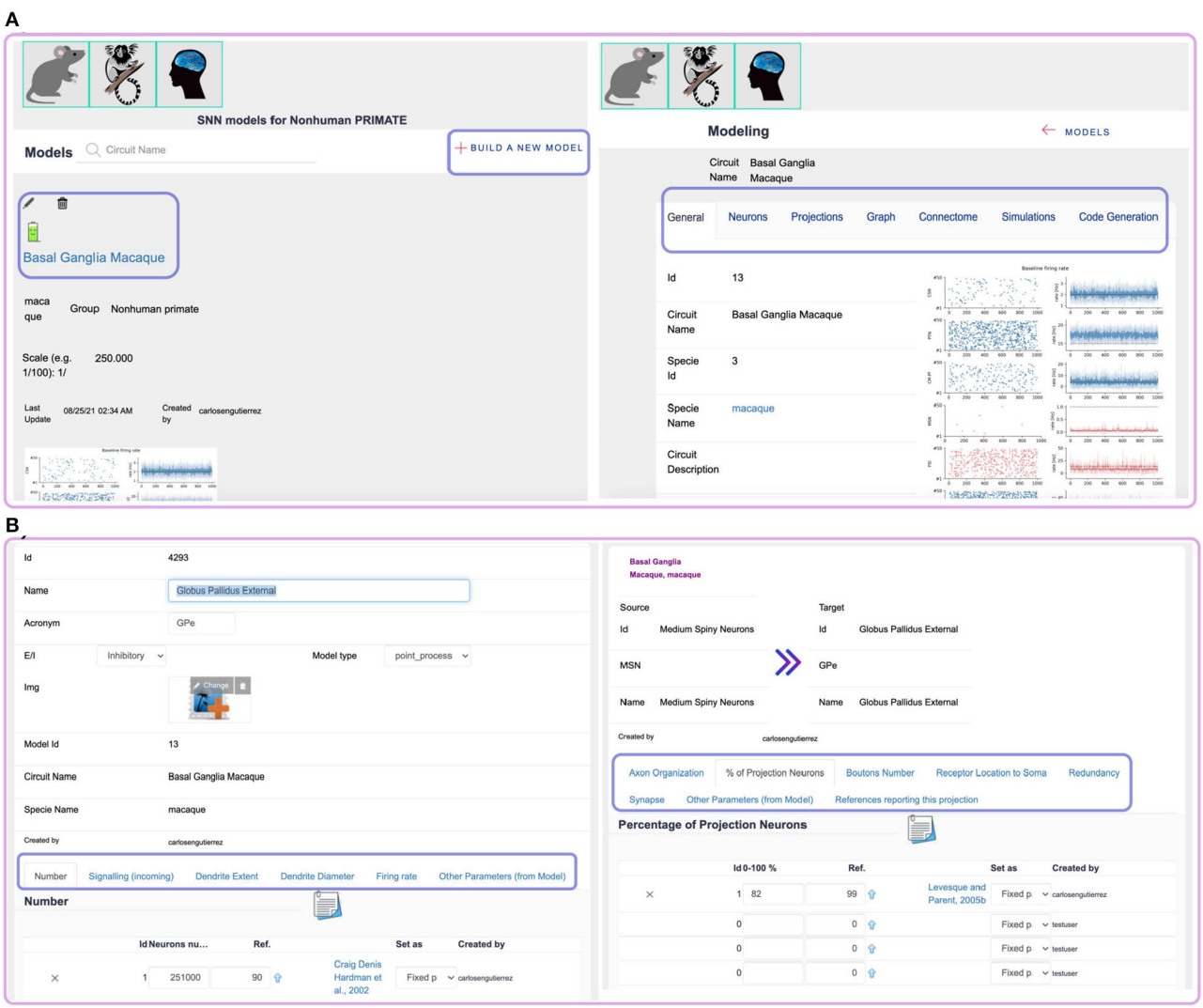


FIGURE 3 | SNNbuilder GUI. **(A)** Left: front-end for species selection and model creation. Right: the modeling workflow is organized by option tabs. **(B)** Left: data requirements for neurons include numbers, PSP, dendrite characteristics, firing rates, and NEST neuron parameters. Right: projection specifications incorporate axonal properties, bouton counts, receptor locations, redundancy, synapses, and NEST synapse parameters.

other URLs, and recording memos over the modeling workflow.

- Neural simulator models: a generic structure to manage data attributes of neuron and synapse models of a neural simulator, like NEST.
- Simulations: for specification of multiple simulations, including stimuli and recordables.

Entities and relations were created in MySQL as tables with primary and foreign keys to preserve data integrity and consistency. The database design applies to any other relational DBMS as well.

3. MODEL BUILDING WORKFLOW

The modeling workflow runs on the GUI (**Figure 2B**), allowing database updates in real time. The process begins by selecting a species, creating a new model instance by the option "Build a new model" and adding a description of the targeted neural circuit or brain region (**Figure 3A** left). At this initial step, the system generates a "model id" for identifying uniquely the model.

Model scale, in relation to biological size, is also specified. Whereas, modelers indicate realistic anatomical data, such as numbers of neurons, bouton counts, axonal domains, a scale parameter adjusts all numbers at code generation time. The scale is relevant for implementation purposes; however, limitations on the reducibility of network sizes (Van Albada et al., 2015) indicate the importance of realistic numbers of neurons and synapses. Given the available computational resources, small scales may run on laptops or desktop computers, while large scales run on servers.

After the initial settings, model specifications (input data) are required (**Figure 3A** right): details of neural

populations, projections or connectivity data, and models from NEST simulator.

3.1. Neural Populations

A neural population is created by an insert operation. This records the population name (or cell type), its excitatory or inhibitory regime, and if available, a related image. Further data requirements are arranged in a tabbed document interface (**Figure 3B** left), organized as Liénard and Girard (2014), as below:

- Number: the number of neurons N within a nucleus at a real scale, considering a single brain hemisphere.
- Signaling: this refers to the neurotransmitter receptor type (AMPA and NMDA for excitatory/glutamate neurotransmitter, GABA for inhibitory/gaba neurotransmitter) of the neuron. Likewise, the PSP (post-synaptic potential) amplitude or change V_n (mV) caused by a single spike mediated by a neurotransmitter n to the membrane potential at the location of the receptor (synapse), and its rise time t_{V_n} (ms).
- Dendrite extent: the average maximal extent l (μm) of the neuronal dendritic field.
- Dendrite diameter: the mean diameter d (μm) of neuronal dendrites along their entire lengths.
- Firing rate: a biologically plausible range $[\phi_0^s, \phi_1^s]$ of the neural population mean firing rate (Hz) for different states s: resting state, excitation (or functional) state, maximum activity, and disease condition. Firing rate is considered a cost function (or objective) and labeled accordingly (see data flags). The future work will consider the integration of an optimization process (see Current limitations).
- Other parameters: parameters of a selected NEST neuron model. Every neural population is paired to a NEST model by an "import from model" operation that selects the neuron model and recalls NEST parameters with default values. After the import, parameter values can be updated. See section 3.3 for more details.
- Objectives/Metrics: it corresponds to user defined objectives and metrics. A configurable set of objectives is available in the main menu (**Figure 7**.3), including, for example, coefficient of variation, inter-spike interval, fano factor (Rajdl et al., 2020), and other arbitrary targets. In this tab section, multiple objectives can be selected and their target values or metrics specified, including the related references. Objectives/metrics are later generated as comments on the simulation script (see Current limitations).

3.2. Projections

An insert operation facilitates data-entry for model connectivity. Projections link the source and target neural populations created in the previous step. Their connectivity is defined by connection rules specified per source-target pair. Further data requirements are organized in a tabbed document interface as well (**Figure 3B** right), with a structure similar to Liénard and Girard (2014):

- Connection rule: it defines the connectivity modality based on NEST connection rules for spatially-structured networks. Indegree- and outdegree-based rules are made available and probability-based, such as constant probability and distance-depended Gaussian probability rules. Transfer functions (see Appendix A) define the indegree and outdegree parameters, whereas a constant probability or SD parameters can be specified by GUI in the case of probability-based rules.
- Axon organization: source-target connection type can be focused or diffused, so synapses are taken from (or made to) neurons within narrow or wide spatial domains, respectively (i.e., a circular or spherical mask). The domain refers to the mean radius (mm or in units relative to the spatial organization of neurons) of a circle (sphere) approximating the 2D shape (3D-volume) of axonal arbors.
- Percentage of projection neurons: the proportion of neurons $P \in [0, 100]\%$ in the source population with axons connecting the target population.
- Bouton number: the mean number of axonal varicosities (or boutons) α where synapses may occur. A biologically plausible range is defined for the sake of exploration; thus, bouton counts are considered as "to-optimize" parameters (see Data flags and Current limitations).
- Receptor location to soma: the mean distance r to the soma of synaptic receptors along dendrites, expressed as a proportion of dendrite extent l. It takes values within ranges for exploration: proximal $r \in [0, 0.2)$, medial $r \in [0.2, 0.6)$, and distal $r \in [0.6, 1]$. It is considered a "to-optimize" parameter (see Current limitations). This parameter is used to calculate an attenuation of the connection weights (see Transfer functions). Specifying r as "None" removes attenuation from connection weights.
- Redundancy (Girard et al., 2020): the mean number ρ of contacts made by axons on each dendritic tree. It is a number between $[1, \nu]$, with ν being the total number of synapses converging on a single neuron. Redundancy can be used to adjust the number of connections and their strength, especially for scaled-down model simulations (see Appendix A: Transfer functions).
- Synapse: records the communication delay (ms) (or axonal delay) of a projection and the corresponding connection weight. If defined, synapse data overwrite the default values of the selected NEST synapse model (see Appendix A: Connection weight).
- Other parameters: parameters of a selected NEST synapse model. Similar to the case for neural populations, every projection is paired to a NEST model by an "import from model" operation that transfers parameters with default values to the projection for further update. See the section "Models of a neural simulator" for more details.

3.3. Models of a Neural Simulator

SNNbuilder uses neuron and synapse models following those of NEST (Hahne et al., 2021), a state-of-the-art simulator for SNN models that focuses on accurate dynamics, varieties of network structure, and scalability for large-scale simulation. NEST provides more than 50 neuron models, over 10 synapse models, and an active support and global community.

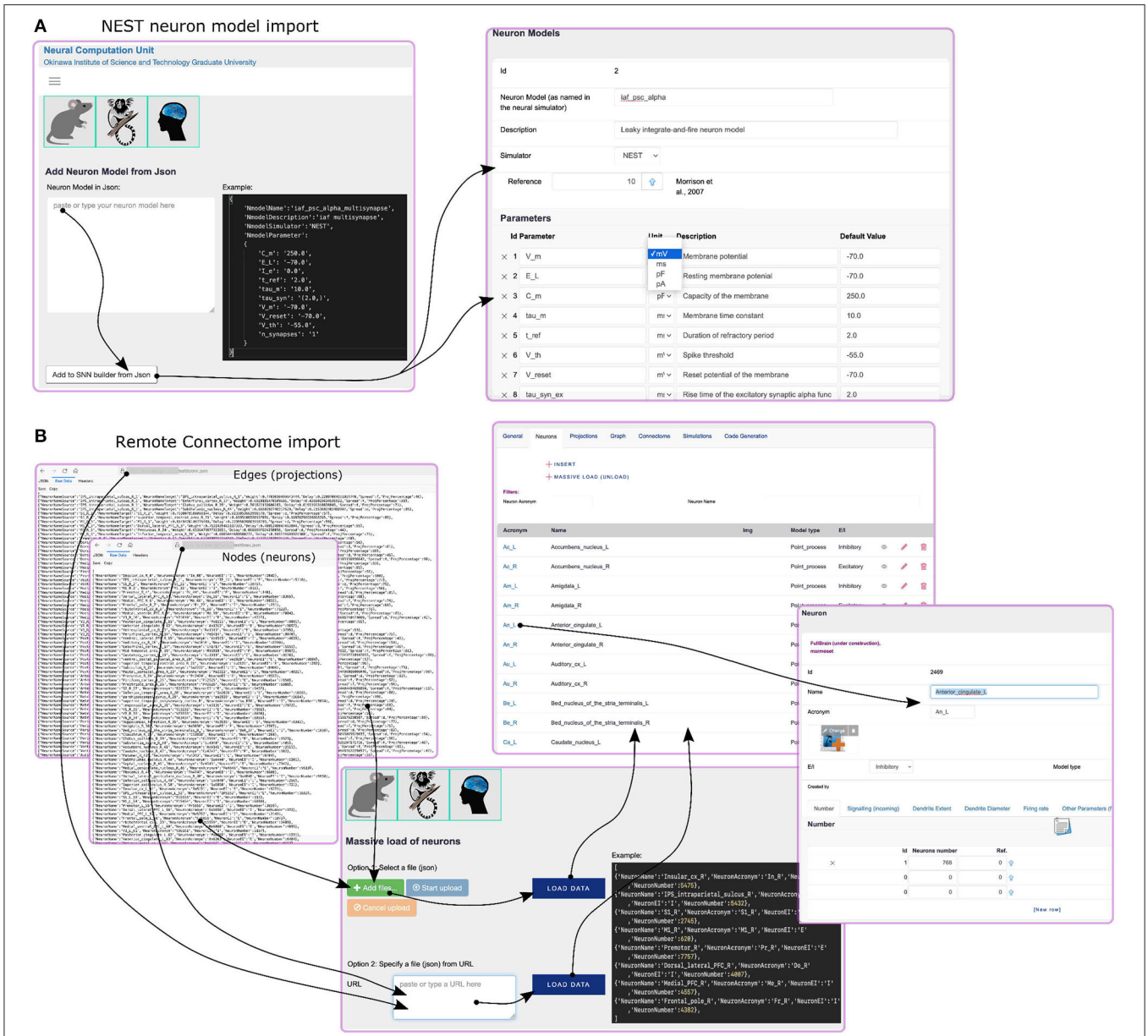


FIGURE 4 | Data load. **(A)** Left: NEST models are incorporated using JSON format data entry in the edit-box area. The required format is shown in the dark background area. Right: after data loading, parameter default values can be updated. **(B)** Import function for connectomic data. Left: connectome nodes and edges are mapped as neuronal populations and projections, respectively; thus, input data are prepared as two JSON files. Center: options for data loading include a file upload or import-from-URL functions. The dark background area shows an example of the required format. Right: after loading to the database, neurons and projections can be explored and updated.

Parameters from NEST neuron or synapse models can be added using the insert or import functions. The latter reads a JSON formatted NEST model from a web edit-box and imports parameter names, descriptions, and default values to the database. The GUI allows manual data-entry or "cut and paste" commands (**Figure 4A** left). The data structure for neuron and synapse models is generic at the database level (**Figure 4A** right), so it can support several neural simulators.

3.4. Data Sources for Modeling

At the time of this report, paper surveys, identification, and manual loading of parameter values are the main activities for model specification at SNNbuilder. Nevertheless, its "online" condition supports potential integration with resources available over the internet, for example, knowledge graphs, public databases, or web services that provide data on-demand, for example, connectomic data (see section 5 and **Supplementary**

Figure B.1). Connectomes are frequently generated as open sources for the advancement of science. Tracer studies, DTI (diffusion tensor image)-based fiber tracking, and functional MRI (magnetic resonance image) data are frequently arranged as region-level (mesoscale data) connectome matrices, where nodes correspond to brain regions and edges to their connections.

To allow such data integration from external sources, our system is prepared to import connectomic data from remote URLs or file-upload. The connectomic data must be provided in JSON format and separated into two files (**Figure 4B** left) for mapping: (i) nodes as neural populations, with specifications (if available) such as population name, number, excitatory/inhibitory regime, and others; and (ii) edges as projections, including available specifications for the axonal delay, connection weight, and others. These functions are available in the GUI (**Figure 4B** center). They provide a straightforward way for incorporating connectomic data and rapid model creation. Moreover, after importing, the user can add additional specifications, implement modifications, assign NEST neuron models, and other improvements (**Figure 4B** right). It is also possible to integrate different data scales (micro, meso, and macro). Data import reduces manual work considerably.

3.5. Data Flags

Flags label characteristics of the data. By default, high-confidence and frequently reported data are considered "fixed parameters," such as neuron numbers, dendrite extent, and diameter, post-synaptic potentials, etc. Parameters, such as axonal bouton counts, and average location of synapses along dendritic trees are considered "to-optimize parameters" and defined as ranges of values for exploration. In the case of multiple entries for the same parameter, a "deactivated" flag is available to "remove" outliers and low-confidence values (**Figure 8**.11). Several activated parameters are possible, a collective "contribution" is calculated in such a case. For multiple numerical values, the average is used (**Supplementary Figure B.2**); in the case of non-numerical multiple values (categories), the appearance frequency is computed as an important index, with the highest frequency value as the collective outcome.

Electro-physiological constraints, such as mean firing rates, are defined as intervals of plausible neural activity, and labeled as "objective functions," crucial for comparisons with simulated neural activity (see Current limitations).

Flag assignment depends on modeler criteria. It is recommended to distinguish between well-known data and poorly documented or inconsistent data from different sources.

3.6. References and Notes

Paper survey-based data entry allows recording of relevant parameter values and data providers and references for traceability. SNNbuilder enables acknowledgment of every detail attached to a model using DOIs or other URLs. Source publications can be accessed and examined directly from the framework GUI (**Supplementary Figure B.2**).

In addition, the GUI includes edit-boxes for digital notes, memos, or comments at every web tab of the workflow. Thus, free-text recording into the database facilitates the creation of a "diary" or "logs," a common practice among researchers.

3.7. Network Viewers

The GUI enables listing and navigating through model specifications, such as neurons and projections; however, to explore a model as a whole, viewers are also helpful (**Figure 5**). A graph-viewer visualizes neuronal populations as boxes (nodes), and projections as edges linking the boxes. The interactive nature of the viewer enables graph exploration and content retrieval from the database.

An additional viewer implements a 2D-matrix visualization for the exploration of connectivity data, such as source and target populations, axonal delays, spatial connection domains, and other network-wiring details. This interface allows straightforward modification of connection weights.

3.8. Simulation Settings

Model simulation criteria are specified by GUI. The main specifications include a description of the simulation, the time resolution (ms), the simulation time (ms), and the number of computational threads. In addition, a common spatial domain for neuron positioning is defined for the sake of consistency and robust simulations in NEST. This version of SNNbuilder supports spatial boundaries (minimum and maximum coordinate values) for randomly and uniformly organized neuron positions in 2D or 3D. Since multiple simulations can be specified for a certain model, different spatial arrangements can be tested.

Specifications of connection weight values might result weak in relation to other parameter values, like membrane resistances, for driving network dynamics during simulation; or too strong, leading to extreme network activity. In those cases, a multiplicative factor affecting the absolute value of all synaptic weights can be defined by the user, called synaptic scaling gain. In this way, simulations can be performed while maintaining, relatively, the specified connection strengths of the neural network model.

Besides simulation settings, details of the stimuli and recordable are also defined. Several simulations can be specified; however, only one should be activated by assigning the corresponding data flag for code generation. Otherwise, the first active simulation is considered at the PyNEST script.

3.8.1. Stimuli

Stimuli are designed as independent spike trains from NEST Poisson generators (see Current limitations) and specified by GUI (**Figure 9**.26). A Poisson generator is created per target population with configurable firing rate (Hz), connection weight, axonal delay (ms), and the start and stop times (ms) of the stimulation, along with its scope. The scope refers to either Poisson spikes trains are sent to all neurons in the target (global scope) or to a spatially-bounded subset of neurons. The spatial bounds are defined by a point (position coordinates) and a radius parameter, which determines the neurons within a circle or sphere under the stimuli.

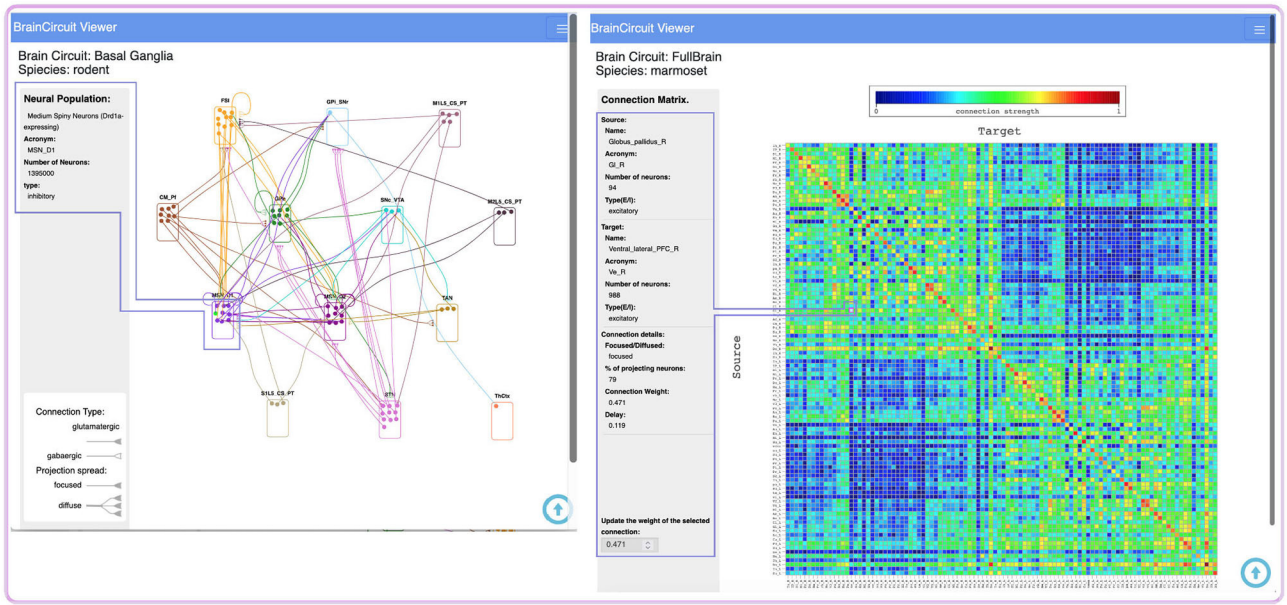


FIGURE 5 | Network viewers. Graph (left) and matrix (right) viewers for exploration of model specifications with interactive data retrieval (area enclosed by blue lines). The latter enables selecting a connection within the matrix and upgrading its weight.

3.8.2. Recordables

There exist two NEST-based recordable options for spikes and membrane potential that define what gets recorded during simulation time. Multiple recordable can be specified, with a single one targeting a single neural population. Spike-type recordable stores the spike times of all neurons at the target population; while membrane potential-type recordable selects, at random, a single neuron for recording the evolution of its membrane potential. Recordables generate, automatically, output data files.

3.9. Model Description and Code Generation

The workflow's final step corresponds to procedures for organizing SNNbuilder output. This includes the generation of a comprehensive list of model specifications, parameter passage to NEST models, and the generation of simulation code for creating neural populations, recording devices, network wiring, and stimuli.

Automatic generation of code is practical for immediate testing, different model configurations, and versions. High-level programming skills are not required and modeling time is used mainly for definition of biological constraints.

Model description and simulation code files are made available through 3 sequential processes (**Figure 9**.27) implemented in Flask (Grinberg, 2018), a python-based web development framework, and executable on GUI:

(1) Get parameters: for a particular model, this process runs SQL (structured query language) queries on the database and gathers the previously specified data for that model. Retrieved data are converted to python dictionaries and arranged in a single JSON file as the model description. In this step, queries make use of the data-flag specifications to filter parameters and compute collective contributions. Parameter values labeled as "deactivated" are not considered. In the case of multiple numerical values loaded for a specific parameter, the average value is considered as the collective outcome (**Supplementary Figure B.2**) and computed at query time. In the case of multiple categorical data, category frequencies are calculated as an "importance index." The most weighted index is selected for parameter initialization. Queries may retrieve dictionaries with "None" records for parameters with no available data. In such cases, default settings are assigned in the next step.

(2) Code build: a builder function takes the JSON file generated at 1) applies transfer functions (see Appendix A) and the specified scale and creates the simulation script in PyNEST for NEST 3. For robust simulations, the builder generates straightforward lines of code (LOC) in the following sequence:

- Initialization: includes LOC for importing the necessary python packages. NEST kernel initialization, and the definition of global variables.
- Creation of neural populations: the process takes parameter values and creates LOC for initialization of neural populations. Parameters from (1) are mapped to NEST neuron and synapse models, updating default values. NEST defaults remain when "None's" are present at parameter specifications. Neuron numbers are adjusted based on the defined scale parameter. Signaling and PSP values set up neuron receptors and synaptic delays. Neuron positions are created in 2D or 3D space, by using a uniform random distribution, within spatial bounds

defined at the simulation settings. Given the specified NEST neuron model and its parameters, LOC for the creation of the neural population is generated.
- Network building: this takes network-wiring details from specifications at (1). For connected population pairs, parameters are mapped to NEST synapse models and connection dictionaries, and LOCs are created for connection rules. For more details on connection weight definition and connection rule parameters, see Transfer functions (Appendix A).
- Stimuli, recordables, and simulation: given the specified stimuli and targets, the process generates LOCs for the creation of Poisson spike-train generators and initialization of their firing rates, spatial scope, and other parameters. Additional LOC for the creation of recording devices of neuronal activity (spikes) or membrane potentials are also included. Finally, the process creates LOC for NEST simulation commands with a defined biological time.

The tool does not include on-line code execution or execution management on its first release (see Current limitations and Discussion).

(3) File download: this takes the results from 1 (model description as JSON file) and 2 (simulation code as python script file), packs them into a zip file, and delivers them. Although the python script runs stand-alone, the specification file is made available for future parameter optimization (see Current limitations).

3.10. Collective Intelligence

The online and centralized database aspects of our approach allow a modern form of collaboration called "collective intelligence." SNNbuilder is designed for multi-user access. Another important feature is the assignment of multiple values for the same parameter. Diverse input values for a single parameter improve its reliability (**Supplementary Figure B.2**), see Data flags and Model description and code generation sections). Over time, settings evolve to better values through different contributions of more users and new data, gradually converging to the most realistic ones.

These characteristics promote "collective intelligence," where humans (and computers) working together act much more intelligently in a collective way than individually (Malone, 2018). As demonstrated by crowd-sourcing experiences (Brabham, 2013), a "bigger brain" works better than a small one. By this collaboration scheme, better models of the brain can be collectively built and shared online across the scientific community. In addition, SNNbuilder includes functions for maintaining digital notes or memos (see Reference and notes section), available at every tab of the GUI (**Supplementary Figure B.2**). In this way, users can record and share comments, questions, and logs over the workflow.

To support this scheme, the application implements a login system for user identification and automatic labeling (tags) of user contributions. When a model is created (**Figures 3A**, 7.3), the owner has the choice to "open" the model to the community; in that case, multiple users can visualize, add or update records, and generate simulation code. Otherwise, the model remains close, and only the owner can access it to perform updates. Every record is owned by a specific user. Security rules disable the deletion of different user contributions; thus, only self-owned records can be removed or disabled.

3.11. Current Limitations

The present release of our work reports some limitations not yet solved or implemented.

Specifications are mainly fixed parameter values (numerical, categorical, or descriptions). Detailed models may require distribution-based values for some parameters, such as connection weights, synapse locations, resting membrane potentials, which are not yet included (see section 5). Complex experimental settings or detailed models may require the incorporation of functional-based metadata for setting up neuronal parameters and connectivity features; however, our application does not support that aspect.

The firing rate specifications can take numerical values and description tags indicating the "state" related to the neural activity, for example, resting, excitation, and disease states; however, the state is not linked to a certain stimulation protocol for its effective simulation. In the present release, states are enabled only for the characterization of the targeted activity.

Specifications of a model cannot be re-used by other models. SNNbuilder considers constantly evolving models; thus, parameter history is not maintained and the latest specifications are taken at code-generation time. Model versioning is not implemented (see Discussion). Data modifications, additions and deletions from multiple users are not tracked over the building workflow; however, ownership records are maintained for acknowledgment of the different contributions.

The current version of our tool provides a subset of the available NEST connection rules. Stimuli are defined using Poisson spike trains, there is no other stimulus modality implemented. Optimization is also not yet included; nevertheless, the database structure was designed to support an optimization engine (see Discussion). The system allows the import of connectomic data; however, the data require preparation in a specific JSON format (see section 5). Furthermore, there is no functionality for accessing HPC resources; therefore, simulation code cannot be executed within SNNbuilder. Code execution steps are managed by the user.

4. MODELING EXAMPLES

We tested SNNbuilder by building two models: a balanced cortical network (Brunel, 2000) showing self-sustained asynchronous-irregular (SSAI) activity (Kriener et al., 2014) and a model of the mouse striatum (Hjorth et al., 2020) reproducing resting state activity (**Figure 6**).

4.1. Self-Sustained Network

Networks of spiking neurons can show SSAI firing under a certain balance of excitatory and inhibitory transmission, with no need for random background input. We reproduced a cortical network model with excitatory and inhibitory populations

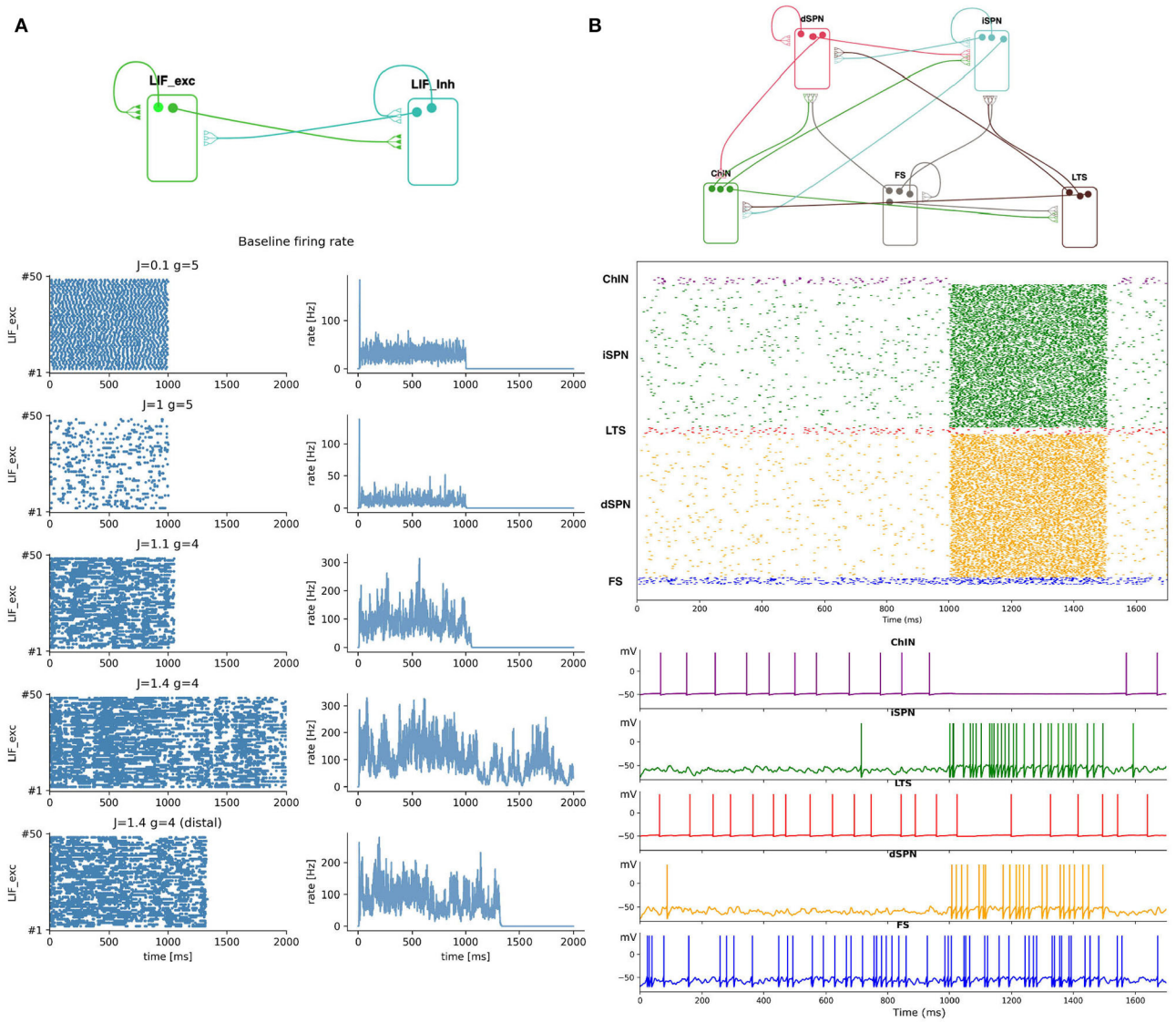


FIGURE 6 | Example models. **(A)** Cortical network simulations with different values of the coupling strength between excitatory and inhibitory populations. Strong coupling allows self-sustained asynchronous irregular activity (SSAI) after stimulus offset (1,000 ms). A single change of the receptor location (synapse) to soma, from proximal to distal (plot at bottom), reduces SSAI lifetime. **(B)** Striatal microcircuitry. Rasters correspond to baseline activity of the striatal nuclei (0–1,000 ms), as well as activity triggered by a strong cortical input (1,000–1,500 ms). Single neuron voltage traces are shown for the same simulation (bottom).

(Brunel, 2000) and explored the generation and duration of SSAI state based on examples from Kriener et al. (2014).

The model was built following the steps below:

- Login to SNNbuilder (https://snnbuilder.riken.jp) (**Figure** 7.1).
- Select the target subject, for example, rodent (**Figure** 7.2).
- Create a model instance using "Build a new model" option (**Figure** 7.3).
- Specify a model name, scale, and other descriptions (**Figure** 7.4).
- Select the model name (**Figure** 7.5) to show tabs for model details (**Figure** 7.6).
- Select "insert" in the "neurons" tab to load data for neural populations (**Figure** 7.7).
- Create excitatory and inhibitory neural populations (**Figure** 7.8).
- Input additional data required in several tabs (**Figure** 7.9).
- In "Number," specify $N_{exc} = 10,000$ and $N_{inh} = 2,500$ for excitatory and inhibitory neurons respectively (**Figure 8**.10).
- Specify post-synaptic potentials (PSPs) for excitatory (AMPA) and inhibitory (GABA) receptors as alpha-functions with a common value of $t_{V_n} = 0.5ms$ (rise time of the synaptic function), and PSP amplitudes $V_{exc} = J$ and $V_{inh} = gJ$ (**Figure 8**.11). This allows exploration of relative inhibitory

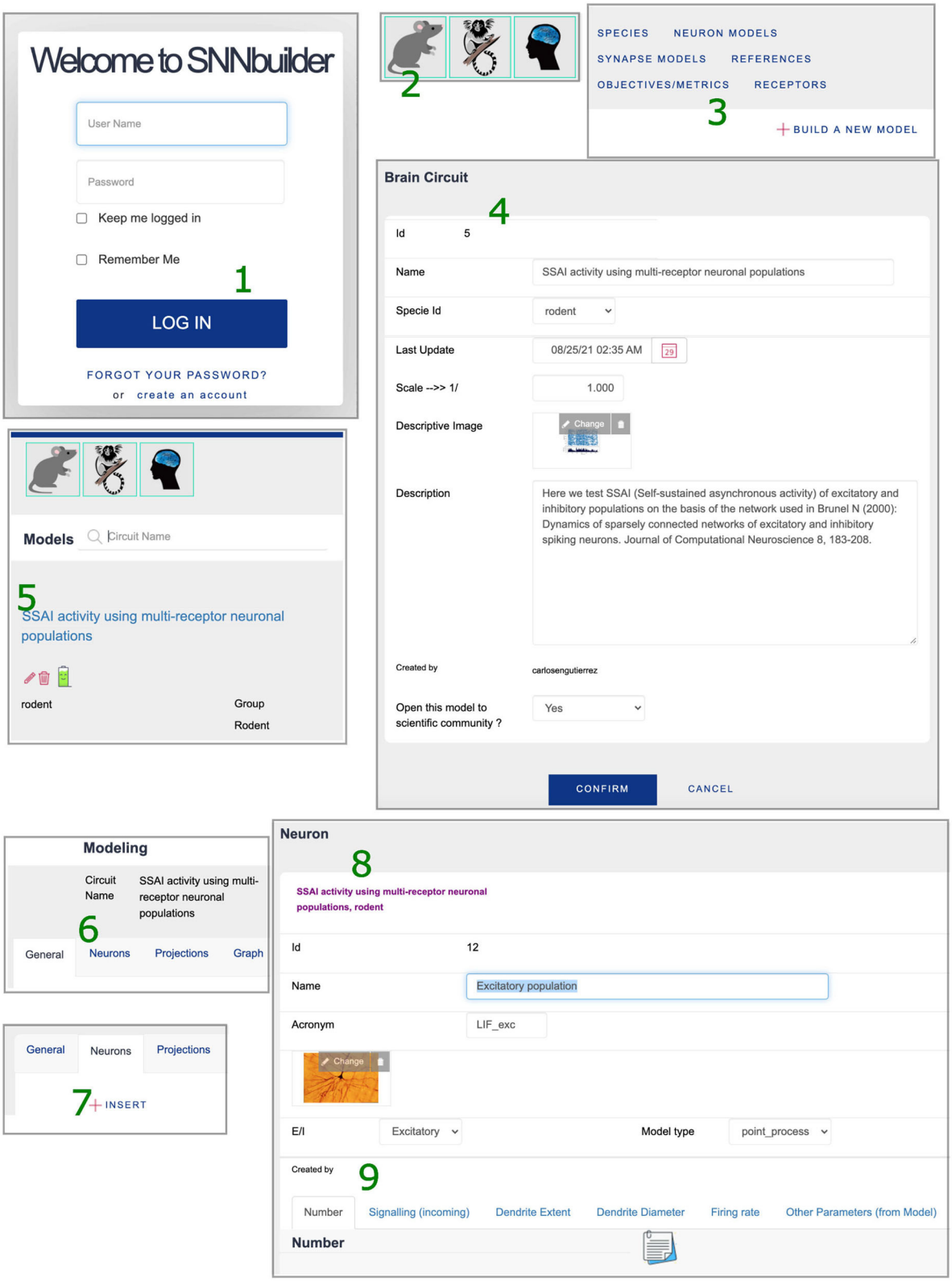


FIGURE 7 | Model creation at SNNbuilder. 1. Login system. 2. Selection of species (subject). 3. "Build a new model" option. 4. Initial specifications. 5. The created model is listed at the front-end. 6. Tabs for modeling workflow. 7. "Insert" option at "Neurons" tab. 8,9. Required specification for neuronal populations.

FIGURE 8 | Neuronal population specifications. 10. Number of neurons. 11. Receptors, PSP amplitudes, and rise times. 12,13. Dendrite characteristics. 14. "Import from model" option. 15,16. Selection of a NEST model and migration of default parameters.

strength by activating a single pair (i.e., dotted square at **Figure 8**.11) as a "fixed parameter" while "deactivating" others.

- Set generic values for the neuronal dendritic extent $l_x = 600\mu m$ (**Figure 8**.12) and dendrite diameter $d_x = 1.6\mu m$ (**Figure 8**.13).
- Implement neural populations as multi-synapse LIF (leaky integrate-and-fire) neurons by the option "import from model" at "Other parameter" tab (**Figure 8**.14). This enables the selection of a NEST neuron model (**Figure 8**.15) and transfer of parameters with default values to the neural populations (**Figure 8**.16).

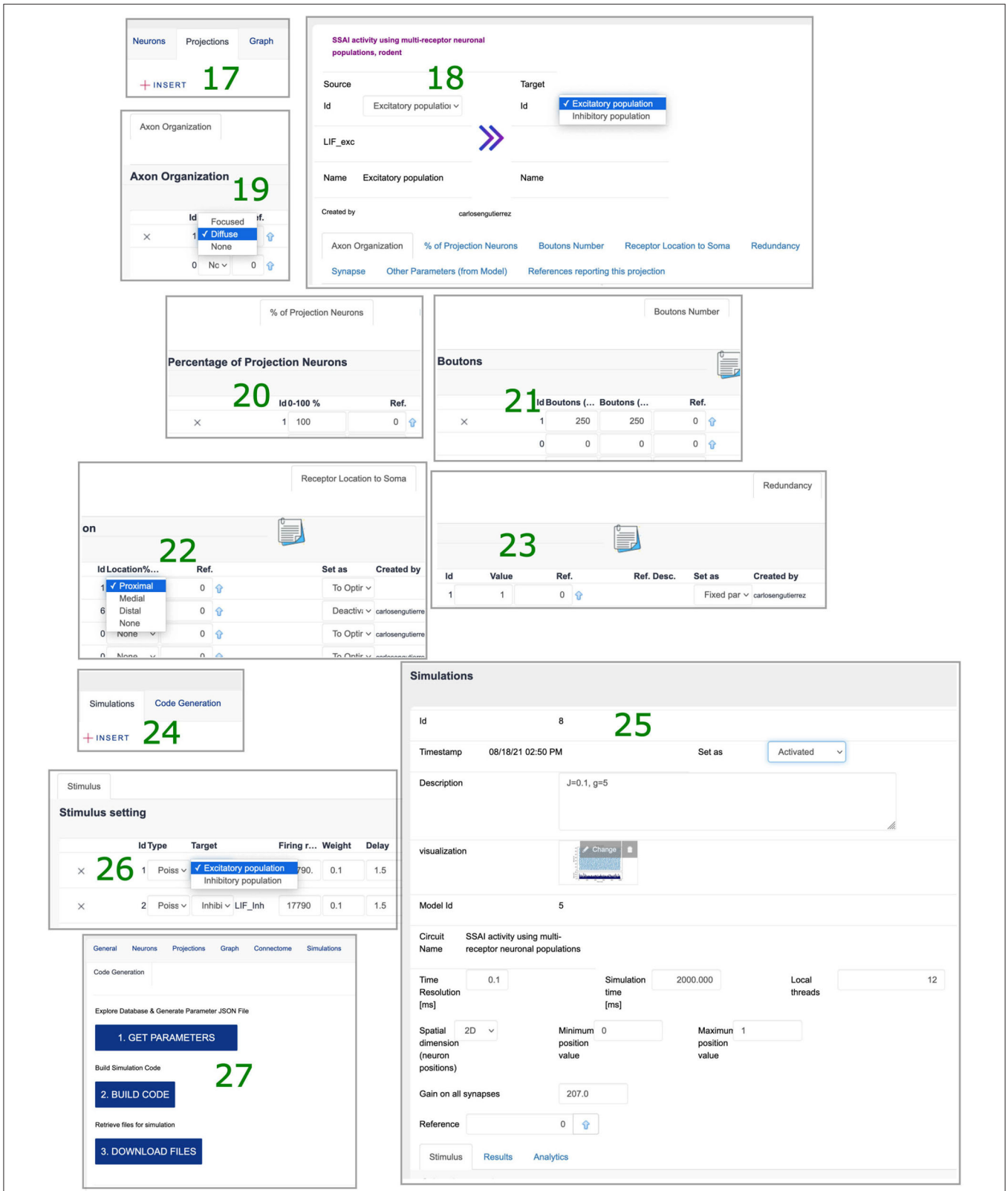


FIGURE 9 | Projection and simulation specifications. 17. "Insert" option at "Projections" tab. 18. Data requirements for network wiring. 19. Axonal organization. 20. Percentage of neurons at source projecting to target. 21. Bouton counts. 22. Synapse location to soma. 23. Redundancy parameter. 24. "Insert" option at "Simulations" tab. 25. Simulation settings. 26. Stimuli to target populations. 27. Model description file and simulation script generation.

- Specify connectivity details using the "Projections" tab, with the "insert" function (**Figure 9**.17).
- Link source and target neural populations and add connectivity specifications by navigating the additional tabs (**Figure 9**.18).
- Set axonal organization as "diffuse" (**Figure 9**.19), with a wide spatial domain, in order to emulate random networks in which neurons are independently connected with an equal probability ϵ.
- Define the percentage of source neurons projecting to the target population as 100% (**Figure 9**.20).
- Assume $\epsilon = 0.1$ to define bouton counts from projections at excitatory neurons as $\alpha_{\{exc,inh\}\rightarrow exc} = \epsilon \times N_{exc}$, and at inhibitory neurons as $\alpha_{\{exc,inh\}\rightarrow inh} = \epsilon \times N_{inh}$ (**Figure 9**.21).
- Specify the synaptic location to soma r_x as "proximal" for establishing a minimal PSP attenuation, with a distance within 20% of a generic dendritic extent (**Figure 9**.22).
- Set a generic redundancy value $\rho = 1$ (**Figure 9**.23).
- Implement projections as NEST static synapse models, with default parameter values, similar to the NEST neuron's case (**Figure 8**.15).
- Create a simulation using the "insert" function (**Figure 9**.24), specify time resolution $dt = 0.1ms$, simulation time for $2,000ms$, and spatial organization of neurons in 2D-space with coordinates (x, y) randomly generated between [0, 1] and other features (**Figure 9**.25).
- Add stimuli for the first 1,000 ms as independent Poisson spike trains of constant rate (**Figure 9**.26).
- Generate model descriptions and simulation code in three sequential steps: get parameters, code building, and files download (**Figure 9**.27).

Simulations run with different values of J and g (**Figure 6A**), for example, $J = \{1.1, 1.4\}$ and $g = 4$ showed different network activities after stimulus offset. While the lifetime for $J = 1.1$ was almost zero, $J = 1.4$ sustained the firing rate, allowing a longer lifetime. Thus, a stronger coupling strength drove the network over the whole simulation time. As reported in Kriener et al. (2014), the SSAI state showed highly irregular spiking activity, with neurons switching between periods of silence or low firing rate, and short bursts or elevated rates, while the average activity of the neural population persisted over the simulation time. It is worth noting that values of J and g are not directly comparable to those reported by Brunel (2000) and Kriener et al. (2014), since attenuation is applied on the PSP strengths based on dendrite parameters (see Transfer functions in Appendix A).

As an additional test, for the latter parametrization, we observed that the SSAI state is affected by a single change in neuron's morphology: a "distal" location r_x of the receptors in relation to the soma (**Figure 9**.22) shortened SSAI lifetime (**Figure 6A**). SNNbuilder easily enabled model changes and code generation for straightforward analyses.

4.2. Striatal Microcircuitry

Rodent local striatal microcircuitry has recently been modeled (Hjorth et al., 2020) using the NEURON simulator (Carnevale and Hines, 2006); however, such simulation involves heavy computations due to detailed cell morphologies. We aimed to replicate these results using point-process neurons, which are computationally much less expensive for systematic analysis of model dynamics.

A network was built following data from Hjorth et al. (2020), comprised of 38,237 direct striatal projection neurons (dSPN), 38,237 indirect striatal projection neurons (iSPN), 1,047 fast-spiking (FS) interneurons, 644 low-threshold spiking (LTS) interneurons, and 886 cholinergic interneurons (ChIN). All neuronal types were implemented as LIF with AMPA and GABA receptors, with PSPs modeled as alpha-functions with specific amplitude values for each connection. PSPs were specified as connection weights at the projection synapse parameters, rather than at the neuron receptor level. The axonal delay was assumed generic for all the connections, equal to $0.2ms$. Neuron parameters, such as membrane time constant, threshold, and resting membrane voltage were taken from Johansson and Silberberg (2020).

Neurons were uniformly distributed in a $1mm^3$ volume, matching the neuronal density in the striatum (Rosen and Williams, 2001). Connections were created using a fixed in-degree rule and a spherical mask with size based on axonal and dendritic field diameters. Other connection-related parameters, such as bouton number, the distance between soma and synapse, and the number of synapses from a single source were also taken and calculated by Hjorth et al. (2020).

Two levels of external input were modeled in the network: the first 1,000 ms of simulation correspond to 2 Hz glutamatergic baseline activity from the cortex and thalamus. In order to simulate synaptic input, all neurons were assumed to have 150 AMPA synapses, all receiving independent inputs that can be modeled as a 300 Hz Poisson spike train, similar to what is described in Hjorth et al. (2009). The next 500 ms correspond to higher-level cortical activity, defined as an $8Hz$ glutamatergic input, and modeled as a 1,200 Hz Poisson spike train superimposed on the baseline input train, after which baseline activity is restored.

The workflow final step, the code generation, provided a python script for simulations. An optimization step, not implemented by the current SNNbuilder release, was performed for this modeling example (see Current limitations). Both local (inhibitory) and external input (excitatory) weights to each population were optimized simultaneously, using grid search and a custom multi-objective function: for each population and for each stimulation regime, a target range of plausible firing rates was defined, and error was defined as the normalized distance to the center of that interval. The set of weights that minimized this error was then selected, with the firing rates of all populations matching those described in Hjorth et al. (2020).

The optimization process, first, found weights that yielded good behavior for the baseline activity (from 0 to 1,000 ms, **Figure 6B**). Once optimized, these values were active during the whole simulation (from 0 to 1,500 ms). Then, new Poisson generators were introduced corresponding to a higher level of cortical activity (from 1,000 to 1,500 ms) and whose weights were optimized while keeping the baseline ones fixed.

A raster plot of the network activity after optimization is shown in **Figure 6B** and voltage traces (mV) of single neurons, in which the two levels of activity are distinguishable.

5. DISCUSSION

Modeling the brain is a challenge that requires collective effort. Large-scale cohesion of researcher knowledge, ideas, publications, and experimental data can be realized on the internet, where humans are hyper-connected, constituting a convenient frame for brain modeling. We have designed SNNbuilder as a web-application to support the collaborative building of sustainable, renewable, and scaleable SNN models.

The introduced framework organizes specifications to model any region of the brain through a straightforward GUI (**Figure 3**). Anatomical, morphological, and physiological data are systematically loaded and updated, and their passage as neural and synaptic parameters is managed by transfer functions (see Appendix A). A generic relational database (**Figure 2A**) is designed to accommodate accumulating data and includes references and notes to accurately acknowledge data sources and to trace model details (**Supplementary Figure B.2**). SNNbuilder workflow (**Figures 3**, **7**, **8**, **9**) was tested on two model examples: a self-sustained asynchronous irregular network and a model of mouse striatal circuitry (**Figure 6**).

Major data sources are scientific publications. Paper surveys require the identification of relevant parameters for modeling, which is time-consuming for humans. Efforts are ongoing to extract data from a large collection of literature and to store the data in open databases. For example, Bjerke et al. (2020) standardized and quantified information about cellular parameters in the murine basal ganglia from public repositories, and Tripathy et al. (2014) extracted electrophysiological properties of diverse neuron types from existing literature. The desired future extension is SNNbuilder compatibility with open database sources, not only for consuming plain data but also for incorporating automatic discovery of parameters by text-mining algorithms and knowledge-graph building.

Compatibility with resources, such as EBRAINS, Brain/MINDS, and NeuroML, are crucial for improving the modeling process. We aim for SNNbuilder-to-application and SNNbuilder-to-databases compatibility, so system input(s) and output(s) can be shared and integrated. A preliminary effort corresponds to the SNNbuilder capability to import NEST models and connectomic data from JSON files, including remotely located files for the latter case (**Figure 4** and **Supplementary Figure B.1**). By this functionality, upon the opening of data, marmoset connectomic retrieval by a web service at Brain/MINDS is possible in the short term. This will provide full or partial data for loading, automatically, neurons and projections in SNNbuilder. An architecture composed of web-services or APIs for straightforward access to SNNbuilder data and models, and web-services for data consumption from open sources (**Supplementary Figure B.1**) is required and considered as future challenge. Moreover, an important standard supporting data sharing across brain projects is Neurodata Without Borders (NWB, Teeters et al., 2015; Rübel et al., 2021). SNNbuilder management of inputs/outputs in NWB format will be considered as well in future releases.

New system functions and features are required as well and included in future work, especially needed for building complex experimental settings and detailed models. For example, parameter definition based on distributions, stimuli protocols associated with objectives or metrics, integration of functional-related metadata, re-use of parameters from other models, new transfer functions and more connection rules, model versioning and history tracking, online simulation management, and parameter optimization.

Our generic database structure supports the future implementation of an optimization engine. Optimization will preserve parameters labeled as "fixed" while exploring "to-optimize" parameters within defined value intervals, assessing model activity against data labeled as "objectives." The generic character of data-to-model conversion will allow comparisons across species as well, since models for different subjects along with their simulation results can be compared on the same dimensions.

A further challenge is code generation for multiple simulators, which may require the development of new mapping processes (transfer functions). On this, compatibility with NeuroML, a simulator-free approach for model description may provide strong advantages. Currently, the SNNbuilder model description in JSON format corresponds to dictionaries listing the specifications. That output could be prepared in compatible XML (Extensible Markup Language) format, following the standard NeuroML. Model description in NeuroML enables simulations on different tools, avoiding code generation/preparation for multiple simulators. Having that advantage, model specifications can be prepared to target biophysical neuron models and complex networks; thus, SNNbuilder may support the state-of-the-art NEURON simulator. Our database will be extended in such a case, for the inclusion of new data entities like detailed morphology and ion channels.

SNNbuilder plans to enable online simulation and result analytics, by simulating models on the server and visualizing results *via* the web-browser, as shown in Spreizer et al. (2021). That will facilitate immediate building-testing iterations for an intuitive understanding of model dynamics. Big spike data could be loaded into the database and meticulously queried and plotted for better interpretation of results. While small simulations could be triggered instantly, large simulations can be prepared and dispatched for high-performance computing, as shown in Feldotto et al. (2022). SNNbuilder straightforward code generation may include job scripts for the setup and simulation of large-scale models on the Fugaku supercomputer (Sato et al., 2020), as well on collaborative simulation infrastructures like Fenix from EBRAINS (Alam et al., 2019). Compatibility with distributed computational resources facilitates the access and usage of services and is included in our future challenges.

As an introductory video from the International Brain Initiative observed, "It takes the world to understand the brain. It is the most complex organ in the human body"

(Adams et al., 2020). Understanding the brain requires not only biological data but also tools to enable the engagement of a diversity of researchers, with different backgrounds and opinions, to support independent, free contribution of ideas. Our framework supports that collaboration for modeling the brain.

DATA AVAILABILITY STATEMENT

SNNbuilder is currently running on the internet and is available at: https://snnbuilder.riken.jp. A test user for tool exploration and code generation of existing models is available (user: testuser password: snnbuilder). For full privilege users, please send a request to carlos.gutierrez@oist.jp. Source code and data base structure are available upon request. The authors of this work are open to discussion and collaboration. Feedback for improving our work is gladly welcome and appreciated.

AUTHOR CONTRIBUTIONS

CG designed and developed SNNbuilder, modeled a self-sustained network example, and ran simulations. HS helped with the framework compatibility with Brain/MINDS connectomic data, and GUI design. HM modeled the striatal circuitry example, ran simulations and optimizations, and supported debugging. KD helped with SNNbuilder concept, design, and future integration within the International Brain Initiative (IBI). All authors contributed to the writing of the manuscript.

FUNDING

This research was supported by the Collaboration Research for Development of Techniques in Brain Science Database Field and the Collaborative Technical Development in Data-driven Brain Science grants from RIKEN Center for Brain Science, the program for Brain Mapping by Integrated Neurotechnologies for Disease Studies (Brain/MINDS) JP18dm0207030 and 21dm0207001 from the Japan Agency for Medical Research and Development (AMED), the Post-K Application Development for Exploratory Challenges (hp160266, hp170251, hp180223, and hp190157) from Ministry of Education, Culture, Sports, Science and Technology of Japan (MEXT), the KAKENHI Grant 16H06563 from Japan Society for the Promotion of Science (JSPS), and internal funding from the Okinawa Institute of Science and Technology Graduate University to KD.

ACKNOWLEDGMENTS

We appreciate the support from Joona Pulliainen and Siang Sheng Jheng from the OIST Scientific Computing and Data Analysis section for providing a development web-server, computing resources, and technical support. We thank Tomomi Shimogori and Yoshihiro Okumura from the Center for Brain Science RIKEN for supplying a production web server, and computational resources for SNNbuilder public deployment. We express our gratitude to Genexus Japan (https://www.genexus.jp), for making available an academic license of their product to accelerate web-system development. We thank Ryoji Furugen (http://polymorph.jp) for his contribution to the graphical interface. Big thanks to Jean Lienard and Benoit Girard for their advice on brain modeling and parametrization.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.855765/full#supplementary-material

REFERENCES

Abbott, A. (2021). How the world's biggest brain maps could transform neuroscience. *Nature* 598, 22–25. doi: 10.1038/d41586-021-02661-w

Adams, A., Albin, S., Amunts, K., Asakawa, T., Bernard, A., Bjaalie, J. G., et al. (2020). International brain initiative: an innovative framework for coordinated global brain research efforts. *Neuron* 105, 212–216. doi: 10.1016/j.neuron.2020.01.002

Alam, S., Bartolome, J., Bassini, S., Carpene, M., Cestari, M., Combeau, F., et al. (2019). "Fenix: distributed e-infrastructure services for ebrains," in *International Workshop on Brain-Inspired Computing* (Cetraro: Springer), 81–89.

Asai, Y., Abe, T., Okita, M., Okuyama, T., Yoshioka, N., Yokoyama, S., et al. (2012). "Multilevel modeling of physiological systems and simulation platform: physiodesigner, flint and flint k3 service," in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet* (Izmir: IEEE), 215–219.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: a python tool for building large-scale functional brain models. *Front. Neuroinform.* 7, 48. doi: 10.3389/fninf.2013.00048

Bjerke, I. E., Puchades, M. A., Bjaalie, J. G., and Leergaard, T. B. (2020). Database of literature derived cellular measurements from the murine basal ganglia. *Scientific Data* 7, 1–14. doi: 10.1038/s41597-020-0550-3

Brabham, D. C. (2013). *Crowdsourcing*. Cambridge, MA; London: MIT Press.

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci.* 8, 183–208. doi: 10.1023/A:1008925309027

Carnevale, N. T., and Hines, M. L. (2006). *The NEURON Book*. Cambridge: Cambridge University Press.

Chen, P. P.-S. (1976). The entity-relationship model–toward a unified view of data. *ACM Trans. Database Syst.* 1, 9–36. doi: 10.1145/320434.320440

Chen, P. P.-S. (2002). "The entity relationship model–toward a unified view of data," in *Software Pioneers* (Berlin; Heidelberg: Springer-Verlag), 311–339.

Dai, K., Gratiy, S. L., Billeh, Y. N., Xu, R., Cai, B., Cain, N., et al. (2020). Brain modeling toolkit: an open source software suite for multiscale modeling of brain circuits. *PLoS Comput. Biol.* 16, e1008386. doi: 10.1371/journal.pcbi.1008386

Dura-Bernal, S., Suter, B. A., Gleeson, P., Cantarelli, M., Quintana, A., Rodriguez, F., et al. (2019). Netpyne, a tool for data-driven multiscale modeling of brain circuits. *Elife* 8, e44494. doi: 10.7554/eLife.44494

Feldotto, B., Eppler, J. M., Jimenez-Romero, C., Bignamini, C., Gutierrez, C. E., Albanese, U., et al. (2022). Deploying and optimizing embodied simulations of large-scale spiking neural networks on hpc infrastructure. *Front. Neuroinform.* 16, 884180. doi: 10.3389/fninf.2022.884180

Girard, B., Lienard, J., Gutierrez, C. E., Delord, B., and Doya, K. (2020). A biologically constrained spiking neural network model of the primate basal

ganglia with overlapping pathways exhibits action selection. *Eur. J. Neurosci.* 53, 2254–2277. doi: 10.1111/ejn.14869

Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., et al. (2010). Neuroml: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput. Biol.* 6, e1000815. doi: 10.1371/journal.pcbi.1000815

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications With Python*. Sebastopol, CA: O'Reilly Media, Inc.

Gutierrez, C. E., Skibbe, H., Nakae, K., Tsukada, H., Lienard, J., Watakabe, A., et al. (2020). Optimization and validation of diffusion mri-based fiber tracking with neural tracer data as a reference. *Sci. Rep.* 10, 1–18. doi: 10.1038/s41598-020-78284-4

Hahne, J., Diaz, S., Patronis, A., Schenck, W., Peyser, A., Graber, S., et al. (2021). *Nest 3.0.* Zenodo. doi: 10.5281/zenodo.4739103

Hjorth, J., Blackwell, K. T., and Kotaleski, J. H. (2009). Gap junctions between striatal fast-spiking interneurons regulate spiking activity and synchronization as a function of cortical activity. *J. Neurosci.* 29, 5276–5286. doi: 10.1523/JNEUROSCI.6031-08.2009

Hjorth, J. J., Kozlov, A., Carannante, I., Nylén, J. F., Lindroos, R., Johansson, Y., et al. (2020). The microcircuits of striatum *in silico. Proc. Natl. Acad. Sci. U.S.A.* 117, 9554–9565. doi: 10.1073/pnas.2000671117

Johansson, Y., and Silberberg, G. (2020). The functional organization of cortical and thalamic inputs onto five types of striatal neurons is determined by source and target cell identities. *Cell Rep.* 30, 1178–1194. doi: 10.1016/j.celrep.2019.12.095

Kriener, B., Enger, H., Tetzlaff, T., Plesser, H. E., Gewaltig, M.-O., and Einevoll, G. T. (2014). Dynamics of self-sustained asynchronous-irregular activity in random networks of spiking neurons with strong synapses. *Front. Comput. Neurosci.* 8, 136. doi: 10.3389/fncom.2014.00136

Liénard, J., and Girard, B. (2014). A biologically constrained model of the whole basal ganglia addressing the paradoxes of connections and selection. *J. Comput. Neurosci.* 36, 445–468. doi: 10.1007/s10827-013-0476-2

Malone, T. W. (2018). *Superminds: The Surprising Power of People and Computers Thinking Together*. Little: Brown Spark.

Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., et al. (2011). Introducing the human brain project. *Procedia Comput. Sci.* 7:39–42. doi: 10.1016/j.procs.2011.12.015

Meyer, R., and Obermayer, K. (2016). pypet: a python toolkit for data management of parameter explorations. *Front. Neuroinform.* 10, 38. doi: 10.3389/fninf.2016.00038

Okano, H., Sasaki, E., Yamamori, T., Iriki, A., Shimogori, T., Yamaguchi, Y., et al. (2016). Brain/minds: a japanese national brain project for marmoset neuroscience. *Neuron* 92, 582–590. doi: 10.1016/j.neuron.2016.10.018

Rajdl, K., Lansky, P., and Kostal, L. (2020). Fano factor: a potentially useful information. *Front. Comput. Neurosci.* 100, 569049. doi: 10.3389/fncom.2020.569049

Rosen, G. D., and Williams, R. W. (2001). Complex trait analysis of the mouse striatum: independent qtls modulate volume and neuron number. *BMC Neurosci.* 2, 1–12. doi: 10.1186/1471-2202-2-5

Rübel, O., Tritt, A., Ly, R., Dichter, B. K., Ghosh, S., Niu, L., et al. (2021). The neurodata without borders ecosystem for neurophysiological data science. *bioRxiv*. doi: 10.1101/2021.03.13.435173

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11, 682. doi: 10.3389/fnins.2017.00682

Sanz Leon, P., Knock, S. A., Woodman, M. M., Domide, L., Mersmann, J., McIntosh, A. R., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinform.* 7, 10. doi: 10.3389/fninf.2013.00010

Sato, M., Ishikawa, Y., Tomita, H., Kodama, Y., Odajima, T., Tsuji, M., et al. (2020). "Co-design for a64fx manycore processor and "fugaku"," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, GA: IEEE), 1–15.

Skibbe, H., Watakabe, A., Nakae, K., Gutierrez, C. E., Tsukada, H., Hata, J., et al. (2019). Marmonet: a pipeline for automated projection mapping of the common marmoset brain from whole-brain serial two-photon tomography. *arXiv preprint arXiv:1908.00876*. doi: 10.48550/arXiv.1908.00876

Spreizer, S., Senk, J., Rotter, S., Diesmann, M., and Weyers, B. (2021). Nest desktop, an educational application for neuroscience. *eNeuro* 8, ENEURO.0274-21.2021. doi: 10.1523/ENEURO.0274-21.2021

Stockton, D. B., and Santamaria, F. (2015). Neuromanager: a workflow analysis based simulation management engine for computational neuroscience. *Front. Neuroinform.* 9, 24. doi: 10.3389/fninf.2015.00024

Teeters, J. L., Godfrey, K., Young, R., Dang, C., Friedsam, C., Wark, B., et al. (2015). Neurodata without borders: creating a common data format for neurophysiology. *Neuron* 88, 629–634. doi: 10.1016/j.neuron.2015.10.025

Tripathy, S. J., Savitskaya, J., Burton, S. D., Urban, N. N., and Gerkin, R. C. (2014). Neuroelectro: a window to the world's neuron electrophysiology data. *Front. Neuroinform.* 8, 40. doi: 10.3389/fninf.2014.00040

Van Albada, S. J., Helias, M., and Diesmann, M. (2015). Scalability of asynchronous networks is limited by one-to-one mapping between effective connectivity and correlations. *PLoS Comput. Biol.* 11, e1004490. doi: 10.1371/journal.pcbi.1004490

Watakabe, A., Skibbe, H., Nakae, K., Abe, H., Ichinohe, N., Wang, J., et al. (2021). Connectional architecture of the prefrontal cortex in the marmoset brain. *bioRxiv*. doi: 10.1101/2021.12.26.474213

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Advantages of publishing in Frontiers

OPEN ACCESS
Articles are free to read for greatest visibility and readership

FAST PUBLICATION
Around 90 days from submission to decision

HIGH QUALITY PEER-REVIEW
Rigorous, collaborative, and constructive peer-review

TRANSPARENT PEER-REVIEW
Editors and reviewers acknowledged by name on published articles

Frontiers
Avenue du Tribunal-Fédéral 34
1005 Lausanne | Switzerland

Visit us: www.frontiersin.org
Contact us: frontiersin.org/about/contact

REPRODUCIBILITY OF RESEARCH
Support open data and methods to enhance research reproducibility

DIGITAL PUBLISHING
Articles designed for optimal readership across devices

FOLLOW US
@frontiersin

IMPACT METRICS
Advanced article metrics track visibility across digital media

EXTENSIVE PROMOTION
Marketing and promotion of impactful research

LOOP RESEARCH NETWORK
Our network increases your article's readership