BrainGenix-Neuron Emulation System (BG-NES): A Novel Neuron Simulation Architecture for Whole Brain Emulation

*Zergham Ahmed [*1], Thomas Liao[2], Prishita Ray[3], Randal Koene[4], Alicia J. Smallwood[5]*

Carboncopies Foundation Inc., 2443 Fillmore ST., #380-6190, San Francisco, CA 94115, United States

Abstract

Whole Brain Emulation (WBE), the endeavor to reproduce the full functionality and behavior of the human brain, has started to gain momentum in the scientific community. There have been attempts to define the problem concretely through roadmaps. Additionally, recent developments in technology related to brain tissue preservation, sample preparation, microscopy, the characterization of functional dynamics, and models of biological neuronal circuits have emerged. However, no computational platform is poised to support an emulated brain. It does not seem feasible to extend existing platforms to accommodate WBE due to problems with scalability. To address this gap, we introduce the BrainGenix-Neuron Emulation System (BG-NES), a proposed neuron simulation architecture suitable for WBE that offers the flexibility, scalability, and fault tolerance it requires. The purpose of BrainGenix-NES is to run emulations based on collections of simulations of neural activity and connectomic structure. BrainGenix-NES is part of a unified platform supporting whole brain emulation, within which it is combined with scan translation and environment rendering systems.

Keywords: Whole Brain Emulation, Computational Neuroscience, Simulation Architecture, High Performance Computing, Software Development

1. Introduction

The process of Whole Brain Emulation (WBE) intends to reproduce the function and behavior of a brain with enough fidelity that it reproduces its emergent mental experience [1]. Although there have been recent developments to address particular challenges to WBE such as brain preservation and translation, there is currently no simulation platform that is able to support the needs of WBE. Therefore, we propose the BrainGenix-Neuron Emulation System (BG-NES), a scalable high-performance, fault tolerant, distributed biological neuron simulation platform designed for WBE.

In our vernacular, we clearly distinguish the terms emulation and simulation. An emulation seeks to reproduce a desired behavior of a system by simulating its features. It does not need to replicate every feature of the system but only those necessary for the desired experience to be had. Furthermore, simulation, however, reproduces the fundamental, underlying principles of a system. Simulating something reproduces the functionality of the system with an understanding of the internal workings. Emulating something reproduces the internal mechanisms at a given level of abstraction, also known as the scale. In WBE, the brain is *emulated* because the goal is to reproduce the expected experience generated by the brain, whilst we simulate the neurons with a mathematical abstraction. A Hodgkin-Huxley model [2] would be an example of a simulation of action potentials. We assume that these simulations may be employed on computer hardware and do not require biological neurons. Thus, the Hodgkin-Huxley model provides a simulation of action potentials and subthreshold activity in neurons. A computational emulation relies on one or more simulated processes. Emulation often strives to achieve a desired experience. Simulation is the constrained implementation of a model process.

[1] * Corresponding author: zahmed@carboncopies.org
[2] tliao@carboncopies.org
[3] pray@carboncopies.org
[4] rkoene@carboncopies.org
[5] asmallwood@carboncopies.org

Designed to support WBE, BG-NES will be a simulation platform that facilitates the implementation of mathematical models that describe specific neural dynamics. Its primary goal is to support an emulation that reproduces the experience that arises from whole brain behavior, and to do so using simulated neurons and synapses.

BrainGenix-NES aims to be an exploratory research tool as well as a platform to run a full-scale WBE for a human brain. Though simulation platforms are not new, current platforms are not intended to address the scalability, fault tolerance and flexibility of scale to handle large, coordinated simulations necessary for WBE. BrainGenix is made up of three different modules: the Neuron Emulation System (NES), the Environment Rendering System (ERS), and the Scan Translation System (STS) (Figure 1). This paper focuses on the Neuron Emulation System (NES) module, however we will briefly introduce the ERS and STS modules in section 5.

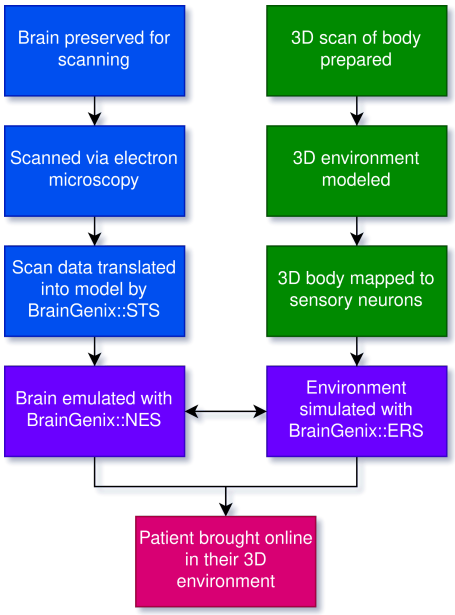

Figure 1: **Overview of process used to emulate a human brain with BrainGenix.** BrainGenix uses its scan translation system (STS) to process scan data into neural models. It uses its environment rendering system (ERS) to construct a 3D virtual environment to host emulation and it's neural emulation system (NES) to run the emulation.

2. Statement of Need and Design

Whole Brain Emulation poses many computational, neurobiological and engineering challenges, such as scalability, the potential loss of compute nodes, neural complexity, and time scaling [3]. To our knowledge, there are no existing software platforms aimed at running a WBE and addressing the challenges associated with that. The BrainGenix system was designed to address this gap in available software by taking into account scale separation, scalability, and fault tolerance necessary for WBE.

Scale separation. One key assumption of WBE is scale separation, which is the idea that a system can be replicated at a certain level of detail (ie, spiking neural networks) but not have to simulate every level of detail (ie, stochastic behavior of molecules) to be able to satisfy a specific set of success criteria. Though all computational models implicitly assume some form of abstraction and scale separation, we chose not to favor one scale of abstraction and thus offer flexibility for model definition. Therefore, the BrainGenix-NES simulation backend is based on the Brian2 simulator [4], which allows models to be defined based on the differential equations inputted by the user. BrainGenix-NES, by using Brian2, supports many biological neuron models of varying complexity and detail. Since BrainGenix-NES operates by manipulation of differential equations, any model written as a differential equation may be added. BrainGenix-NES aims to be easily extensible to incorporate any model or neural dynamic seamlessly without having to code additional modules, a problem with existing simulation platforms.

Scalability. Since scalability is a main focus of BG-NES, the platform allows for simulation of billions of neurons. There are many computations affecting the probability and frequency of neural firing and these computations differ among the types of neurons. The

human brain is composed of billions of neurons so any platform used for WBE must be scalable to achieve this order of magnitude. BrainGenix-NES does this by utilizing a massively parallel design, accommodating use of thousands of machines. BrainGenix-NES utilizes Zookeeper and Kafka to achieve this. Considering the uncertainty surrounding scale separation in the field of WBE,

Fault tolerance. Fault tolerance is crucial to maintaining emulation integrity in the event of errors or server loss. As more compute nodes are used, the number of failure points and chances of failure increase as well. Since a full-scale WBE would most likely require tens of thousands of nodes, BG-NES utilizes production-ready fault tolerant systems and other algorithms unique to the platform to keep the emulation intact.

The core functionality of BrainGenix-NES can be divided into its *Main Simulation Engine*, *Simulation Optimization*, *Simulation Editor*, and *APIs*.

2.1 Main Simulation Engine-

The primary backend feature of BrainGenix-NES is its Main Simulation Engine (MSE). It interfaces with the Brian2 simulator to run and store simulations as well as provide security by managing user permissions.

The MSE is distributed across several nodes and therefore relies on Apache Zookeeper for node coordination as well as Apache Kafka for node communication. Finally, the MSE utilizes a SQL database for simulation storage along with a comprehensive logging system (Figure 2).

2.2 Simulation Optimization

The simulation optimization algorithm improves the simulation efficiency through its proposed Load Distribution Heuristic (LDH), fault tolerance, and optimization of computation and network load. More detail about these improvements will be discussed in section 3.

2.3 Simulation Editor

The simulation editor provides tools to generate neurons and synapses according to parameters set by the user. A user defined list of parameters are captured and stored for later examination. Additionally, snapshot functionality exists to capture any given state of a simulation. A snapshot saves the current state of the simulation so that it can be restored to that point in time later. A Neuron Model Testing Environment (NMTE) is included which provides a sandbox for researchers to test chosen models before running large-scale simulations. NMTE uses a single Brian2 instance, whereas the MSE distributes the simulation across multiple Brian2 instances.

2.4 APIs

BrainGenix utilizes two types of Application Programming Interface (API). The management API allows external programs as well as the Graphical User Interface (GUI) and Command Line Interface (CLI) to run all administrative tasks. Additionally, the simulation API provides an interface for external systems to interact with NES simulations. Further, this API uses the Apache Kafka protocol to handle requests.

An example workflow is provided in Figure 2.

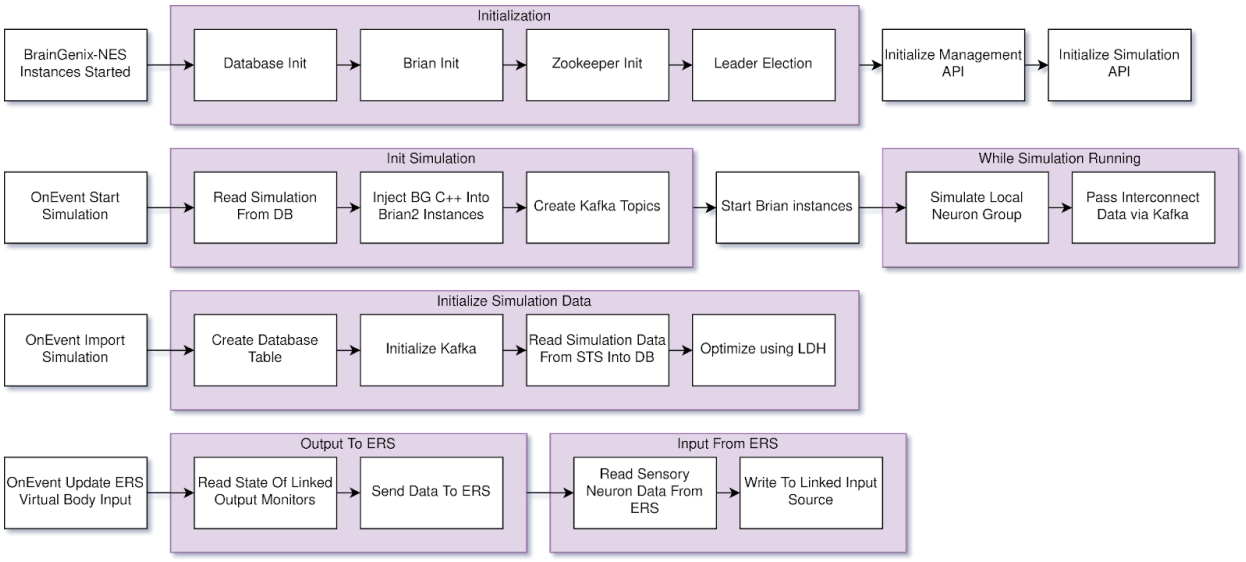

Figure 2: **Block diagram of sample internal workflow with the BrainGenix system.** These three examples illustrate the backend functionalities of BrainGenix-NES.

3. Features

Work is ongoing for this project. For the most up to date information please visit our technical specifications document: [BrainGenix-NES Technical Specifications.]

3.1 Node Coordination System

Apache Zookeeper [8] provides a fast, fault-tolerant, and simple interface for BrainGenix-NES node coordination and is capable of handling potentially thousands of instances of follower node plugins.. Zookeeper manages the election system, a system where when the elected leader node fails, an elected follower node will assume leadership. This allows for fault-tolerant node coordination.

3.2 Load Distribution Heuristic

There are two algorithms in place that allow for BrainGenix-NES to optimize its simulation performance.

The Load Distribution Heuristic is used to optimize the network and compute usages across the entire cluster. Initially, the BrainGenix-NES system assigns neuron indices by sequential blocks. It then balances the load of the compute durations to prevent an excess of neurons on any given node. This allows for a system with non-uniform nodes (i.e. a Raspberry Pi and PowerEdge) to work in the same cluster efficiently.

The network optimization aspect is achieved by first calculating the percentage of “local” and “global” synapses. Local synapses connect neurons simulated by their own node. Whereas, global synapses connect neurons across different nodes. After calculating the percentage of both types of synapses, the heuristic calculates the percentage of global synapses per node, which locates the region where each neuron has the most synapses. A Neuron Query Algorithm (NQA) is used when calculating this percentage, which avoids overloading the database server. Once the percentage breakdown for all neurons and their synapses is calculated, the node with the highest percentage of synapse connections for a given neuron is assigned that neuron. This process of assigning neurons to nodes reduces and balances the network load. It should be noted that the system exclusively exchanges neurons, not offloading or changing the total number of neurons between the system. Ultimately, this allows for the compute optimization to be maintained more efficiently.

The compute aspect of optimization involves distributing the number of neurons efficiently across the BrainGenix-NES cluster by timing the compute duration. This information is reported back to the leader, and an average is calculated from the data of the entire cluster. Outliers are selected from a user defined margin, and neurons are removed or added to that node to balance its load. To ensure

accurate times, nodes are run for a number of times defined by the user, and the per-node compute time is calculated from the average of the runs.

3.3 Fault Tolerance

It is crucial that the integrity of the simulation is maintained. A node failure will result in the loss of neurons simulated by that node.

Load from Last Checkpoint is used as the default fault tolerance strategy for BrainGenix-NES. In the event of a failure, the simulation would be suspended, purged from RAM, and reloaded from the last checkpoint. The alternative mechanism of fault tolerance is to have a number of spare nodes for the simulation on standby. One potential solution to this time-reloading problem would be to have a number of spare nodes for the simulation on standby. The downside of this is that the cluster would require a large amount of available bandwidth and a large number of spare, idle servers. Nevertheless, this may be the best solution for a research environment, as it would protect simulation integrity while minimizing the cost to run a cluster. It may also be beneficial to a production environment as a severe failover state, in the event that more nodes fail after a triggered SoF.

3.4 Neuron and Synapse Generation

User defined neurons are written to the BrainGenix-NES SQL database (Figure 3). Follower nodes will compare data with the database to ensure the working data is up to date. The simulation may then be reoptimized depending on the number of neurons added.

The Synapse Generator is similarly used to add synapses into a separate database table, according to the parameters specified by the user. Users are able to manipulate the following (nonexhaustive list):

- Range from existing neuron
- Offset for distal neurons
- Connection ranges for neurons with multiple axons
- Random delays or effects on connection generation behavior in a region
- Connection area “shape” using a 3D mesh such as cone, sphere, etc.

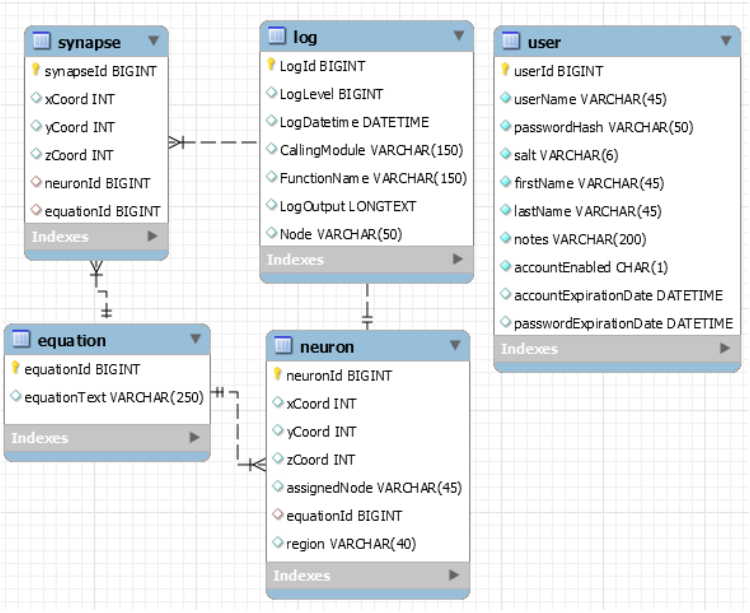

Figure 3: **BrainGenix-NES database overview.** Database storage for elements used by simulation

3.5 Neuron Model Testing Environment

The NMTE provides researchers with a tool to verify the performance of their models before running large-scale simulations. The NMTE uses the Brian2 simulation engine, however it directly uses a single Brian2 instance and does not rely on BrainGenix-NES

code to distribute and optimize the simulation. The NMTE is designed to run a few hundred to a few thousand neurons and is used for model verification. It provides a real-time 3D interactive simulator as well as graphical output for the data.

Another useful function for this testing environment is to create emulations smaller than human whole brain emulations e.g. drosophila.

3.6 Sources

Sources provide different methods of input into neurons in BrainGenix-NES. Direct Equation Sources allow the user to provide inputs to connected neurons in the form of defined equations. Uniform Region Sources provide stochastic noise as input to regions of neurons. It is used to understand how manipulating a particular layer of the brain affects the behavior of the entire brain. Direct Input Sources feed input, from sources external to BrainGenix-NES via the simulation API, to their connected neurons. An application of this would be the connection between a BrainGenix-NES emulation and virtual body simulator, in which input is provided to the emulation from the virtual body simulator with the direct input source serving as an intermediary. Global sources allow for written effects to be applied to all neurons throughout the entire simulation. This enables users to create background noise patterns emulating real-world conditions without the need to simulate the entire brain. An example illustrating this would be the simulation of theta rhythms by using an appropriate frequency sine wave.

3.7 Monitors

The Direct Output Monitor is used to connect the output from specific neurons to external systems. This is useful for virtual body applications such as connecting a muscle to BrainGenix-NES.

The Field Output Monitor measures the sum of a group of neurons. It enables measurement of neural oscillations from parts of the neural network. These monitors are accessible via the API.

3.8 CLI and GUI

The BrainGenix-NES command-line interface (CLI) and graphical user interface (GUI) simplifies management and usage of the system. Most existing simulators lack any form of GUI. The GUI is projected to have a user login page along with ACL, cluster overview with graphs, 3D interactive neuron editor and viewer, and ability to select neurons and view their voltage histories as well as other configurable parameters.

3.9 Management API

The BrainGenix-NES Management API allows external programs to interact with the system. The Management API is based around the FastAPI Python framework [9], as well as Uvicorn [10], which emphasizes performance and is regarded to be on par with NodeJS and Go. The Management API is responsible for the interaction between the BrainGenix-NES system and the GUI/Web interface and CLI. Interactions from a client are sent through an intermediate API server which presents a unified interface across BrainGenix-NES. Authentication is handled during the connection progress. A token is provided by the server after.

All password encryption and other cryptographic functions are handled by external modules that are maintained by larger organizations whenever possible. This is done to prevent BrainGenix-NES from lagging behind the industry standard cybersecurity.

3.10 Extensibility

BrainGenix may be extended by connecting external systems to the API. To provide more management features, external systems should use the management API, whereas applications looking to interact directly with the simulation should make use of the Apache Kafka based API. Alternatively, most aspects of the system are available from the Python code, and can be easily edited and extended via modification of that.

3.11 Development and Availability

Currently, BrainGenix is still in development at https://github.com/carboncopies/BrainGenix-NES.
BrainGenix is licensed under the GNU Affero General Public License v3.0 license. This is done to establish an emulation environment that will be completely open source, ensuring the emulation's sustainability. Our working documentation and technical specifications document linked in the repository give further details on the implementation. Contributors are welcome to the project.

4. Related Work

It might be more efficient to build off current platforms rather than starting from scratch, however, different neuron simulation platforms are designed with varying goals and purposes. Many of these purposes such as flexibility, performance, features, level of model detail have trade offs among each other. The question is whether the design and trade-offs chosen for other platforms pose any barriers to extension to support the scale separation, scalability and fault tolerance necessary for WBE.

Flexibility

Modeling platforms seem to have bias in focusing on specific models even though they all offer extendability for study of brain networks and individual neurons. For example, NEST [5] focuses on modeling large networks with simple individual neurons, while NEURON [6] and GENESIS [7] are more suited for biologically detailed models including reconstruction of spatial-temporal integration in non-linear dendrites without the scale [8].

Brian2 was incorporated in the proposed approach due to its flexibility in defining models and ease-of-use for neuroscientists. As mentioned previously, flexibility is important for WBE due to the uncertainty surrounding scale separation. The ability of Brian2 that allows for users to specify connectivity patterns through mathematical equations [4] will allow for ease-of-use when working with the massive interconnectivity required for WBE. Furthermore, other major simulator platforms use additional languages outside of Python (i.e. domain language NMODL used by NEURON to describe ionic channel properties) [4]. However, these additional languages are useful for describing certain biological features which become more difficult with the same Python code. For this reason, Brian2, as well as BrainGenix, aims to allow implementation of these biological features with differential equations.

Scale

PGENESIS, an improvement of GENESIS more suited for parallelism, shows the most promise for scalability as it is a biologically detailed simulator that is able to handle network sizes as large as 9×10^6 neurons with 18×10^9 synapses as well as 2.2×10^6 neurons with 45×10^9 synapses [7]. Yet, even this impressive scale is not sufficient for WBE.

One major issue with Brian2 is that it does not have support for large networks, supercomputers, and high-performance clusters [4] needed for the scale of WBE. Our proposed distributed system based on Apache Zookeeper addresses this issue. The trade-off is that support for biologically detailed, multi-compartmental models is currently limited in Brian2 when compared to other platforms such as NEURON and PGENESIS, in which they are well supported [4]. However, we have shown that PGENESIS has issues with scale and scalability relative to WBE and further, NEURON has not yet demonstrated the necessary scalability in the context of the Blue Brain Project [6]. BrainGenix NES proposes a compromise by extending the support for multi-compartmental models while also using a distributed system for scalability.

Fault Tolerance

None of the platforms mentioned so far address and test fault tolerance. BrainGenix-NES includes a fault tolerance mechanism for loading the last saved checkpoint of an emulation state as well as running with the fault if desired. If the total number of failed neurons is less than a threshold value specified by the user, the emulation can continue without suspending the remaining neurons until BrainGenix-NES allocates another node in the cluster to assign them to. If the threshold value is crossed, then the simulation is set to suspend.

5. Conclusion

BrainGenix-NES will be a novel platform that addresses the simulation challenges of WBE and provides a flexible framework for simulating neurons. To our knowledge, BrainGenix is the only simulator that addresses scale separability, scalability and fault tolerance to work with massive connectivity. There are many future directions for improving its design. Potential improvements could be made by implementing a backend built from the ground up with the goal of distributed simulation. Additionally, performance could be improved by implementing GPU acceleration. Once implemented, we plan to benchmark the performance of BrainGenix-NES compared to other simulators.

CRediT authorship contribution statement

Zergham Ahmed: Conceptualization, Writing - Original Draft, Project Administration, Visualization**, Prishita Ray:** Software, Writing - Review & Editing**, Alicia Smallwood:** Writing - Review & Editing, Supervision, Project administration, Resources**, Randal Koene:** Writing - Review & Editing, Supervision, **Thomas Liao:** Conceptualization, Software, Writing - Original Draft, Supervision, Project administration, Visualization, Resources

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This work received no financial support.

Data and code availability

Data, code, and documentation are available at the following URL:
https://github.com/carboncopies/BrainGenix-NES

Acknowledgements

The authors of this paper would like to acknowledge administrative support from the Carboncopies Foundation. We thank Teri O'Donnell for concept inspiration, Tyson Ruszler for debugging assistance and consultation, Brad Leu for creating the SQL Database Diagram, Arseniy Mukovozov for software development and editing, Sean Solomon and Joshua French for code review.

References

[1] N. Bostrom, A. Sandberg, Whole Brain Emulation: A Roadmap, (n.d.) 130.
[2] A.L. Hodgkin, A.F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol. 117 (1952) 500–544. https://doi.org/10.1113/jphysiol.1952.sp004764.
[3] R. Cattell, A. Parker, Challenges for Brain Emulation: Why is Building a Brain so Difficult?, Nat. Intell. INNS Mag. 1 (2012) 17–31.
[4] M. Stimberg, R. Brette, D.F. Goodman, Brian 2, an intuitive and efficient neural simulator, ELife. 8 (n.d.). https://doi.org/10.7554/eLife.47314.
[5] M.-O. Gewaltig, M. Diesmann, NEST (NEural Simulation Tool), Scholarpedia. 2 (2007) 1430. https://doi.org/10.4249/scholarpedia.1430.
[6] P. Kumbhar, M. Hines, J. Fouriaux, A. Ovcharenko, J. King, F. Delalondre, F. Schürmann, CoreNEURON : An Optimized Compute Engine for the NEURON Simulator, Front. Neuroinformatics. 13 (2019). https://www.frontiersin.org/article/10.3389/fninf.2019.00063 (accessed January 23, 2022).
[7] J.C. Crone, M.M. Vindiola, A.B. Yu, D.L. Boothe, D. Beeman, K.S. Oie, P.J. Franaszczuk, Enabling Large-Scale Simulations With the GENESIS Neuronal Simulator, Front. Neuroinformatics. 13 (2019). https://doi.org/10.3389/fninf.2019.00069.
[8] R.A. Tikidji-Hamburyan, V. Narayana, Z. Bozkus, T.A. El-Ghazawi, Software for Brain Network Simulations: A Comparative Study, Front. Neuroinformatics. 11 (2017). https://doi.org/10.3389/fninf.2017.00046.
[9] Ramírez, S. (2021). FastAPI. Retrieved from https://github.com/tiangolo/fastapi
[10] Encode. (2021). Uvicorn. Retrieved from https://github.com/encode/uvicorn