Title: Predicting synaptic connectivity for large-scale microcircuit simulations using *Snudda*

Authors: J J Johannes Hjorth[1, *], Jeanette Hellgren Kotaleski[1, 2], Alexander Kozlov[1, 2]

[1] Science for Life Laboratory, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, SE-10044 Stockholm, Sweden;

[2] Department of Neuroscience, Karolinska Institute, SE-17165 Stockholm, Sweden

* Corresponding author: Johannes Hjorth <hjorth@kth.se>

ORCID: 0000-0002-0550-0739 (JHK); 0000-0002-9302-0750 (JH); 0000-0003-3994-0799 (AK)

Abstract

Simulation of large-scale networks of neurons is an important approach to understanding and interpreting experimental data from healthy and diseased brains. Owing to the rapid development of simulation software and the accumulation of quantitative data of different neuronal types, it is possible to predict both computational and dynamical properties of local microcircuits in a 'bottom-up' manner. Simulated data from these models can be compared with experiments and 'top-down' modelling approaches, successively bridging the scales. Here we describe an open source pipeline, using the software *Snudda*, for predicting microcircuit connectivity and for setting up simulations using the NEURON simulation environment in a reproducible way. We also illustrate how to further 'curate' data on single neuron morphologies acquired from public databases. This model building pipeline was used to set up a first version of a full-scale cellular level model of mouse dorsal striatum. Model components from that work are here used to illustrate the different steps that are needed when modelling subcortical nuclei, such as the basal ganglia.

Keywords: large-scale simulations, striatum, basal ganglia, brain microcircuits, synaptic connectivity, model building pipeline

Acknowledgements: The simulations were performed on resources provided by the Swedish National Infrastructure for Computing at PDC (Center for Parallel Computing). 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 wish to thank Johanna Frost-Nylén, Robert Lindroos and Ilaria Carannante for helpful discussions. We also thank Robin de Schepper, Kadri Pajo, and Wilhelm Thunberg for help with software compatibility.

Declarations

Funding Horizon 2020 Framework Programme (785907, HBP SGA2); Horizon 2020 Framework Programme (945539, HBP SGA3); Vetenskapsrådet (VR-M-2017-02806, VR-M-2020-01652); Swedish e-science Research Center (SeRC); KTH Digital Futures.

Conflicts of interest/Competing interests The authors have no relevant financial or non-financial interests to disclose.

Availability of data and material Data used for single-cell neuron models as well as synaptic connectivity within the example network of the striatal microcircuit is available on GitHub (in the snudda/data and snudda/examples folders at https://github.com/hjorthmedh/Snudda).

Code availability (see Information Sharing Statement)

Information Sharing Statement The presented software *Snudda* (version XXX, DOI:XXX) and *treem* (version YYY, DOI:YYY) are available on GitHub and PyPI:

- Snudda - https://github.com/Hjorthmedh/Snudda, https://pypi.org/project/snudda/
- treem - https://github.com/a1eko/treem, https://pypi.org/project/treem/

Introduction

Neuroscientists are producing data at an ever growing rate, and sharing the data in public databases. Within the computational neuroscience field, hypothesis-driven modelling has over many decades generated new ideas that in turn have been tested via experiments. Recently a data-driven mechanistic modelling approach has also gained ground thanks to new technologies allowing the collection of large quantities of useful data. In particular, large-scale spiking neural network models have been reconstructed in a data-driven manner and simulated (Markram et al. 2015; Einvoll et al. 2019; Casali et al. 2019; Migliore et al. 2018; Hjorth et al. 2020). Collecting data from the brain at multiple biological scales from mouse, non-human primates, and human, are important goals of several of the big brain initiatives (Insel et al. 2013; Amunts et al. 2019; Okano et al. 2015; Grillner et al. 2016), and will further facilitate and speed up this modelling process. In parallel, various brain simulation tools have been optimized to capitalize on supercomputers (Hepburn et al, 2012; Plesser et al. 2015; Carnevale and Hines, 2006; Hines et al. 2009, Kumbhar et al. 2019; Ray and Bhalla, 2008; Gleeson et al. 2010; Jordan et al. 2020; Akar et al. 2019; Billeh et al, 2020). In this respect, the principles of FAIR – Findable, Accessible, Interoperable, Reusable (Wilkinson et al. 2016) – are important for catalysing this process, both with regard to the experimental data, the data-driven models as well as the software used during the modelling and simulation process. We also believe that to be able to reproduce the actual model reconstruction process, given the same or new additional data, is one important aspect of the FAIR criteria when making the modelling process transparent, repeatable, reusable and comparable.

Here we present our open source modelling pipeline that facilitates a reproducible, data-driven reconstruction of cellular level network/microcircuit models. This pipeline inspired by the cortical column microcircuit (Markram et al. 2015) has been applied to predict a full-scale microcircuit model of the mouse dorsal striatum (Hjorth et al.2020). *Snudda* is a software to create a detailed network of connected neurons, where the connectivity is derived from reconstructed neuronal morphologies as well as from more qualitative experimental knowledge (see also Reimann et al. 2015). ‘Snudda’ means ‘touch’ in Swedish, and it supports the creation of a network with connectivity based on touch detection. If detailed morphological data exist, the algorithm looks for close appositions between axons and dendrites, which are locations for putative synapses. Thus the morphology restricts where connections can be positioned. Snudda can also define the axon using a probability cloud if a reconstructed axon is missing. This is an extension of the method where the connection probability is proportional to the overlap of two spheres representing axons and dendrites (Humphries et al. 2009). Based on a set of rules, as described below, the putative synapses are then pruned to match the connectivity seen from pairwise experimental recordings, or other types of connectivity experiments. The same technique can be applied to also place gap junctions. The generated network can then be simulated using parallel NEURON (Carnevale and Hines 2006). Similar approaches have been used to build the somato-sensory cortex microcircuit (Markram et al. 2015; Colangelo et al. 2019), visual cortex model (Billeh et al. 2020), cerebellar network (Wichert et al. 2012; Sudhakar et al 2017; Casali et al. 2019) and hippocampal neurons (Migliore et al. 2018).

The reconstruction of a local microcircuit model (such as striatum) consists of the following

steps: a) experimental data acquisition of the electrophysiological and morphological properties of neuronal types, and also characterisation of synapses, b) optimization of neuron and synapse models, c) placement of the model neurons in the brain volume to be modelled, d) prediction of microcircuit connectivity *in silico*, e) constraining and emulating inputs for the model, and finally f) simulating the microcircuitry. Our software Snudda is used for steps c)-f). Software for improving the morphological reconstructions in preparatory step b) is described at the end. The code is publicly available on GitHub (https://github.com/Hjorthmedh/Snudda/) and (https://github.com/a1eko/treem). Below we will go through the different steps, and provide code to set up an example network using Snudda. The network example corresponds to a 0.5 mm cube within the mouse striatum. We will assume that we already have a set of electrophysiologically optimized neurons and synapses, e.g. using the optimization tool BluePyOpt (Van Geit et al. 2016). Examples of neuron and synapse models relevant for striatum are provided on GitHub (in the `Snudda/snudda/examples` folder with scripts and notebooks).

Getting started with *Snudda*

Snudda is available for download from https://github.com/hjorthmedh/Snudda or through `pip3 install snudda`. Both source code and the data files necessary to set up a striatal network are provided, Table 1 provides an overview of the *Snudda* directory structure. *Snudda* is compatible with Linux, Mac and Windows 10.

snudda	snudda directory, code usually executed from this folder
snudda/data	snudda data folder
snudda/data/mesh	3D meshes for brain structures
snudda/data/neurons/striatum	morphology and parameters for striatal neurons in subfolders
snudda/data/neurons/mechanisms	NEURON mechanisms folder with mod-files
snudda/data/synapses	synapse model parameters
snudda/data/input_config	input configuration files
snudda/input_tuning	scripts for synaptic input tuning
snudda/plotting	scripts for plotting networks, includes subfolder with blender scripts
snudda/utils	scripts for small tasks
examples	examples on how to run snudda, contains shell scripts and notebooks
tests	contains unit and regression tests
tests/networks	networks used or created for testing
tests/validation	morphologies and mechanisms used for testing

Table 1. Snudda directory structure.

In the directory `snudda/data/neurons` there are separate subdirectories for each neuron type. Each of those directories contains multiple subdirectories, one for each unique morphology from that neuron type. The neuron directories include the morphology in SWC format, a JSON parameter file with one or more sets of optimised neuron parameters from BluePyOpt, a JSON mechanism file specifying which mechanisms are present in each compartment, and a JSON modulation file which specifies the neuron modulation of the neuron. The JSON file format was chosen as it is a standardised and human readable way to store structured data. The `neurons` folder also has a mechanism folder containing the NEURON model description language `.mod` files with NEURON mechanisms for ion channels.

To keep the networks separate, each generated network has its own directory which contains a `network.json` file that links together all the different components that make up the network. The `network.json` file can be manually created, or in the case of the striatal network there is a way to automatically generate a `network.json` file of user specified size. The script `init.py` can be extended to create networks of other brain structures. A Jupyter Notebook in `examples/notebooks` shows an alternative example for how to define brain slices and other structures.

In the main `Snudda` directory there is an example folder with useful scripts and Jupyter Notebooks for generating and running networks. The directory `snudda/plotting` contains scripts to plot simulation results as well as visualise the network or parts of it using Blender.

Creating and running an example network

First we create an example striatal network, then further down we go through all configuration details. The `network-config.json` can be generated using the `snudda init` shell command. In the below example a homogeneous cube with 0.5 mm side length in the mouse striatum is generated (**Figure 1A**). This corresponds to 10,062 neurons (**Figure 1B**) using the estimated average density of striatal neurons (Rosen and Williams, 2001), but the number of neurons can be varied depending on computational resources and research question. The neurons are taken from the `data/neurons` directory, where every neuron type has its own directory, e.g. `dspn`. A neuron type is represented by one or more single-cell models, each in its own subdirectory with a unique morphology and optimized parameter set. When the network is initialised, the `init.py` code will look in the folders of the different neuron types and instantiate single-cell models in random order. No modifications of the neuron models other than rotations are applied at runtime. In order to improve the cell diversity, one should populate the neuron types directories under `data/neurons` with sufficient number of different single-cell models.
The following commands can be used in a terminal window to generate the example network that we have used for the figures in this article. We go into more detail in the sections later in the article. There are additional Jupyter Notebook examples in examples/notebooks. The first step is to create a configuration file `network.json` specifying e.g. 10,062 neurons in a directory called `smallSim`.

```
simName=smallSim
snudda init $simName --size 10062 --overwrite
```

Next we need to place the neurons in a specified volume. For large simulations the neurons are placed inside a volume representing the mouse striatum (**Figure 1A**), while smaller networks use a simple cube (**Figures 1B,C**). This is done to preserve physiological neuron densities in the simulations. The mesh definition of the striatal volume, or other structures, can be extracted from databases such as the Allen Brain Atlas of the mouse brain. The command to place the neurons inside the volume defined by the mesh is:

```
snudda place $simName
```

The next step is the touch detection (**Figure 2A**). Here the algorithm voxelizes the space, and looks for overlaps (within a certain predefined distance) between axons and dendrites from different neurons (**Figure 3A,B**) (Hellwig, 2000).

```
snudda detect $simName --volumeID Striatum
```

The touch detection will create a putative set of synapses at all the close appositions between axons and dendrites. However, not all close appositions correspond to real synapses, as explained in detail further down. The next step prunes the set of putative synapses to match the connectivity seen in experimental pairwise recordings (**Figure 2B**, distance dependent connectivity). The rules used for pruning are similar to what Markram et al. (2015) created for their cortical network. The parameters for the pruning (**Figure 3C-G**) are specified in the `network.json` file, explained more in detail below. The command to perform the pruning is:

```
snudda prune $simName
```

To generate figures analysing the connectivity (**Figure 2C,D**), e.g. distance dependent connection probability, histogram showing the number of synapses between connected neighbours, histogram showing the number of connected neighbours. See `snudda/analyse_striatum.py`, also examples/notebooks for Jupyter notebooks.

Next we need to generate synaptic input for the network simulation. Here we specify how much time we want to generate inputs for. The parameters for the synaptic input from cortex and thalamus are defined in a separate JSON file:

```
cp -a data/input_config/input-tinytest-v9-freq-vectors.json
$simName/input.json
snudda input $simName --input $simName/input.json --time 3.5
```

The last step involves compiling the `.mod` files, and then running the simulation.

```
nrnivmodl data/neurons/mechanisms
snudda simulate $simName --time 3.5
```

There are two functions in the plotting directory that allow the user to plot either the spike raster or the voltage traces, `plotting/plot_spike_raster.py` (**Figure 4**) and `plotting/plot_traces.py`, the latter requires you to have run the simulation with the `--voltOut` parameter to also save the voltage traces. The simulation output files are stored in the `$simName/simulations` directory.

Snudda connectivity configuration explained

Running the command `snudda init mysimulation --size N` generates the Snudda network configuration file network-config.json where N is the size of the network to create. The `network-config.json` contains the blueprints for the striatal network hierarchically organised into *RandomSeed, Volume*, *Units*, *Connectivity* and *Neurons*. All parameters in the configuration file are specified in SI units. Below we go into more detail how Snudda is configured.

The *RandomSeed* block specifies the random seeds used for the different steps of the network creation.

```
"RandomSeed": {
  "init": 605423731,
  "place": 713568010,
  "detect": 3462736139,
  "prune": 2408118524,
  "input": 1253064917,
  "simulate": 3292847414
},
```

The *Volume* block can contain several named volumes, such as for example “Striatum”, “Cortex”, “Thalamus”. For each volume block we define *type* (e.g. “mesh”), *dMin* (minimum distance between somas in the volume), *meshFile* (this specifies the Wavefront OBJ file that defines the mesh enclosing the volume) and *meshBinWidth* (voxelization size of the mesh for determining what is inside and outside the mesh during cell placement). Future versions of Snudda will allow for density variations within the volume and directional gradients for the neurons.

```
"Volume": {
    "Striatum": {
        "type": "mesh",
        "dMin": 1.5e-05,
        "meshFile": "striatum.obj",
```

```
            "meshBinWidth": 5e-05
        }
    },
```

The next block is the optional *PopulationUnits* block, which allows us to define network heterogeneities within the volumes, these units may receive common input which is different from the input to other neurons. Snudda also supports different connectivity within and between these population units. These could mimic functional units that have been organised due to developmental processes or learning, e.g. a functional unit within the striatum could map to an 'action channel' (Gurney et al. 2001; Lindahl and Hellgren Kotaleski, 2016; Berthet et al. 2016). There are currently two methods supported "random" which randomly picks a fraction of the neurons for each population unit, or "radialDensity" where a density function placed in the volume determines the neurons in each population unit (see Snudda/examples/notebooks on GitHub).

An example block for "PopulationUnits" looks as follows. Here two units are defined: UnitID 1 with 20% of dSPN and iSPN neurons in Striatum, and UnitID 2 with 30%.

```
"PopulationUnits": {
        "AllUnitID": [1, 2],
        "Striatum": {
                "method": "random",
                "fractionOfNeurons": [0.2, 0.3],
                "unitID": [1, 2],
                "neuronTypes": [
                        ["dSPN", "iSPN"],
                        ["dSPN", "iSPN"]
                ],
                "structure": "Striatum"
        }
}
```

In the *Connectivity* block we define the rules guiding how the different neuron populations are connected together. Each connection pair has its own block (e.g. "iSPN,dSPN"). In the example below this is illustrated with the iSPN-dSPN pair (indirect-pathway and direct-pathway striatal projection neurons, respectively), which are connected by GABA synapses.

```
    "Connectivity": {
        "iSPN,dSPN": {
            "GABA": {
                "conductance": [
                    2.4e-10,
                    1e-10
                ],
                "channelParameters": {
                    "tau1": [
```

```
                        0.0013,
                        1000.0
                    ],
                    "tau2": [
                        0.0124,
                        1000.0
                    ],
                    "failRate": 0.4,
                    "parameterFile":
"$DATA/synapses/striatum/PlanertFitting-ID-tmgaba-fit.json",
                    "modFile": "tmGabaA"
                },
                "pruning": {
                    "f1": 0.3,
                    "softMax": 4,
                    "mu2": 2.4,
                    "a3": 1.0,
                    "distPruning": "1-exp(-(0.4*d/60e-6)**2)"
                },
                "pruningOther": {
                    "f1": 0.3,
                    "softMax": 4,
                    "mu2": 2.4,
                    "a3": 1.0,
                    "distPruning": "1-exp(-(0.4*d/60e-6)**2)"
                }
            }
        },
```

Each connection has a *conductance* parameter, which specifies the mean and standard deviation of the conductance. The *channelParameters* gives flexibility by specifying a dictionary with the channel specific parameters, in this case it is *tau1*, *tau2*, *failRate* (the synapse failure rate) that are passed directly to the NEURON channel model. Next *parameterFile* (JSON file with additional channel parameters) and *modFile* (NEURON channel .mod file). The final two blocks *pruning* and *pruningOther* specifies the pruning parameters for neurons within the same population unit, and neurons in different population units. The pruning parameters help shape the connectivity by parameterising the rules that define which putative connections should be removed, and which ones should be kept. The probability to keep a synapse is equal to the product of the individual pruning steps:

$P_{keep} = f1 * P_{mu} * P_{SM} * a3 * P_{dist}$

The *f1* parameter defines how large a fraction of the putative synapses should be kept, a value of 1.0 or None means that this pruning step is not used (**Figure 3C**).

The *mu2* defines a sigmoid curve used to decide whether to keep or remove all synapses between a coupled pair of neurons (**Figure 3D**):

$$P_{mu}=1/(1+\exp(-8/mu2*(n-mu2)))$$

The *softMax* specifies at which value we start applying a soft cap to the total number of synapses between the pair:

$$P_{SM}=2*softMax/((1+\exp(-(n-softMax)/5))*n)$$

where P_{SM} is probability to keep a synapse, and *n* is the initial number of synapses between the two neurons (**Figure 3E**). The *a3* parameter specifies what fraction of all connected pairs to keep, e.g. 0.8 means that 20% of all connected pairs will have all their synapses removed (**Figure 3F**). The *distPruning* defines a distance *d* dependent function P_{dist} : $d \rightarrow [0,1]$, where *d* is the distance from the soma along the dendrites (**Figure 3G**).

The final block *Neurons* defines the different neuron populations. Here each neuron template has its own block. For each neuron template we specify four files: *morphology* (a SWC file defining the soma, axon and dendrites), *parameters* (neuron parameters optimised using BluePyOpt), *mechanisms* (NEURON mechanisms), *modulation* (a JSON file defining the neuromodulation). The template can be used to define multiple neurons, the number defined by *num*. The *hoc* parameter is optional, and intended to be used in the future when exporting to SONATA format for use with Neurodamus (Williams et al. 2018). The *neuronType* can be either “neuron” or “virtualNeuron”, the latter can be used to define axons from other structures providing input to the striatum. The *rotationMode* lets us specify if the neurons should be left unrotated, or rotated in some manner. The *volumeID* defines which volume the neurons belong to.

```
    "Neurons": {
        "dSPN_0": {
            "morphology": "$DATA/neurons/striatum/dspn/str-dspn-
1/WT-0728MSN01-cor-rep-ax.swc",
            "parameters": "$DATA/neurons/striatum/dspn/str-dspn-
1/parameters.json",
            "mechanisms": "$DATA/neurons/striatum/dspn/str-dspn-
1/mechanisms.json",
            "modulation": "$DATA/neurons/striatum/dspn/str-dspn-
1/modulation.json",
            "num": 1218,
            "hoc": null,
            "neuronType": "neuron",
            "rotationMode": "random",
            "volumeID": "Striatum"
        },
```

The `$DATA` keyword is a shorthand for the `snudda/data` folder.

Configuring External Synaptic Input

The input spikes to the network are generated as prescribed in the input configuration file in JSON format. Below we will give a simple example of how to set up input, and there are more examples available on Github in the [Snudda/examples/notebooks]() directory.

In this example the dSPN will each receive 200 inputs, with 1Hz Poisson random spikes. The configuration also specifies the conductance and the `tmGlut` mod file that is used by NEURON to simulate the input synapses.

```
{
    "dSPN": {
        "Ctx" : {
            "generator" : "poisson",
            "frequency" : 1,
            "conductance" : 0.5e-9,
            "nInputs" : 200,
            "modFile": "tmGlut"
        }
    }
}
```

To complement the cortical (“Ctx”) input with thalamic, add a second input block with parameters inside the “dSPN” target block and give it a name e.g. “Thalamic”. The entire dSPN population will then receive both Cortical and Thalamic input.

Only one target block is applied to each neuron. When Snudda generates input for the network it iterates through all the different neurons in the simulation and picks the most specific target block that matches that neuron. In the example below a dSPN with neuron ID 5 and morphology dSPN_0 will match all three blocks, but the neuron ID is most specific so the “5” block will be used. A dSPN_1 morphology neuron will only match the dSPN block and will use that.

```
{
    "dSPN": {
        "Ctx": {
            "generator": "poisson",
            "start": [2, 5],
            "end": [3, 7],
            "frequency": [4, 2],
            "conductance": 0.5e-9,
            "nInputs": 200,
            "modFile": "tmGlut"
        }
    },

    "dSPN_0": {
        "Ctx": {
            "generator": "poisson",
```

```
            "start": 0,
            "end": 10,
            "frequency": 1,
            "conductance": 0.5e-9,
            "nInputs": 200,
            "modFile": "tmGlut"
        }
    },

    "5": {
        "Ctx": {
            "generator": "poisson",
            "start": 2,
            "end": 10,
            "frequency": 3,
            "conductance": 0.5e-9,
            "nInputs": 250,
            "modFile": "tmGlut"
        }
    }
}
```

In the dSPN target configuration the “start”, “end” and “frequency” are specified as vectors. Here the input is 4Hz at 2-3 seconds, and 2Hz at 5-7 seconds. We can also use population units to specify heterogenous external input to the target volume, see examples/notebooks on Github.

To create advanced inputs not supported by Snudda custom spike times can be read from a CSV file (with one spike train per row) by using `“generator”: “csv”` and `“csvFile”:”path/to/your/csvfile”`.

A more complex input example is given below. The “type” defines what sort of input the synapses form, e.g. “AMPA_NMDA” or “GABA”. The number of inputs to each neuron can either be defined directly using `nInputs` or indirectly by specifying the density of inputs `synapseDensity` along the dendrites. If both parameters are given, the code will use the density but scale it so that the `nInputs` are created. There is also an optional `parameterFile` that can be used to define a set of parameters for the synaptic channel.

The `populationCorrelation` describes how correlated the Poisson input is that is generated by mixing a shared mother process (each spike is included with probability P=sqrt(C)) and a number of independent child processes (inclusion probability 1 - P) to get the resulting input trains (Hjorth et al. 2009).

```
{
    "dSPN": {
      "CorticalSignal" : {
          "generator" : "poisson",
          "start" : 1.0,
```

```
            "end" : 1.5,
            "type: " : "AMPA_NMDA",
            "synapseDensity" : "0.05/(1+exp(-(d-30e-6)/5e-6))",
            "frequency" : 1,
            "populationCorrelation" : 0.0,
            "jitter" : 0.002,
            "conductance" : 0.5e-9,
            "modFile": "tmGlut",
            "parameterFile":
"$DATA/synapses/striatum/M1RH_Analysis_190925.h5-parameters-
MS.json"
        },
        "Thalamic" : {
            "generator" : "poisson",
            "type: " : "AMPA_NMDA",
            "synapseDensity" : "0.05*exp(-d/200e-6)",
            "frequency" : 1,
            "populationCorrelation" : 0.0,
            "jitter" : 0.01,
            "conductance" : 0.5e-9,
            "modFile": "tmGlut",

"parameterFile":"$DATA/synapses/striatum/TH_Analysis_191001.h5-
parameters-MS.json"
        }
      },
```

A virtual neuron is not simulated, instead the input specified for the virtual neuron becomes its activity. This can be used to model for example the activation of cortical axons in the striatum (here defined as a virtual neuron).

When using synapse density to place excitatory input onto the neurons, larger neurons will receive more input than smaller neurons, of the same type. However, size does not necessarily correlate with excitability of the neuron, or steepness of the I-V curve which depends on intrinsic channels. To handle this variation of excitability Snudda allows the user to scale the number of synapses reaching a neuron, a process which in a real network might be regulated by neuronal homeostatic processes.

What happens under the hood?

For each volume modelled the cell placement is restricted to be inside the mesh specified. The neurons are placed one by one, with coordinates randomly sampled from a uniform distribution. If a neuron position is inside the mesh, and there are no other neurons within a distance *dMin* from it, the position is accepted. To avoid an artificial increase of neuron density at the border, the neuron positions placed outside the mesh are also tracked. These

padding positions are not counted towards the total, and are discarded afterwards. Orientation of the neurons in the striatum is completely random, but it is possible to specify other ways to sample the orientation.

For touch detection the space is divided into voxels of 3 μm side length (**Figure 3B**). Synapses are only detected when axon and dendrite are present in the same voxel, the voxel size sets the maximal interaction distance. To parallelise the touch detection the voxels are grouped into *hypervoxels*, containing 100^3 voxels each. The mouse dorsal striatum occupies about 26.2 mm^3 (https://mouse.brain-map.org/), and contains almost 2 million neurons (Rosen and Williams 2001). The first step is to identify which neurons belong to which hypervoxels. For a large portion of the neurons they will be present in more than one hypervoxel. This procedure is done in parallel where the worker nodes of the parallel computer get allocated a subset of the neurons and based on the vertex coordinates of the neurites calculate which hyper voxel the neurons are in. The results are gathered, creating a list of neurons for each hypervoxel. The hypervoxels are then sorted based on the number of neurons inside, and those with most neurons are processed first. This is to avoid a situation where a computationally intensive hypervoxel is started towards the end of the touch detection. To perform the touch detection, a line parsing algorithm takes small steps Δx, Δy, Δz along all line segments of the dendrites, marking the voxels they intersect. The voxels contained within the soma are also marked. It then repeats the procedure for the axon line segments of the morphologies. Voxels that contain both axons and dendrites are considered to have a putative synapse if the two neuron types are allowed to have a connection between them (**Figure 3A,B**).

The purpose of the touch detection is to find the potential locations where neurons can connect to each other based on the restrictions set by the morphologies. The result of the above touch detection is a set of putative synapses, which is larger than the set of actual synapses. In the pruning step, the set of putative synapses is reduced to match the connectivity statistics from experimental pairwise recordings.

In experiments it is common to report only the binning size and the number of connected neuron pairs, and the total number of pairs. It would be beneficial for circuit modelling if the distance for each pair was also recorded, we could then extract distance dependent connectivity profiles and compare those to what the computer models predict. The pruning is divided into multiple steps, described above. The touch detection for a cubic millimeter can be run within several hours on a desktop, and the whole striatum can be created in an hour on a supercomputer. As an example, creating a striatal network with 10,000 neurons on a desktop Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz with 6 cores (12 workers with hyper threading) and 64GB RAM took: Init ~1s, Place 7s, Detect 1h and Prune 6 min. For 20,000 (50,000) neurons the corresponding times are ~1s (~1s), 12s (29s), 1h 53 min (4h 39min), 14 min (51min).

In addition to JSON configuration files, the resulting network data is stored in HDF5 files.

The challenge of limited morphology data

A big challenge of the biologically detailed anatomically constrained simulations of the neural microcircuits is availability of the high quality morphological reconstructions of the main neuron types, in sufficient numbers and variability. Open public morphometric repositories, similar to ModelDB for models present in the world-wide web since 1996 (McDougal et al. 2017), were pioneered in 2006 by G. Ascoli with NeuroMorpho.Org (Akram et al. 2018; http://neuromorpho.org/) and continued by other research centers like Allen Brain Institute (Jones et al. 2009; https://portal.brain-map.org), Janelia Research Campus (Gerfen et al. 2018; Economo et al. 2019; http://mouselight.janelia.org/), eBRAINS Knowledge Graph (https://kg.ebrains.eu/), to name a few, have become increasingly popular among computational neuroscientists.

Single-cell morphological reconstructions vary in quality due to the difference in experimental procedures leading to varying degrees of physical integrity of the neurites, spatial resolution, tissue shrinkage, slicing, etc. Need for consistent quality assurance of morphological reconstructions facilitated development of morphology processing tools for morphometric measurements, data processing and error correction, such as L-measure (Scorcioni et al. 2008; http://cng.gmu.edu:8080/Lm), TREES toolbox (Cuntz et al. 2010; https://www.treestoolbox.org/), btmorph (Torben-Nielsen, 2014; https://bitbucket.org/btorb/btmorph) and NeuroM/NeuroR (Anwar et al. 2009; https://github.com/BlueBrain/NeuroM; https://github.com/BlueBrain/NeuroR). Here we will illustrate typical use cases of manipulating morphological data on the example of a small Python module *treem* (https://github.com/a1eko/treem), developed by the authors in conjunction with *Snudda* as a complementary instrument to above mentioned packages.

Module *treem* provides data structure and command-line tools for accessing and manipulating the digital reconstructions of the neuron morphology in Stockley-Wheal-Cannon format, SWC (Cannon et al. 1998). Access to morphological data from the source code is supported by several Python classes. Common operations with SWC files are possible from the user-written scripts or via the command-line tool `swc`. For the detailed description of the user interface, see API and CLI references in the online documentation (https://treem.readthedocs.io).

A common reconstruction error is so called “z-jump” (Brown et al. 2011) when a part of the neurite gets shifted along the z-axis by a few micrometers as shown in **Figure 5A** (top panel). These can result from an accumulated error during the manual reconstruction or as a mistake in automatic procedure. Possible z-jumps can be eliminated in *treem* by the `repair` command using one of the four methods, *align*, *split*, *tilt* or *join*, as illustrated in **Figure 5A**. Choice of the repair method as well as the assessment of the result should ideally be left to the author of the reconstructed data; if this is not possible the preference is given to the method which better preserves cell symmetry.

Since neuronal tissue can shrink due to dehydration during histological preparation, correction factors are required before a reconstructed morphology can enter the simulation pipeline. Shrinkage correction involves scaling of the entire reconstruction in (x, y)-plane, expansion in z-direction, as well as decreasing contraction of selected neurites, e.g.

dendrites, by stretching along their principal axes (termed “unravelling” in Markram et al. 2015) or length-preserving spatial filtering (not shown, see online documentation).

Another important omission is that the neurons located close to the slice surface often have their neurites cut and thus missing in the digital reconstruction. Cut neurites can be replaced using the intact branches of the same topological order from the inner part of the slice, assuming spherical or axial symmetry of the neuron morphology as shown in **Figure 5B**. In *treem* this is achieved with the `repair` command (see online documentation for the example commands to reproduce **Figure 5B**).

One of the aspects of the large-scale simulations is realistic variability of the model parameters mimicking the natural spread of morpho-electric characteristics in live neurons. To enforce variability in the simulation based on the limited number of reconstructed neurons, we apply random manipulations to the morphological reconstructions. Examples of the length-preserving modifications implemented in *treem* are shown in **Figure 6A**. Methods *jitter*, *twist* and *rotate* do not change the length of the dendritic branches and thus do not affect electrophysiological features of the optimized models but help to recover spatial symmetry of the morphological reconstructions as shown in **Figure 6B**. To distribute excitability of the single-cell models, digital reconstructions can be scaled randomly in 3D, as was done in the large-scale simulation by Hjorth et al. (2020) and illustrated in the online documentation of *treem* (https://treem.readthedocs.io).

Discussion

In this paper we illustrate how to use our open source modelling pipeline to build microcircuit models. An important goal is that the model building process should be transparent and possible to reproduce by other labs, and the model should be extendable when new data accumulate. The pipeline is developed for setting up large-scale simulations of subcortical nuclei, such as striatum. In our current pipeline, based on the software *Snudda*, neurons can be placed in a defined volume, and then prediction of the location of synapses (as well as gap junctions) can be made using neuron morphologies and the specified pruning rules. Also data regarding for example short term synaptic plasticity or failure rates can be represented for the different synapse types. Finally a simulation using the NEURON simulation environment can be launched. A challenge when using cellular level data from public databases is that sometimes the data for the reconstructed neuronal morphologies only include soma and dendrites, missing the axon entirely. Therefore Snudda supports the prediction of synapses using different approaches. If the detailed morphology is available, the reconstructed axons and dendrites can be used to constrain which neurons are within reach of one another. If the axon is missing, the user can instead specify an axonal density which is then used for the synapse detection. Also our pipeline provides the opportunity to ‘repair’ the dendritic morphologies, and we have illustrated ways to do this using the software *treem*. In addition *treem* can provide jittering of morphological parameters to increase the variability in the modelled population of neurons, which is useful to avoid artefacts when there are too few available morphologies for each neuron type.

Models of neocortical microcircuits have been built with similar approaches (see above),

however, the specific softwares used to build these cortical microcircuit models might not always be open source or generalize to other brain regions because they require a certain amount, type and quality of data, and what types of data that exist regarding neuron types vary for different brain regions. Therefore we developed *Snudda* and *treem* inspired by the described workflows used for the neocortical microcircuits. We believe that our open source pipeline might become useful when building biophysically detailed microcircuit models of other subcortical brain regions, such as the other basal ganglia nuclei.

When using the current modelling pipeline, it is assumed that one has a collection of quantitatively detailed neuron models for each neuron type to be used in the modelled microcircuit. Such neuron models might come from public databases (see above). But most likely several of the neuron types in the selected microcircuit to be modelled might have to be built from scratch. Here the challenges are several as described above. Data on electrophysiological recordings published might be incomplete in such a way that only a few selected traces, as shown in the published manuscripts, exist. Also transcriptional, electrophysiological and morphological data might come from different experiments. Ideally, however, it would be best to have recordings from the neurons that were morphologically reconstructed, such as in patch-seq technique (e.g. Fuzik et al. 2016). Although the electrophysiological properties are well studied, one might lack the knowledge of which ion conductances are expressed in the neurons. Fortunately, such data are starting to emerge, and for example, for the striatum transcriptomics data already exist (Munoz-Manchado et al. 2018; Ho et al. 2018; Gokce et al. 2016; Saunders et al. 2018). If one has a good hypothesis of which channels are expressed, characterisation as well as models are starting to be collected at resources such as the Channelpedia (Ranjan et al. 2011; http://channelpedia.net). Although still not trivial, if one has both the morphology and electrophysiological data of a particular neuron type, workflows have already been developed for optimizing neuron models (Van Geit et al. 2016; Migliore et al. 2018; Masoli et al. 2020).

A natural future goal would, however, be to be able to link microcircuit models built in different labs, e.g. a cortical microcircuit connected to a striatal microcircuit. Then interoperability between models as well as model specification, such as SONATA, would be crucial. We have on our road map for Snudda to support the SONATA standard (Dai et al. 2020) and work has already started on it to become interoperable with the EBRAINS infrastructure (https://ebrains.eu/).

References

Akram, M., Nanda, S., Maraver, P. *et al.* An open repository for single-cell reconstructions of the brain forest. *Sci Data* **5,** 180006 (2018). https://doi.org/10.1038/sdata.2018.6

Akar NA *et al.* Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures. *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Pavia, Italy, 2019, pp. 274-282, doi: 10.1109/EMPDP.2019.8671560.

Amunts K, Knoll AC, Lippert T, et al. The Human Brain Project-Synergy between neuroscience, computing, informatics, and brain-inspired technologies. *PLoS Biol*. 2019;17(7):e3000344. Published 2019 Jul 1. doi:10.1371/journal.pbio.3000344

Anwar H., Riachi I., Schürmann F., Markram H. (2009). “An approach to capturing neuron morphological diversity,” in Computational Neuroscience: Realistic Modeling for Experimentalists. De Schutter E., editor. (Cambridge: The MIT Press) 211–232. DOI: 10.7551/mitpress/9780262013277.003.0010

Berthet P, Lindahl M, Tully PJ, Hellgren-Kotaleski J, Lansner A. Functional Relevance of Different Basal Ganglia Pathways Investigated in a Spiking Model with Reward Dependent Plasticity. *Front Neural Circuits*. 2016;10:53. Published 2016 Jul 21. doi:10.3389/fncir.2016.00053

Billeh YN, Cai B, Gratiy SL, et al. Systematic Integration of Structural and Functional Data into Multi-scale Models of Mouse Primary Visual Cortex. *Neuron*. 2020;106(3):388-403.e18. doi:10.1016/j.neuron.2020.01.040

Brown, K.M., Barrionuevo, G., Canty, A.J., De Paola, V., Hirsch, J.A., Jefferis, G.S., Lu, J., Snippe, M., Sugihara, I., Ascoli, G.A. (2011). The DIADEM Data Sets: Representative Light Microscopy Images of Neuronal Morphology to Advance Automation of Digital Reconstructions. Neuroinformatics 9(2-3):143–157.

Cannon RC, Turner DA, Pyapali GK, Wheal HV. An on-line archive of reconstructed hippocampal neurons. *J Neurosci Methods*. 1998;84(1-2):49-54. doi:10.1016/s0165-0270(98)00091-0

Carnevale T, Hines M. 2006. The NEURON Book. Cambridge, UK: Cambridge University Press, 2006.

Casali S., Marenzi E., Medini C., Casellato C., D’Angelo E. Reconstruction and Simulation of a Scaffold Model of the Cerebellar Network. Front. Neuroinf. 13:37. 2019. DOI**:**https://doi.org/10.3389/fninf.2019.00037

Colangelo C, Shichkova P, Keller D, Markram H, Ramaswamy S. Cellular, Synaptic and Network Effects of Acetylcholine in the Neocortex. *Front Neural Circuits*. 2019;13:24. Published 2019 Apr 12. doi:10.3389/fncir.2019.00024

Cuntz H, Forstner F, Borst A, Häusser M (2010). One rule to grow them all: A general theory of neuronal branching and its practical application. PLoS Comput Biol 6(8): e1000877.

Dai K, Hernando J, Billeh YN, et al. The SONATA data format for efficient description of large-scale network models. *PLoS Comput Biol*. 2020;16(2):e1007696. Published 2020 Feb 24. doi:10.1371/journal.pcbi.1007696

Economo MN, Winnubst J, Bas E, Ferreira TA, Chandrashekar J. Single-neuron axonal reconstruction: The search for a wiring diagram of the brain. *J Comp Neurol*. 2019;527(13):2190-2199. doi:10.1002/cne.24674

Einevoll GT, Destexhe A, Diesmann M, Grün S, Jirsa V, de Kamps M, Migliore M, Ness TV,

Plesser HE, Schürmann F. (2019) The Scientific Case for Brain Simulations. Neuron. 2019 May 22;102(4):735-744. doi: 10.1016/j.neuron.2019.03.027.

Fuzik J, Zeisel A, Máté Z, et al. Integration of electrophysiological recordings with single-cell RNA-seq data identifies neuronal subtypes. *Nat Biotechnol*. 2016;34(2):175-183. doi:10.1038/nbt.3443

Gerfen CR, Economo MN, Chandrashekar J. Long distance projections of cortical pyramidal neurons. *J Neurosci Res*. 2018;96(9):1467-1475. doi:10.1002/jnr.23978

Gleeson P, Crook S, Cannon RC, et al. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput Biol*. 2010;6(6):e1000815. Published 2010 Jun 17. doi:10.1371/journal.pcbi.1000815

Gokce O, Stanley GM, Treutlein B, Neff NF, Camp JG, Malenka RC, Rothwell PE, Fuccillo MV, Sudhof TC, Quake SR (2016) Cellular Taxonomy of the Mouse Striatum as Revealed by Single-Cell RNA-Seq. *Cell Rep.* 16(4):1126-1137. DOI: 10.1016/j.celrep.2016.06.059

Grillner S, Ip N, Koch C, et al. Worldwide initiatives to advance brain research. *Nat Neurosci*. 2016;19(9):1118-1122. doi:10.1038/nn.4371

Gurney K, Prescott TJ, Redgrave P. A computational model of action selection in the basal ganglia. II. Analysis and simulation of behaviour. *Biol Cybern*. 2001;84(6):411-423. doi:10.1007/PL00007985

Hellwig B. A quantitative analysis of the local connectivity between pyramidal neurons in layers 2/3 of the rat visual cortex. *Biol Cybern*. 2000;82(2):111-121. doi:10.1007/PL00007964

Hepburn I, W.Chen, S.Wils and E. De Schutter, STEPS: Efficient simulation of stochastic reaction-diffusion models in realistic morphologies. BMC Systems Biology 2012, 6:36. doi:10.1186/1752-0509-6-36.

Hines ML, Davison AP, Muller E. NEURON and Python. *Front Neuroinform*. 2009;3:1. Published 2009 Jan 28. doi:10.3389/neuro.11.001.2009

Hjorth J, Blackwell KT, Kotaleski JH. Gap junctions between striatal fast-spiking interneurons regulate spiking activity and synchronization as a function of cortical activity. *J Neurosci*. 2009;29(16):5276-5286. doi:10.1523/JNEUROSCI.6031-08.2009

Hjorth JJJ et al. (2020) The microcircuits of striatum in silico. Proc Natl Acad Sci U S A. 2020 Apr 28;117(17):9554-9565. doi: 10.1073/pnas.2000671117. Epub 2020 Apr 22.
Ho H et al. (2018) A Guide to Single-Cell Transcriptomics in Adult Rodent Brain: The Medium Spiny Neuron Transcriptome Revisited. Front. Cell. Neurosci. 12:159. DOI: 10.3389/fncel.2018.00159

Humphries MD, Wood R, Gurney K. Dopamine-modulated dynamic cell assemblies generated by the GABAergic striatal microcircuit. *Neural Netw*. 2009;22(8):1174-1188. doi:10.1016/j.neunet.2009.07.018

Insel TR, Landis SC, Collins FS. Research priorities. The NIH BRAIN Initiative. *Science*. 2013;340(6133):687-688. doi:10.1126/science.1239276

Jones AR, Overly CC, Sunkin SM. The Allen Brain Atlas: 5 years and beyond. *Nat Rev Neurosci*. 2009;10(11):821-828. doi:10.1038/nrn2722

Jordan J, Helias M, Diesmann M, Kunkel S. Efficient Communication in Distributed Simulations of Spiking Neuronal Networks With Gap Junctions. *Front Neuroinform*. 2020;14:12. Published 2020 May 5. doi:10.3389/fninf.2020.00012

Kanari L, Ramaswamy S, Shi Y, et al. Objective Morphological Classification of Neocortical Pyramidal Cells. *Cereb Cortex*. 2019;29(4):1719-1735. doi:10.1093/cercor/bhy339

Kumbhar P, Hines M, Fouriaux J, et al. CoreNEURON : An Optimized Compute Engine for the NEURON Simulator. *Front Neuroinform*. 2019;13:63. Published 2019 Sep 19. doi:10.3389/fninf.2019.00063

Markram H, Muller E, Ramaswamy S, Reimann MW, Abdellah M et al. (2015) Reconstruction and Simulation of Neocortical Microcircuitry. *Cell* 163(2): 456-492.

Masoli S, Tognolina M, Laforenza U, Moccia F, D'Angelo E. Parameter tuning differentiates granule cell subtypes enriching transmission properties at the cerebellum input stage. Commun Biol. 2020 May 8;3(1):222. doi: 10.1038/s42003-020-0953-x.

McDougal RA, Morse TM, Carnevale T, et al. Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience. *J Comput Neurosci*. 2017;42(1):1-10. doi:10.1007/s10827-016-0623-7

Migliore R, Lupascu CA, Bologna LL, Romani A, Courcol J-D, Antonel S, et al. (2018) The physiological variability of channel density in hippocampal CA1 pyramidal cells and interneurons explored using a unified data-driven modeling workflow. PLoS Comput Biol 14(9): e1006423. DOI:https://doi.org/10.1371/journal.pcbi.1006423

Munoz-Manchado et al. (2018) Diversity of Interneurons in the Dorsal Striatum Revealed by Single-Cell RNA Sequencing and PatchSeq. Cell Rep 24(8):2179-2190.e7. DOI: 10.1016/j.celrep.2018.07.053

Okano H, Miyawaki A, Kasai K. Brain/MINDS: brain-mapping project in Japan. *Philos Trans R Soc Lond B Biol Sci*. 2015;370(1668):20140310. doi:10.1098/rstb.2014.0310

Plesser H, Diesmann M, Gewaltig M, Morrison A (2015) Nest: the neural simulation tool. In *Encyclopedia of computational neuroscience*, ed. Jaeger D Jung R 1849-1852Springer New York.

Ranjan R, Khazen G, Gambazzi L, et al. Channelpedia: an integrative and interactive database for ion channels. *Front Neuroinform*. 2011;5:36. Published 2011 Dec 30.

doi:10.3389/fninf.2011.00036

Ray S, Bhalla US. PyMOOSE: Interoperable Scripting in Python for MOOSE. *Front Neuroinform*. 2008;2:6. Published 2008 Dec 19. doi:10.3389/neuro.11.006.2008

Reimann MW, King JG, Muller EB, Ramaswamy S, Markram H (2015) An algorithm to predict the connectome of neural microcircuits. *Front Comput Neurosci.* 9:120. doi:10.3389/fncom.2015.00120. eCollection 2015.

Rosen GD, Williams RW (2001) Complex trait analysis of the mouse striatum: independent QTLs modulate volume and neuron number. BMC Neurosci. 2001;2:5.

Saunders A, Macosko EZ, Wysoker A, Goldman M et al. Molecular Diversity and Specializations among the Cells of the Adult Mouse Brain. *Cell* 2018 Aug 9;174(4):1015-1030.e16. DOI: 10.1016/j.cell.2018.07.028

Scorcioni R., Polavaram S., Ascoli G.: L-Measure: a web-accessible tool for the analysis, comparison and search of digital reconstructions of neuronal morphologies. Nature Protocols. 3:866-76,(2008).

Sudhakar, S.K., Hong, S., Raikov, I., Publio, R., Lang, C., Close, T., Guo, D., Negrello, M., and De Schutter, E. (2017). Spatiotemporal network coding of physiological mossy fiber inputs by the cerebellar granular layer. PLoS Comput. Biol. 13, e1005754.

Torben-Nielsen B, An efficient and extendable Python library to analyze neuronal morphologies. Neuroinformatics, 2014, online first (10.1007/s12021-014-9232-7).

Van Geit W, Gevaert M, Chindemi G, Rössert C, Courcol J, Muller EB, Schürmann F, Segev I and Markram H (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

Wichert I, Jee S, De Schutter E and Hong S. Pycabnn: Efficient and Extensible Software to Construct an Anatomical Basis for a Physiologically Realistic Neural Network Model. Front. Neuroinform. 2020 doi: 10.3389/fninf.2020.00031

Wilkinson MD, Dumontier M, Aalbersberg IJ, et al. The FAIR Guiding Principles for scientific data management and stewardship [published correction appears in Sci Data. 2019 Mar 19;6(1):6]. *Sci Data*. 2016;3:160018. Published 2016 Mar 15. doi:10.1038/sdata.2016.18.

Williams, Timothy J., Balakrishnan, Ramesh, Delalondre, Fabien, Schuermann, Felix, Muller, Eilif, and Gewaltig, Marc Oliver. *Large-Scale Simulation of Brain Tissue, Blue Brain Project, EPFL*. United States: N. p., 2018. Web. doi:10.2172/1483995.

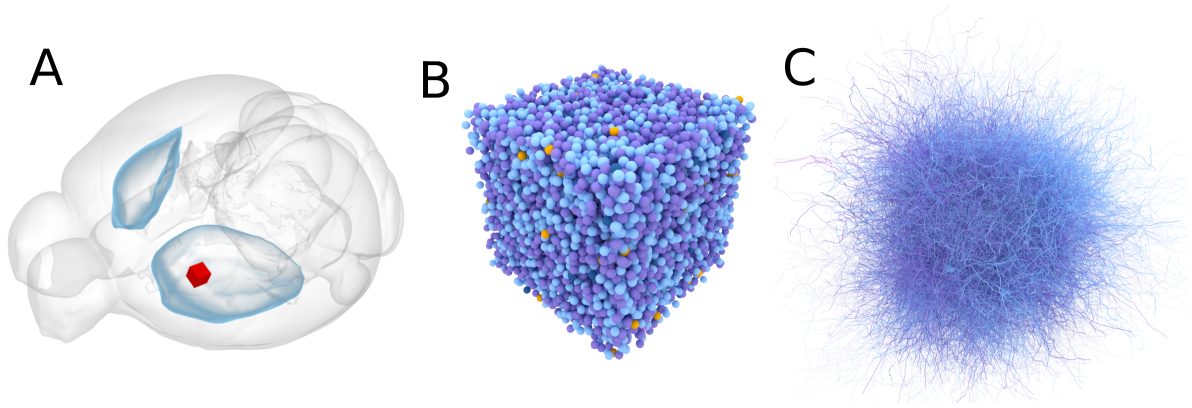

FIGURE 1 Example of the volume definition. **(A)** Selection of the volume of interest (red cube, size of the side 500 micrometers) inside the left part of the dorsal striatum (blue shells beneath the cerebral cortex). **(B)** The 10062 neuron somas placed within the red cube. **(C)** Illustration of 100 neurons showing the complexity of axons and dendrites.

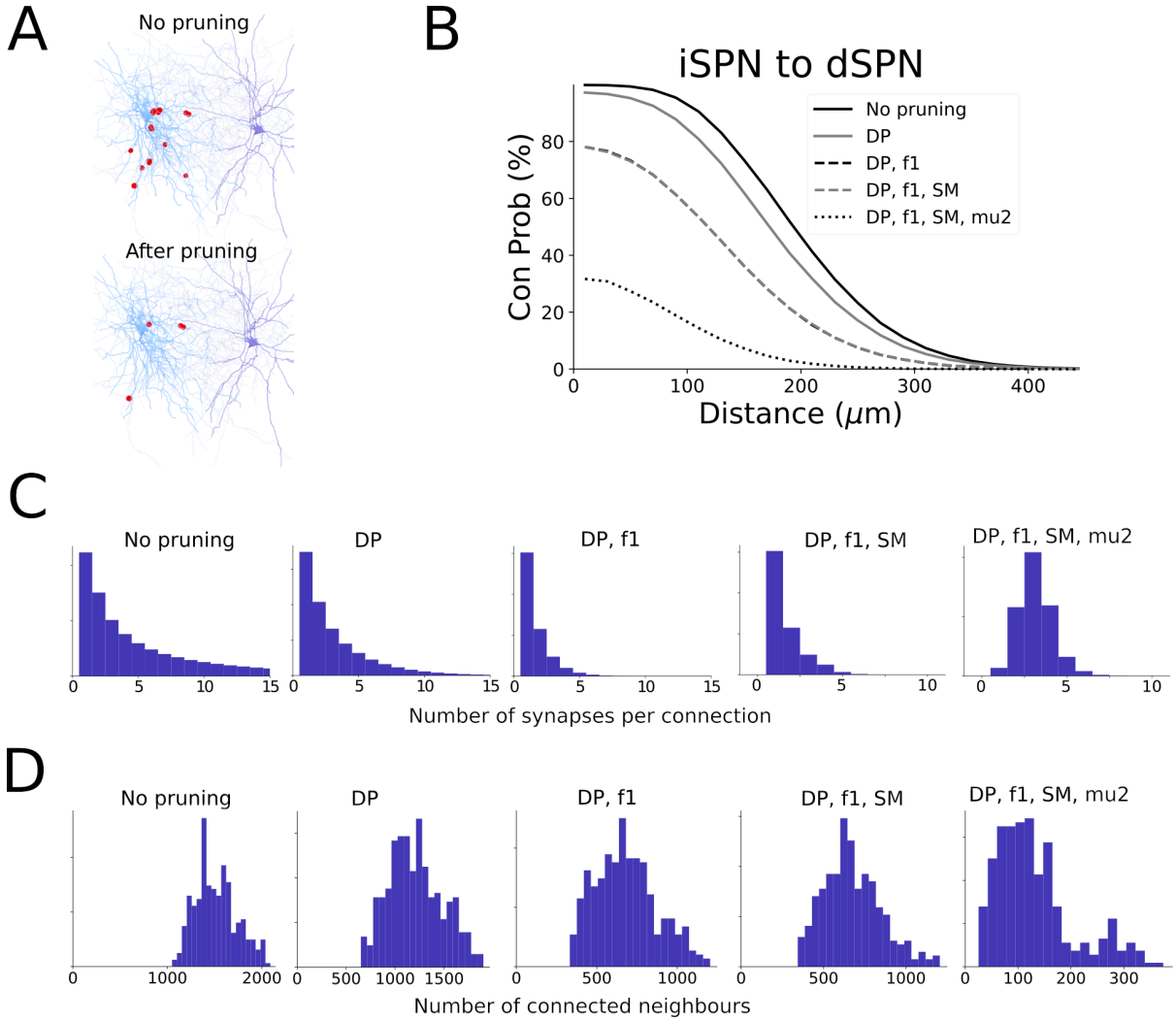

FIGURE 2 Example of the synaptic pruning procedure when connecting neurons within microcircuit. **(A)** Putative synapses detected between iSPN and dSPN shown top, and remaining synapses after pruning shown below. **(B)** Connection probability as a function of distance after each of the pruning steps. Distance dependent pruning filters synapses based on the distance to the soma on the postsynaptic neuron. A fraction *f1* of all synapses is removed. The soft max synapse filter does not disconnect any connected pairs as it only reduces the number of synapses of pairs that are connected by a large number of synapses. Finally *mu2* filters neuron pairs with few synapses, and leads to a large reduction in connectivity. **(C)** Number of synapses between connected pairs. **(D)** Number of connected neighbours each post synaptic neuron has.

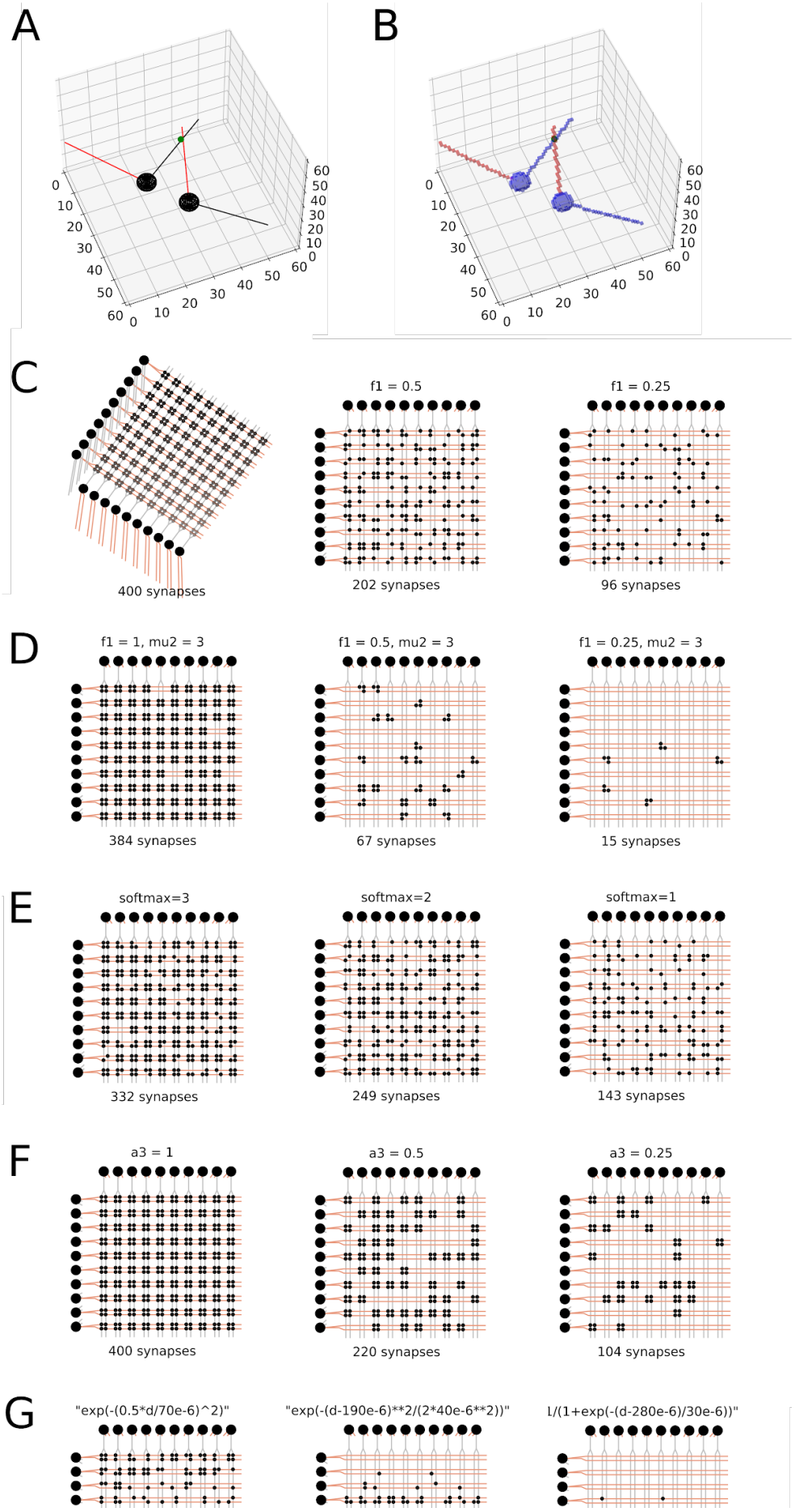
A
B
C
400 synapses
f1 = 0.5
202 synapses
f1 = 0.25
96 synapses
D
f1 = 1, mu2 = 3
384 synapses
f1 = 0.5, mu2 = 3
67 synapses
f1 = 0.25, mu2 = 3
15 synapses
E
softmax=3
332 synapses
softmax=2
249 synapses
softmax=1
143 synapses
F
a3 = 1
400 synapses
a3 = 0.5
220 synapses
a3 = 0.25
104 synapses
G
"exp(-(0.5*d/70e-6)^2)"
"exp(-(d-190e-6)**2/(2*40e-6**2))"
1/(1+exp(-(d-280e-6)/30e-6))"

FIGURE 3 Touch detection and pruning. **(A)** Illustration of ball and stick neurons with soma and dendrites marked in black, axon in red and synapse in green. **(B)** Corresponding two neurons in the hypervoxel representation. The neurites are traced by taking small steps along Δx, Δy, Δz corresponding to the direction of the neurite. Where axon and dendrites occupy the same voxel a putative synapse is detected, here marked by a green dot. The volume of the soma is also voxelized. **(C)** Ball and stick neurons arranged so their synapses are on a grid, with four synapses connecting every neuron pair. Pruning with parameter f1=1, 0.5 and 0.25 keeping 100%, 50% and 25% of all synapses. **(D)** Combination of f1 and mu2 pruning. Here P = f1 * 1.0 / (1.0 + exp(-8.0 / mu2 * (nSynapses - mu2))), where P is the probability to keep a synapse, nSynapses is the total number of synapses connecting the neuron pre-post pair. **(E)** SoftMax pruning step. If there are more than softMax synapses then the probability of keeping synapses between that pair is P = 2 * softMax / ((1 + exp(-(nSynapses - softMax) / 5)) * nSynapses). **(F)** Pruning a3 = 1, 0.5 and 0.25 removing all synapses between a connected pair in 0%, 50% and 75% of the cases. **(G)** Distance dependent pruning of proximal, medial and distal synapses. A Jupyter Notebook to generate this figure is available on Snudda GitHub.

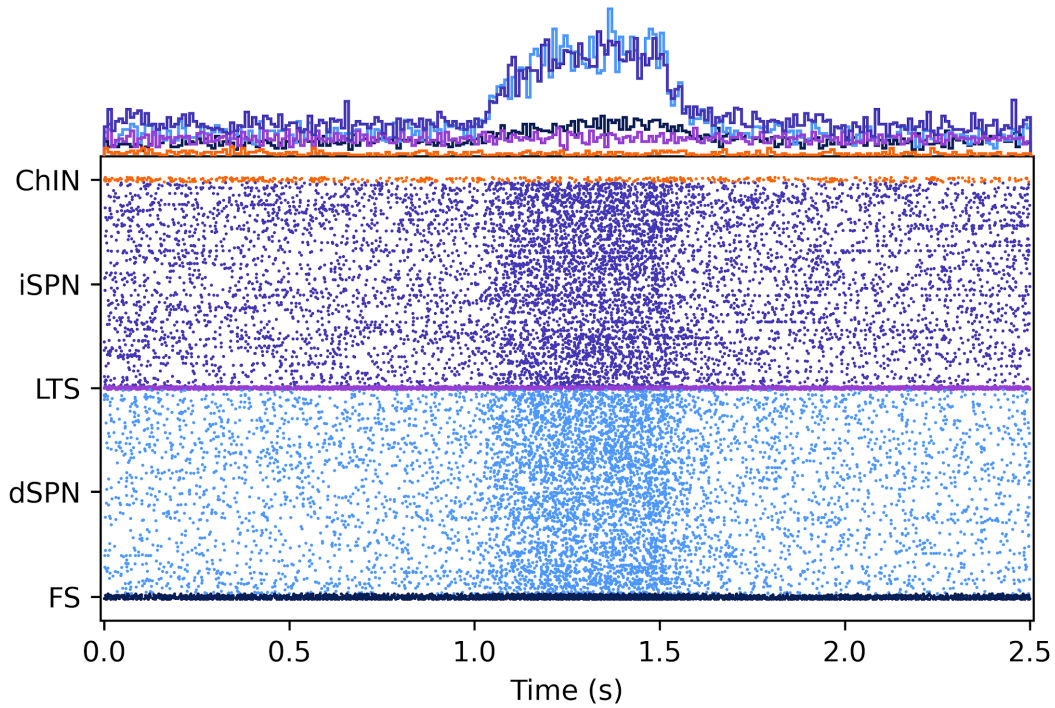

FIGURE 4 Simulation of 10062 striatal neurons (4872 dSPN, 4872 iSPN, 133 FS, 113 ChIN, 72 LTS) receiving cortical and thalamic input. The cortical drive is increased for half a second at 1.0 seconds. The histograms above the spike raster show the total number of spikes in the respective neuron populations.

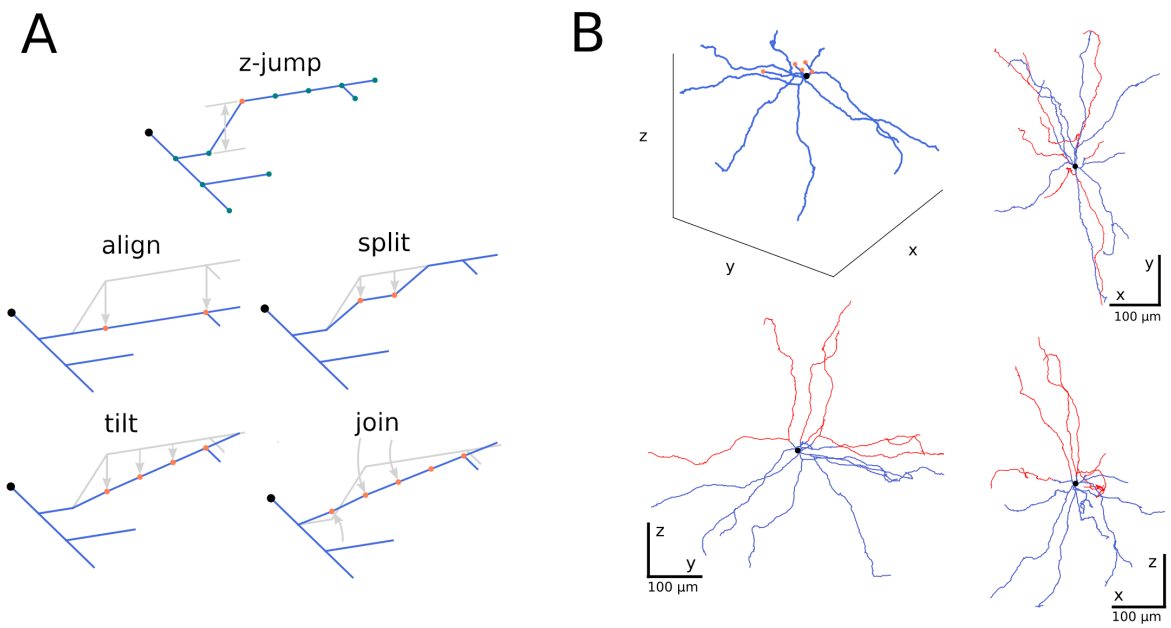

FIGURE 5 Repairing digital reconstruction of the neuron morphology. **(A)** Correcting "z-jump" reconstruction errors (top panel). Dots illustrate reconstructed points (soma is black, dendrites are blue, the orange dot labels the node at the point of presumed discontinuity). Four correction methods implemented in the Python module *treem* are shown below (*align*, *split*, *tilt* and *join*). **(B)** Repairing the dendrites cut at the slice border. Orange dots in a 3D plot show the cut points of the dendrites. Red lines in 2D projections show "repaired" dendrites, i.e. extended neurites using undamaged reconstructions of the same topological order as the cut branches.

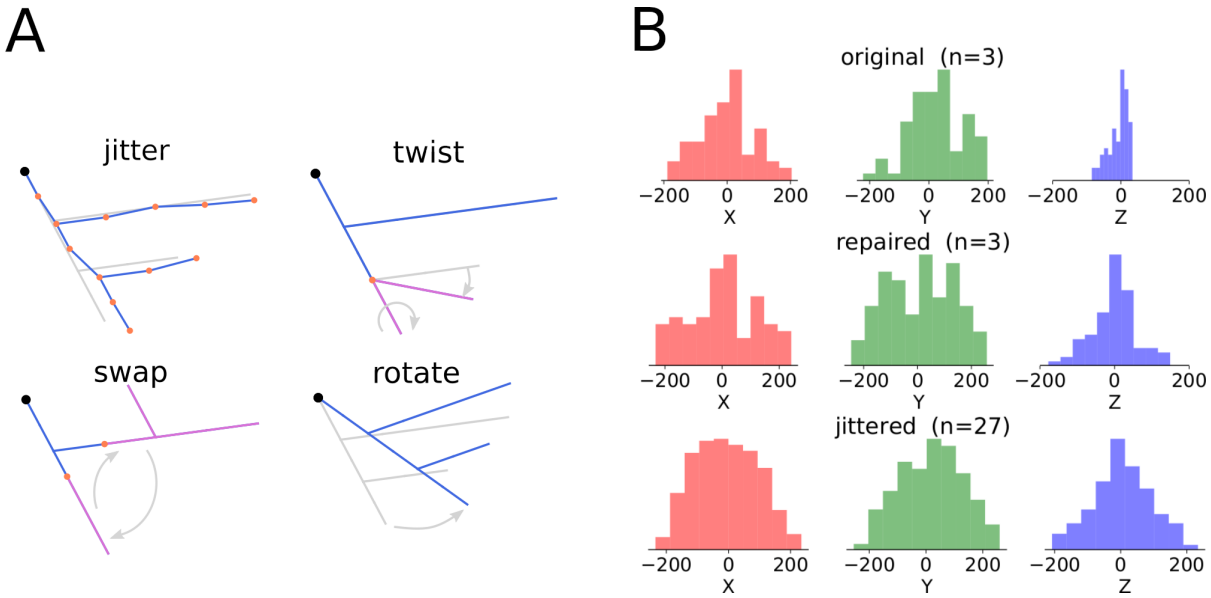

FIGURE 6 Adding variability to the morphological reconstructions. **(A)** Examples of modification methods used in *treem* (modifications preserving the total length are shown). **(B)** Distribution of the coordinates of the dendritic terminations of the fast-spiking interneurons at different stages of morphology processing - original reconstructions (n=3), repaired reconstructions (n=3) and “jittered”, i.e. randomly manipulated reconstructions, nine variants per each cell (n=27). Here, only random twisting of the dendritic branches at the bifurcation points was applied which proved to be sufficient to restore the symmetry.