Simulation of the Inferior Olive in Arbor

Lennart P. L. Landsmeer
Showcase for the Arbor Workshop

The inferior olive model - an example of a model running in Arbor

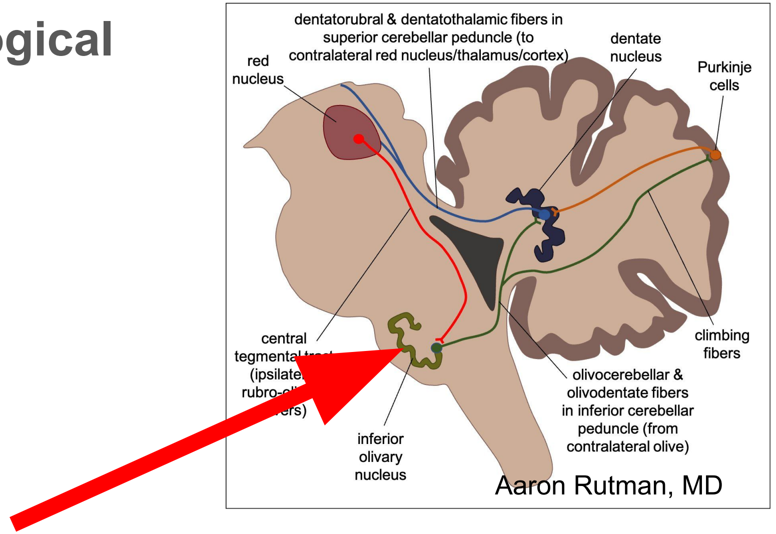
Central in motor control, learning & timing

Located in the brain stem

Part of the olivocerebellar loop (PF ➜ PC –| DCN –| IO –| PC)

RQ: relation between morphological/topological clustering and dynamical clustering

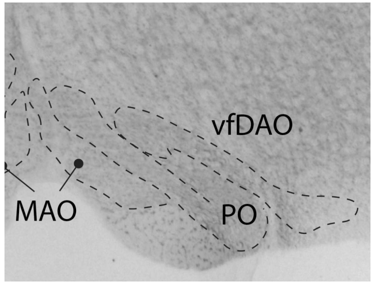
Model part of my master thesis
at Erasmus MC Rotterdam and TU Delft
under Dr. Mario Negrello

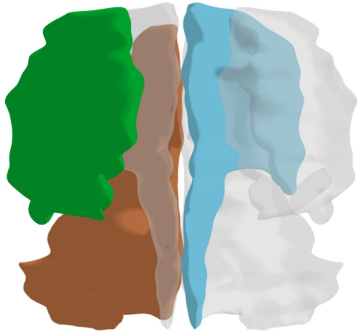


dentatorubral & dentatothalamic fibers in superior cerebellar peduncle (to contralateral red nucleus/thalamus/cortex)
red nucleus
dentate nucleus
Purkinje cells
central tegmental trac
(ipsilate
rubro-oli
ers)
inferior olivary nucleus
climbing fibers
olivocerebellar & olivodentate fibers in inferior cerebellar peduncle (from contralateral olive)
Aaron Rutman, MD

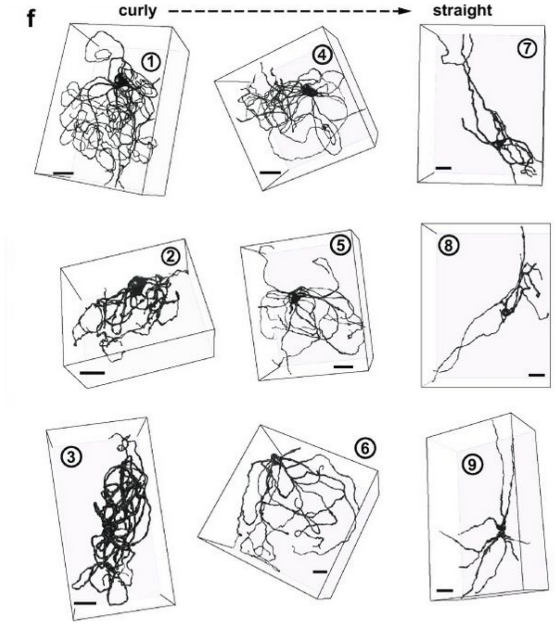
Step 1. Experimental Constraints -> IO Network generation

Geometry

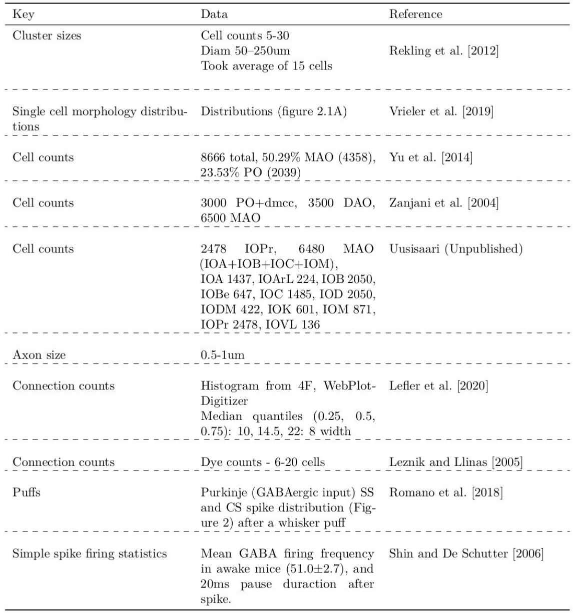




Morphology

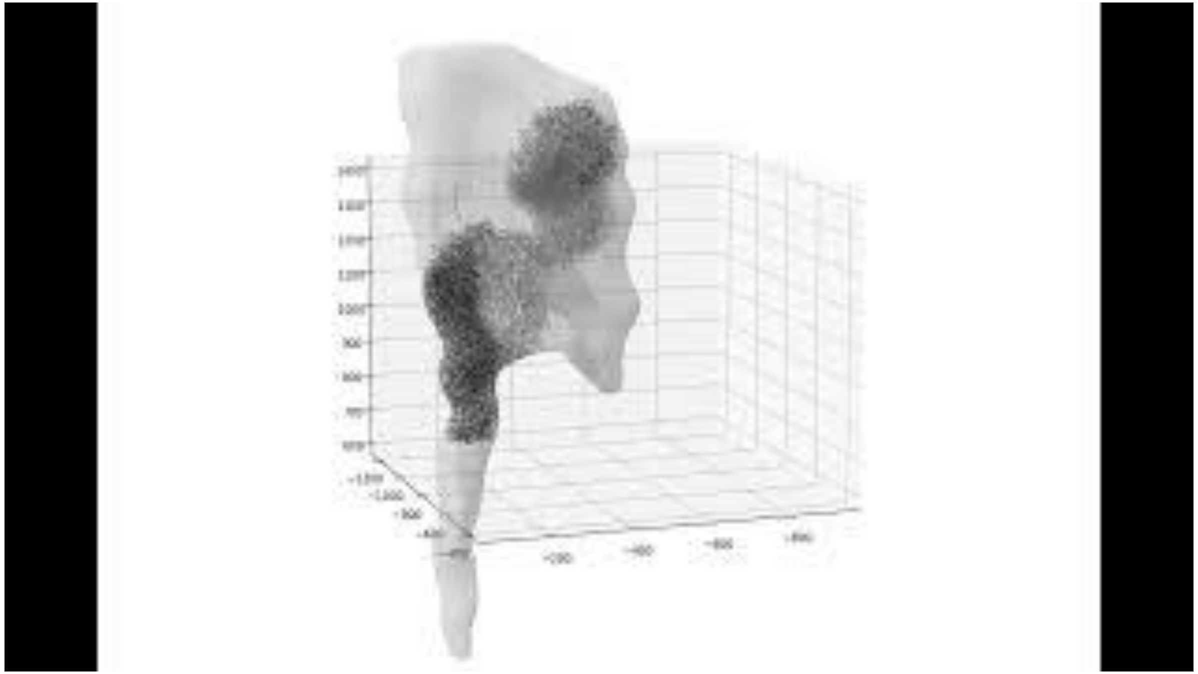


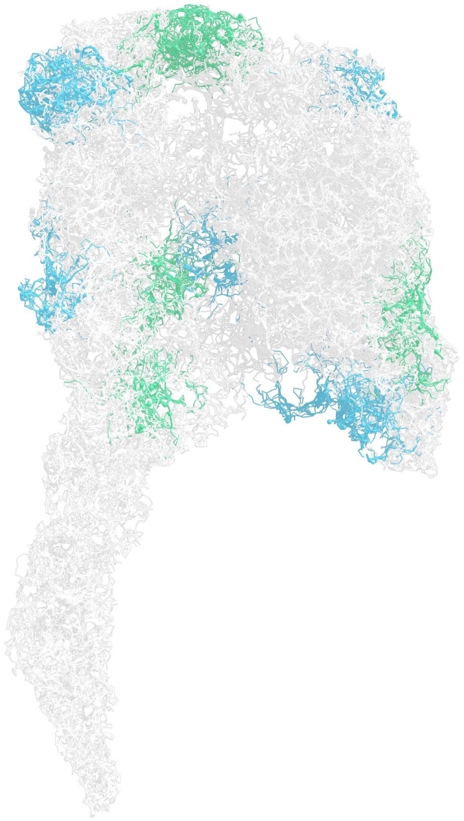
Network



Key	Data	Reference
Cluster sizes	Cell counts 5-30 Diam 50–250um Took average of 15 cells	Rekling et al. [2012]
Single cell morphology distributions	Distributions (figure 2.1A)	Vrieler et al. [2019]
Cell counts	8666 total, 50.29% MAO (4358), 23.53% PO (2039)	Yu et al. [2014]
Cell counts	3000 PO+dmcc, 3500 DAO, 6500 MAO	Zanjani et al. [2004]
Cell counts	2478 IOPr, 6480 MAO (IOA+IOB+IOC+IOM), IOA 1437, IOArL 224, IOB 2050, IOBe 647, IOC 1485, IOD 2050, IODM 422, IOK 601, IOM 871, IOPr 2478, IOVL 136	Uusisaari (Unpublished)
Axon size	0.5-1um	
Connection counts	Histogram from 4F, WebPlot-Digitizer Median quantiles (0.25, 0.5, 0.75): 10, 14.5, 22: 8 width	Lefler et al. [2020]
Connection counts	Dye counts - 6-20 cells	Leznik and Llinas [2005]
Puffs	Purkinje (GABAergic input) SS and CS spike distribution (Figure 2) after a whisker puff	Romano et al. [2018]
Simple spike firing statistics	Mean GABA firing frequency in awake mice (51.0±2.7), and 20ms pause duraction after spike.	Shin and De Schutter [2006]

Step 1. Network generation visualized





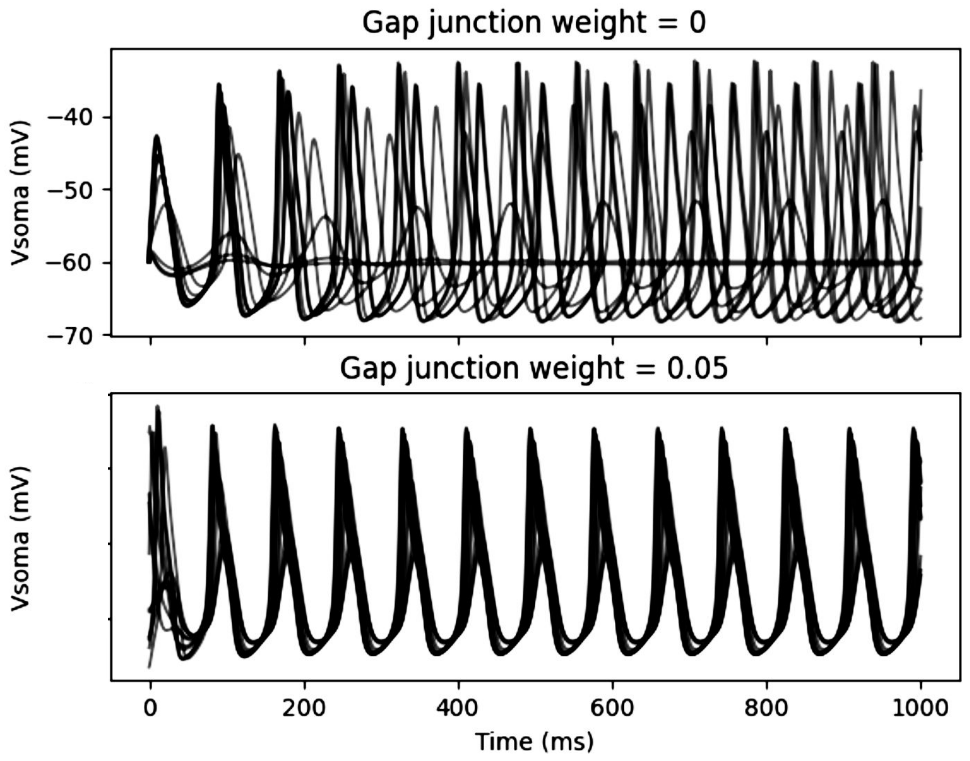
Step 2. Dynamics (ion channel mechanisms in Arbor)

Subthreshold oscillations

Cluster synchronization

Two spike types (normal & ADP)

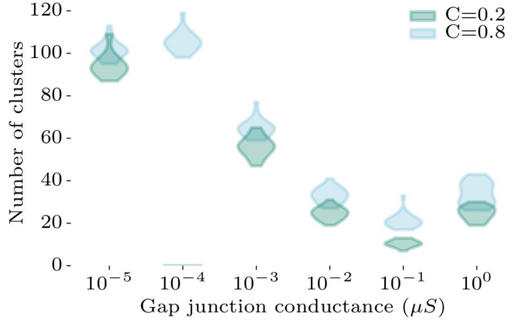
Single cell translation done by student Rocher Smol



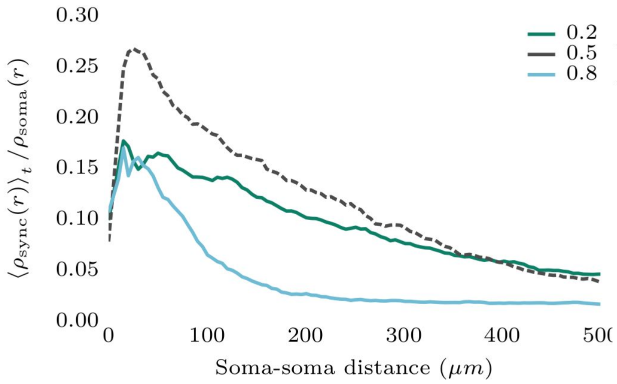
Step 3. Simulation

Step 4. Results: Effect topology on synchronization

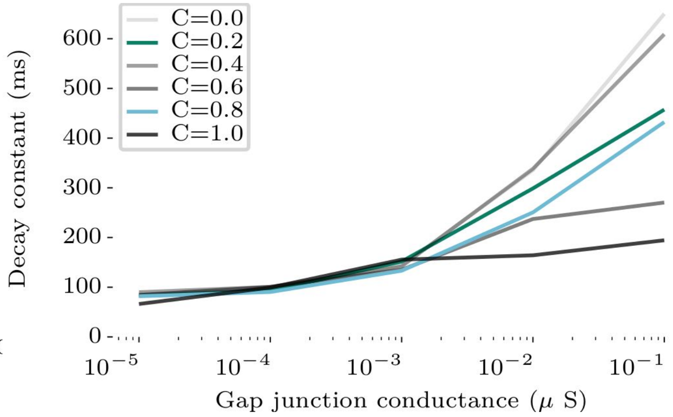
A. Cluster counts



B. RDF

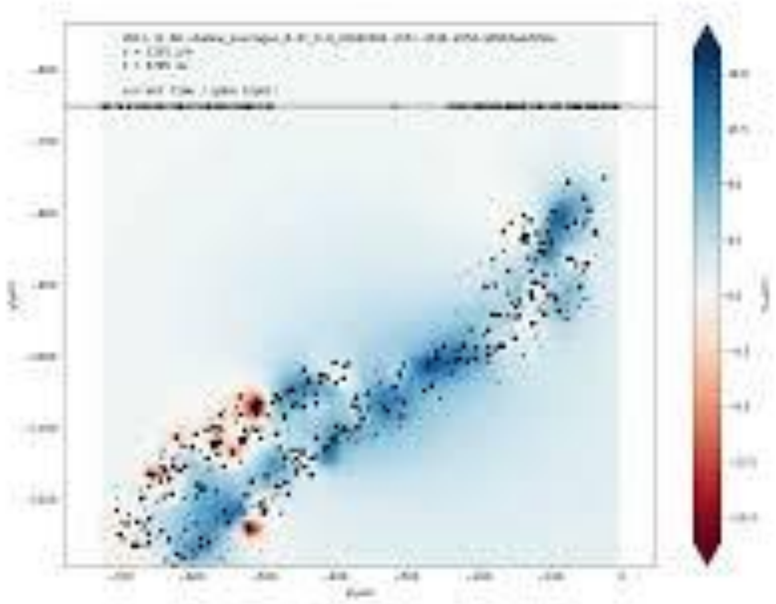


C. Memory

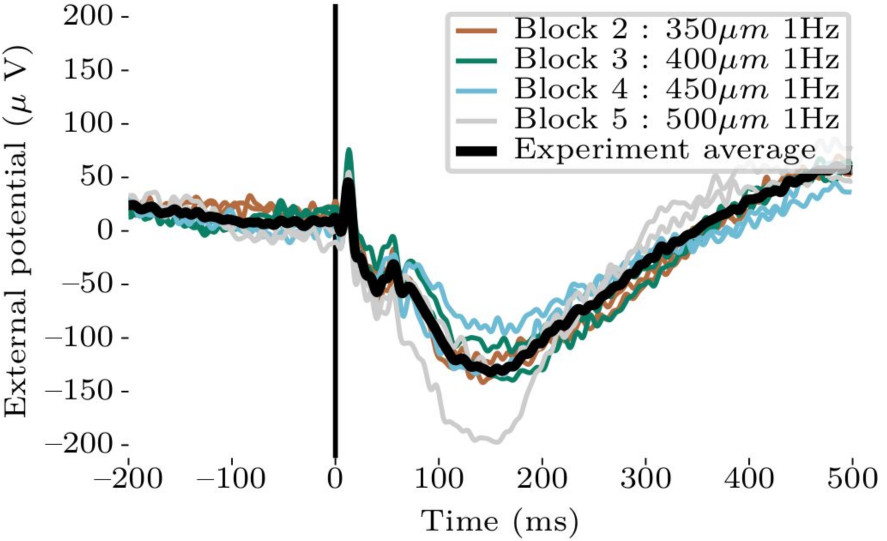


1. Cluster synchronization happens regardless of topological clustering
2. But topological cluster does affect memory and cluster shape

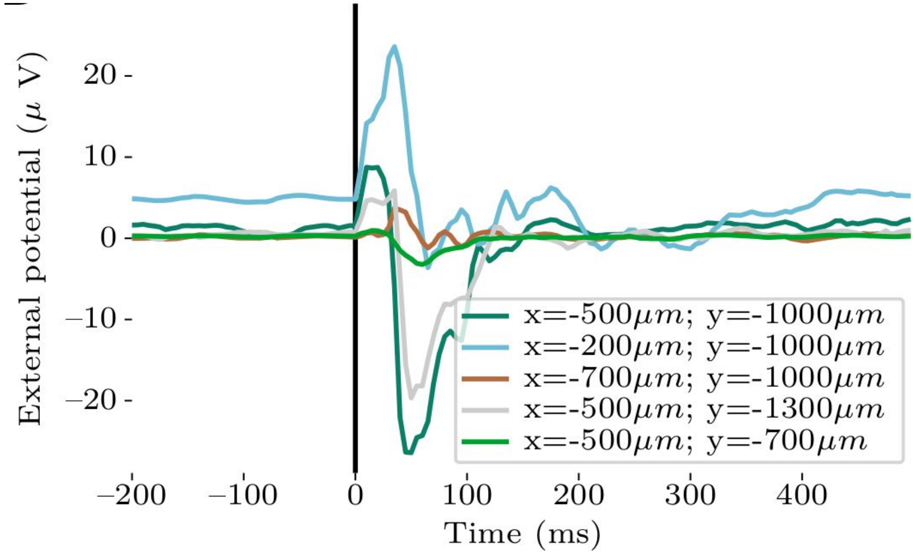
Step 5. Local field potentials



Step 5. Local fields potentials: experiment vs model



EXPERIMENT



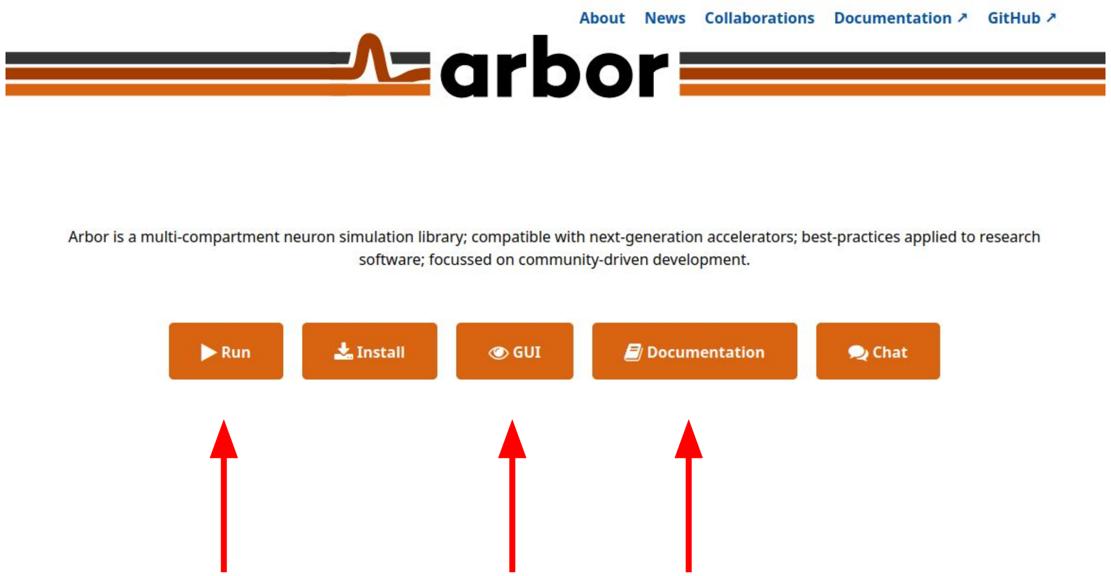
MODEL

Next up: workshop

Questions? (now or come talk to me at my poster)

Workshop: interactive part

arbor-sim.org



Arbor is a multi-compartment neuron simulation library; compatible with next-generation accelerators; best-practices applied to research software; focussed on community-driven development.

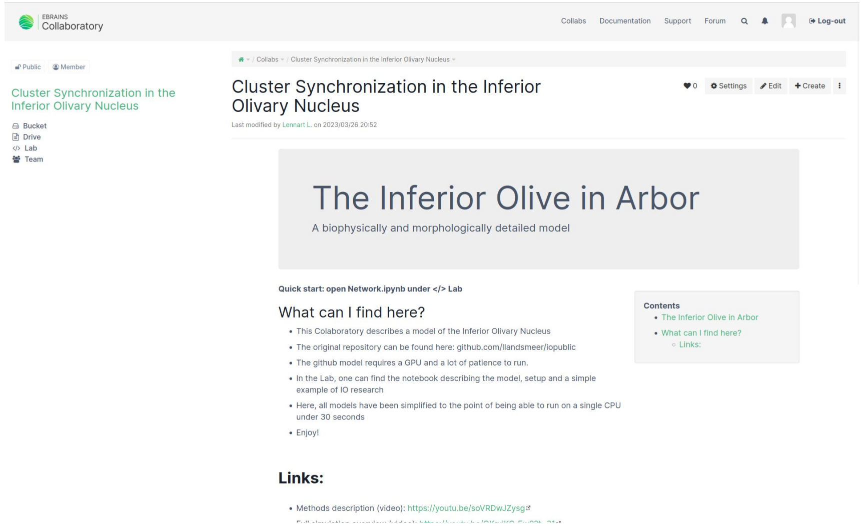
Links

wiki.ebrains.eu/bin/view/Collabs/io-clusters/

Or go to **wiki.ebrains.eu** and search for **“Inferior Olivary Nucleus”**

arbor-sim.org/playground/

DIY



Cluster Synchronization in the Inferior Olivary Nucleus

Cluster Synchronization in the Inferior Olivary Nucleus

Last modified by Lennart L. on 2023/03/26 20:52

The Inferior Olive in Arbor

A biophysically and morphologically detailed model

Quick start: open Network.ipynb under </> Lab

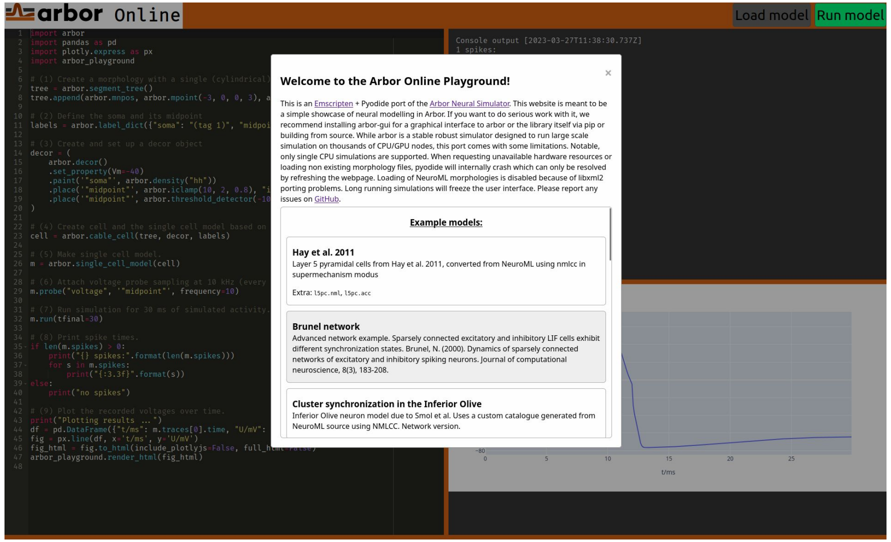
What can I find here?

- This Colaboratory describes a model of the Inferior Olivary Nucleus
- The original repository can be found here: github.com/llandsmeer/iopublic
- The github model requires a GPU and a lot of patience to run.
- In the Lab, one can find the notebook describing the model, setup and a simple example of IO research
- Here, all models have been simplified to the point of being able to run on a single CPU under 30 seconds
- Enjoy!

Contents
- The Inferior Olive in Arbor
- What can I find here?
  - Links:

Links:

- Methods description (video): https://youtu.be/soVRDwJZysg



```
import arbor
import pandas as pd
import plotly.express as px
import arbor_playground

# (1) Create a morphology with a single (cylindrical)
tree = arbor.segment_tree()
tree.append(arbor.mnpos, arbor.mpoint(-3, 0, 0, 3), a

# (2) Define the soma and its midpoint
labels = arbor.label_dict({"soma": "(tag 1)", "midpoi

# (3) Create and set up a decor object
decor = (
    arbor.decor()
    .set_property(Vm=-40)
    .paint('"soma"', arbor.density("hh"))
    .place('"midpoint"', arbor.iclamp(10, 2, 0.8), "i
    .place('"midpoint"', arbor.threshold_detector(-10
)

# (4) Create cell and the single cell model based on
cell = arbor.cable_cell(tree, decor, labels)

# (5) Make single cell model.
m = arbor.single_cell_model(cell)

# (6) Attach voltage probe sampling at 10 kHz (every
m.probe("voltage", '"midpoint"', frequency=10)

# (7) Run simulation for 30 ms of simulated activity.
m.run(tfinal=30)

# (8) Print spike times.
if len(m.spikes) > 0:
    print("{} spikes:".format(len(m.spikes)))
    for s in m.spikes:
        print("{:3.3f}".format(s))
else:
    print("no spikes")

# (9) Plot the recorded voltages over time.
print("Plotting results ...")
df = pd.DataFrame({"t/ms": m.traces[0].time, "U/mV":
fig = px.line(df, x='t/ms', y='U/mV')
fig_html = fig.to_html(include_plotlyjs=False, full_html=False)
arbor_playground.render_html(fig_html)
```

```
Console output [2023-03-27T11:38:30.737Z]
1 spikes:
```

Welcome to the Arbor Online Playground!

This is an Emscripten + Pyodide port of the Arbor Neural Simulator. This website is meant to be a simple showcase of neural modelling in Arbor. If you want to do serious work with it, we recommend installing arbor-gui for a graphical interface to arbor or the library itself via pip or building from source. While arbor is a stable robust simulator designed to run large scale simulation on thousands of CPU/GPU nodes, this port comes with some limitations. Notable, only single CPU simulations are supported. When requesting unavailable hardware resources or loading non existing morphology files, pyodide will internally crash which can only be resolved by refreshing the webpage. Loading of NeuroML morphologies is disabled because of libxml2 porting problems. Long running simulations will freeze the user interface. Please report any issues on GitHub.

Example models:

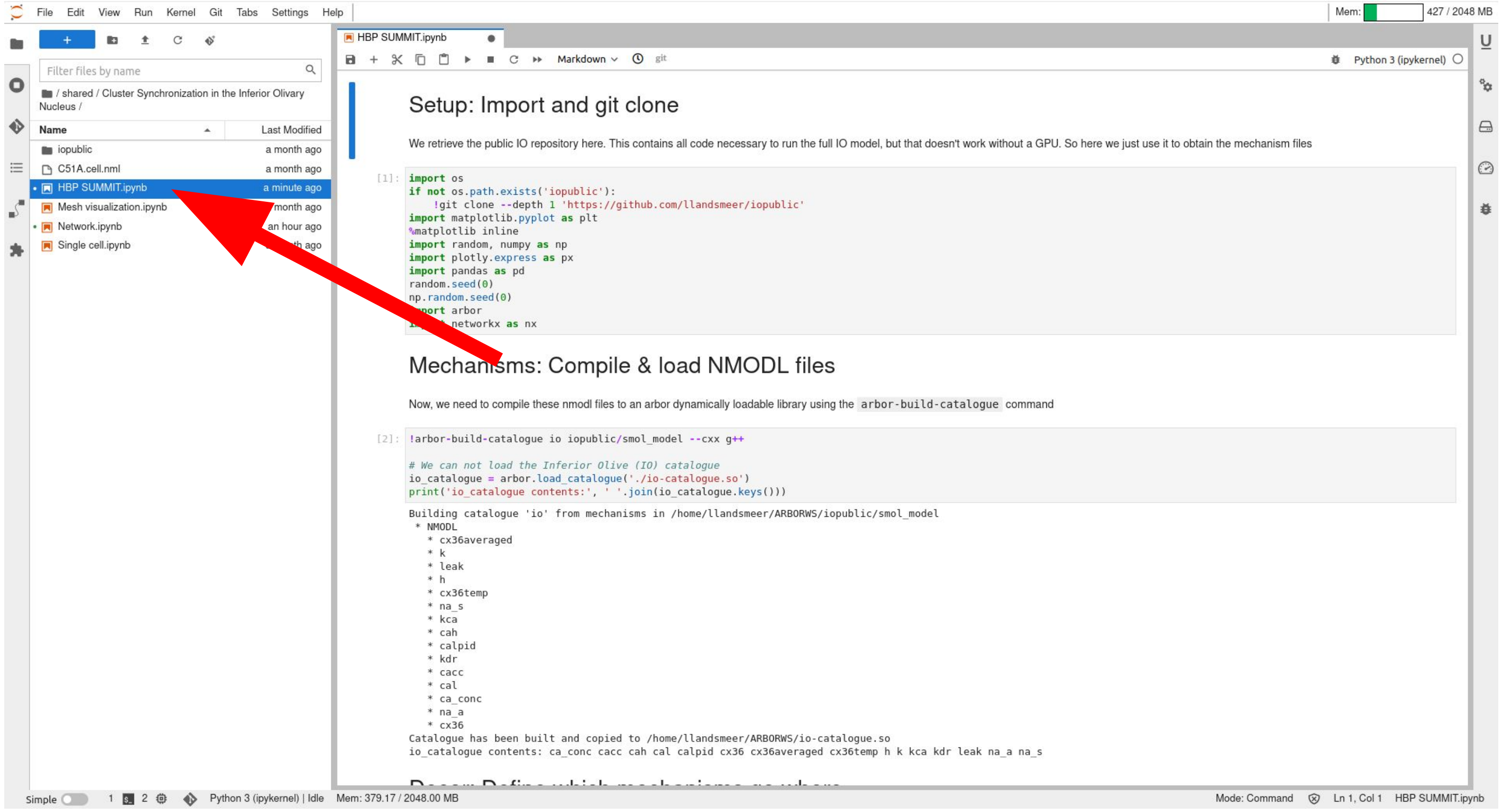
Hay et al. 2011
Layer 5 pyramidal cells from Hay et al. 2011, converted from NeuroML using nmlcc in supermechanism modus

Extra: `l5pc.nml`, `l5pc.acc`

Brunel network
Advanced network example. Sparsely connected excitatory and inhibitory LIF cells exhibit different synchronization states. Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. Journal of computational neuroscience, 8(3), 183-208.

Cluster synchronization in the Inferior Olive
Inferior Olive neuron model due to Smol et al. Uses a custom catalogue generated from NeuroML source using NMLCC. Network version.

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Name	Last Modified
iopublic	a month ago
C51A.cell.nml	a month ago
HBP SUMMIT.ipynb	a minute ago
Mesh visualization.ipynb	month ago
Network.ipynb	an hour ago
Single cell.ipynb	h ago

Setup: Import and git clone

We retrieve the public IO repository here. This contains all code necessary to run the full IO model, but that doesn't work without a GPU. So here we just use it to obtain the mechanism files

```
[1]: import os
     if not os.path.exists('iopublic'):
         !git clone --depth 1 'https://github.com/llandsmeer/iopublic'
     import matplotlib.pyplot as plt
     %matplotlib inline
     import random, numpy as np
     import plotly.express as px
     import pandas as pd
     random.seed(0)
     np.random.seed(0)
     import arbor
     import networkx as nx
```

Mechanisms: Compile & load NMODL files

Now, we need to compile these nmodl files to an arbor dynamically loadable library using the `arbor-build-catalogue` command

```
[2]: !arbor-build-catalogue io iopublic/smol_model --cxx g++

     # We can not load the Inferior Olive (IO) catalogue
     io_catalogue = arbor.load_catalogue('./io-catalogue.so')
     print('io_catalogue contents:', ' '.join(io_catalogue.keys()))
```

```
Building catalogue 'io' from mechanisms in /home/llandsmeer/ARBORWS/iopublic/smol_model
 * NMODL
   * cx36averaged
   * k
   * leak
   * h
   * cx36temp
   * na_s
   * kca
   * cah
   * calpid
   * kdr
   * cacc
   * cal
   * ca_conc
   * na_a
   * cx36
Catalogue has been built and copied to /home/llandsmeer/ARBORWS/io-catalogue.so
io_catalogue contents: ca_conc cacc cah cal calpid cx36 cx36averaged cx36temp h k kca kdr leak na_a na_s
```

Workflow:

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

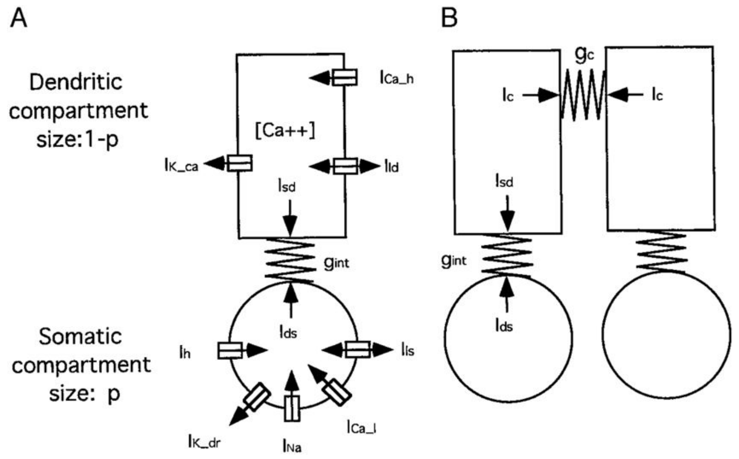
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Schweighofer, Nicolas, Kenji Doya, and Mitsuo Kawato. "Electrophysiological properties of inferior olive neurons: a compartmental model." *Journal of neurophysiology* 82.2 (1999): 804-817.

Setup: import and git clone

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

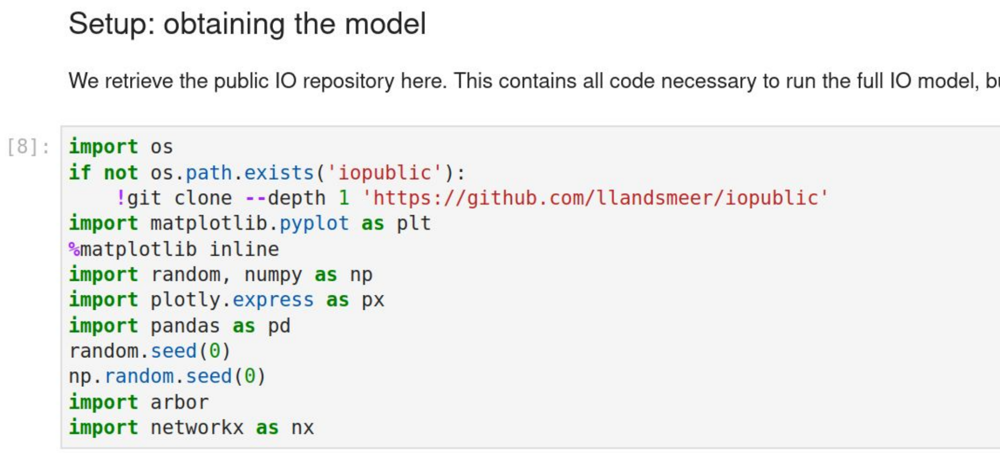
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Setup: obtaining the model

We retrieve the public IO repository here. This contains all code necessary to run the full IO model, bu

```
[8]: import os
     if not os.path.exists('iopublic'):
         !git clone --depth 1 'https://github.com/llandsmeer/iopublic'
     import matplotlib.pyplot as plt
     %matplotlib inline
     import random, numpy as np
     import plotly.express as px
     import pandas as pd
     random.seed(0)
     np.random.seed(0)
     import arbor
     import networkx as nx
```

Mechanisms: compile & load NMODL files

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

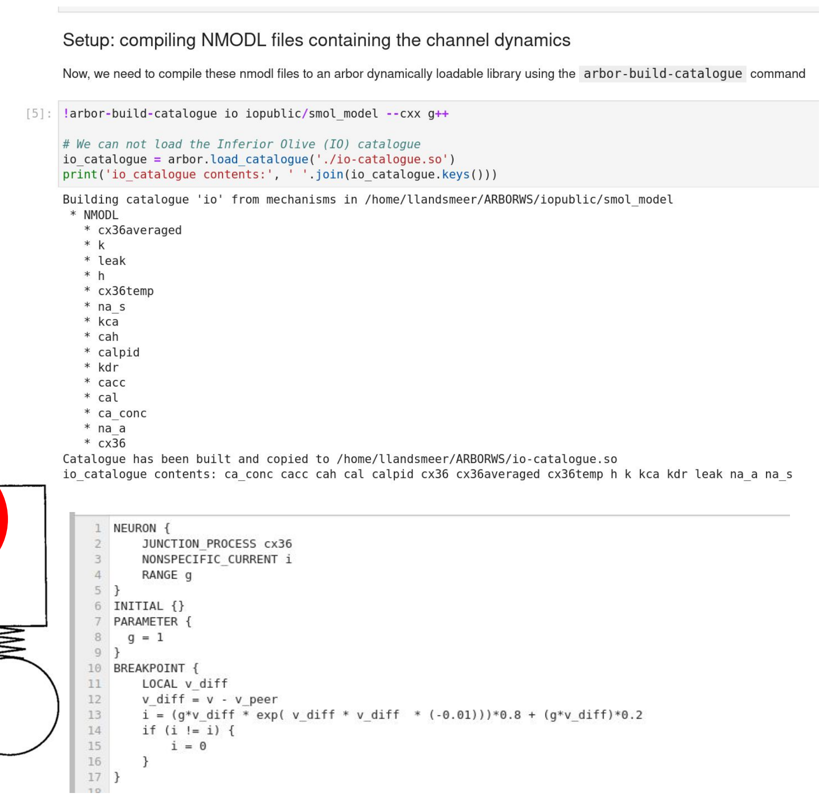
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Setup: compiling NMODL files containing the channel dynamics

Now, we need to compile these nmodl files to an arbor dynamically loadable library using the `arbor-build-catalogue` command

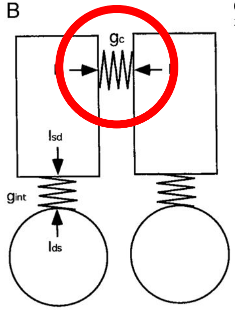
```
[5]: !arbor-build-catalogue io iopublic/smol_model --cxx g++

     # We can not load the Inferior Olive (IO) catalogue
     io_catalogue = arbor.load_catalogue('./io-catalogue.so')
     print('io_catalogue contents:', ' '.join(io_catalogue.keys()))
```

```
Building catalogue 'io' from mechanisms in /home/llandsmeer/ARBORWS/iopublic/smol_model
 * NMODL
   * cx36averaged
   * k
   * leak
   * h
   * cx36temp
   * na_s
   * kca
   * cah
   * calpid
   * kdr
   * cacc
   * cal
   * ca_conc
   * na_a
   * cx36
Catalogue has been built and copied to /home/llandsmeer/ARBORWS/io-catalogue.so
io_catalogue contents: ca_conc cacc cah cal calpid cx36 cx36averaged cx36temp h k kca kdr leak na_a na_s
```

```
NEURON {
    JUNCTION_PROCESS cx36
    NONSPECIFIC_CURRENT i
    RANGE g
}
INITIAL {}
PARAMETER {
  g = 1
}
BREAKPOINT {
    LOCAL v_diff
    v_diff = v - v_peer
    i = (g*v_diff * exp( v_diff * v_diff  * (-0.01)))*0.8 + (g*v_diff)*0.2
    if (i != i) {
        i = 0
    }
}
```

B



Decor: define which mechanisms go where

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

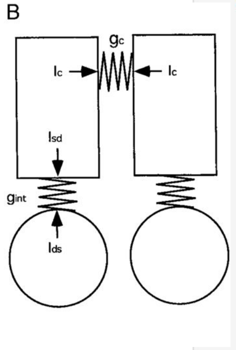
Single cell: Simulate a single voltage trace

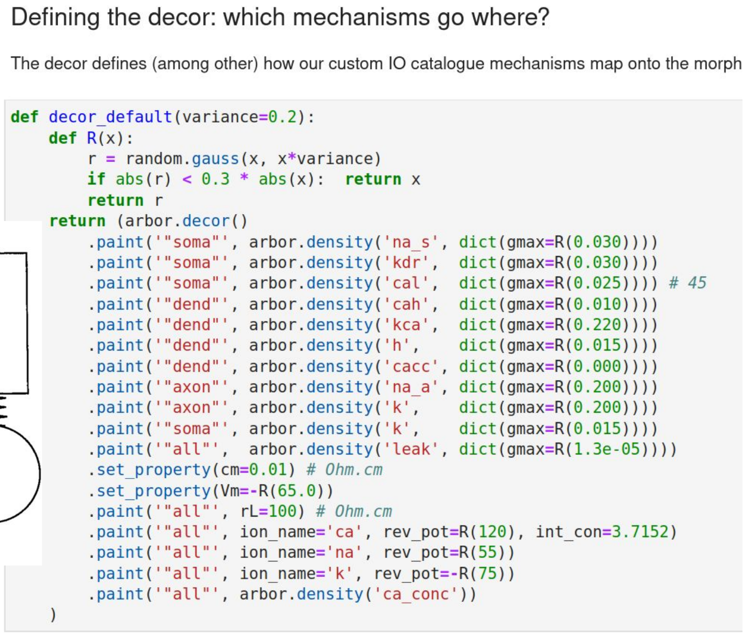
Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/





Defining the decor: which mechanisms go where?

The decor defines (among other) how our custom IO catalogue mechanisms map onto the morph

```
def decor_default(variance=0.2):
    def R(x):
        r = random.gauss(x, x*variance)
        if abs(r) < 0.3 * abs(x):  return x
        return r
    return (arbor.decor()
        .paint('"soma"', arbor.density('na_s', dict(gmax=R(0.030))))
        .paint('"soma"', arbor.density('kdr',  dict(gmax=R(0.030))))
        .paint('"soma"', arbor.density('cal',  dict(gmax=R(0.025)))) # 45
        .paint('"dend"', arbor.density('cah',  dict(gmax=R(0.010))))
        .paint('"dend"', arbor.density('kca',  dict(gmax=R(0.220))))
        .paint('"dend"', arbor.density('h',    dict(gmax=R(0.015))))
        .paint('"dend"', arbor.density('cacc', dict(gmax=R(0.000))))
        .paint('"axon"', arbor.density('na_a', dict(gmax=R(0.200))))
        .paint('"axon"', arbor.density('k',    dict(gmax=R(0.200))))
        .paint('"soma"', arbor.density('k',    dict(gmax=R(0.015))))
        .paint('"all"',  arbor.density('leak', dict(gmax=R(1.3e-05))))
        .set_property(cm=0.01) # Ohm.cm
        .set_property(Vm=-R(65.0))
        .paint('"all"', rL=100) # Ohm.cm
        .paint('"all"', ion_name='ca', rev_pot=R(120), int_con=3.7152)
        .paint('"all"', ion_name='na', rev_pot=R(55))
        .paint('"all"', ion_name='k', rev_pot=-R(75))
        .paint('"all"', arbor.density('ca_conc'))
    )
```

Recipe: putting everything together

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

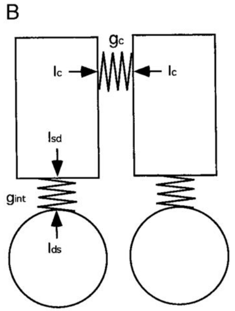
Single cell: Simulate a single voltage trace

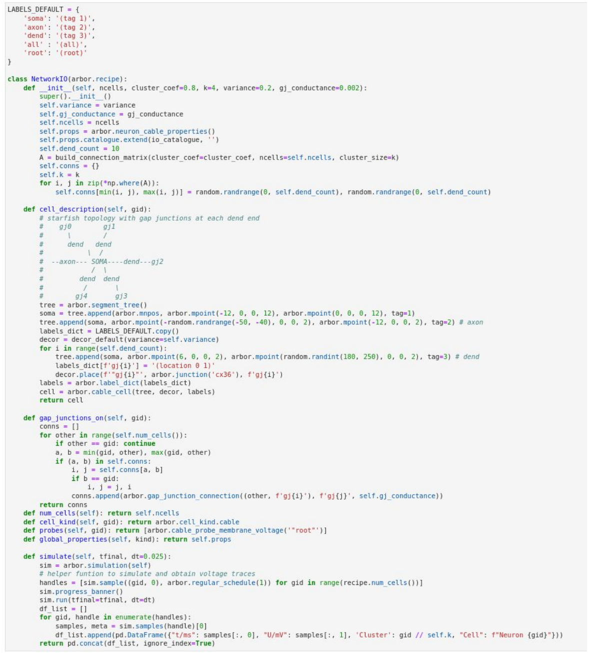
Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/





```python
LABELS_DEFAULT = {
    'soma': '(tag 1)',
    'axon': '(tag 2)',
    'dend': '(tag 3)',
    'all' : '(all)',
    'root': '(root)'
}

class NetworkIO(arbor.recipe):
    def __init__(self, ncells, cluster_coef=0.8, k=4, variance=0.2, gj_conductance=0.002):
        super().__init__()
        self.variance = variance
        self.gj_conductance = gj_conductance
        self.ncells = ncells
        self.props = arbor.neuron_cable_properties()
        self.props.catalogue.extend(io_catalogue, '')
        self.dend_count = 10
        A = build_connection_matrix(cluster_coef=cluster_coef, ncells=self.ncells, cluster_size=k)
        self.conns = {}
        self.k = k
        for i, j in zip(*np.where(A)):
            self.conns[min(i, j), max(i, j)] = random.randrange(0, self.dend_count), random.randrange(0, self.dend_count)

    def cell_description(self, gid):
        # starfish topology with gap junctions at each dend end
        #    gj0        gj1
        #      \        /
        #      dend   dend
        #           \  /
        #  --axon--- SOMA----dend---gj2
        #             /  \
        #         dend  dend
        #          /       \
        #        gj4       gj3
        tree = arbor.segment_tree()
        soma = tree.append(arbor.mnpos, arbor.mpoint(-12, 0, 0, 12), arbor.mpoint(0, 0, 0, 12), tag=1)
        tree.append(soma, arbor.mpoint(-random.randrange(-50, -40), 0, 0, 2), arbor.mpoint(-12, 0, 0, 2), tag=2) # axon
        labels_dict = LABELS_DEFAULT.copy()
        decor = decor_default(variance=self.variance)
        for i in range(self.dend_count):
            tree.append(soma, arbor.mpoint(6, 0, 0, 2), arbor.mpoint(random.randint(180, 250), 0, 0, 2), tag=3) # dend
            labels_dict[f'gj{i}'] = '(location 0 1)'
            decor.place(f'"gj{i}"', arbor.junction('cx36'), f'gj{i}')
        labels = arbor.label_dict(labels_dict)
        cell = arbor.cable_cell(tree, decor, labels)
        return cell

    def gap_junctions_on(self, gid):
        conns = []
        for other in range(self.num_cells()):
            if other == gid: continue
            a, b = min(gid, other), max(gid, other)
            if (a, b) in self.conns:
                i, j = self.conns[a, b]
                if b == gid:
                    i, j = j, i
                conns.append(arbor.gap_junction_connection((other, f'gj{i}'), f'gj{j}', self.gj_conductance))
        return conns
    def num_cells(self): return self.ncells
    def cell_kind(self, gid): return arbor.cell_kind.cable
    def probes(self, gid): return [arbor.cable_probe_membrane_voltage('"root"')]
    def global_properties(self, kind): return self.props

    def simulate(self, tfinal, dt=0.025):
        sim = arbor.simulation(self)
        # helper funtion to simulate and obtain voltage traces
        handles = [sim.sample((gid, 0), arbor.regular_schedule(1)) for gid in range(recipe.num_cells())]
        sim.progress_banner()
        sim.run(tfinal=tfinal, dt=dt)
        df_list = []
        for gid, handle in enumerate(handles):
            samples, meta = sim.samples(handle)[0]
            df_list.append(pd.DataFrame({"t/ms": samples[:, 0], "U/mV": samples[:, 1], 'Cluster': gid // self.k, "Cell": f"Neuron {gid}"}))
        return pd.concat(df_list, ignore_index=True)
```

Single cell: simulate a single voltage trace

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

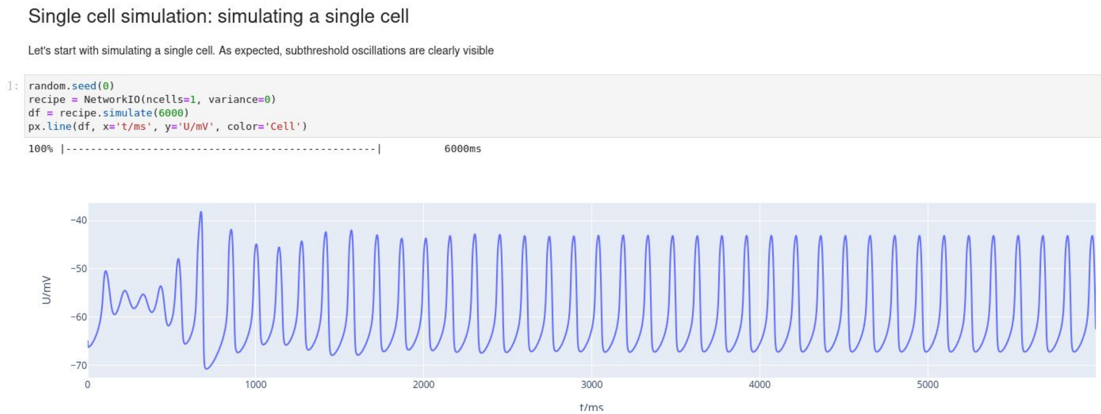
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Single cell simulation: simulating a single cell

Let's start with simulating a single cell. As expected, subthreshold oscillations are clearly visible

```
random.seed(0)
recipe = NetworkIO(ncells=1, variance=0)
df = recipe.simulate(6000)
px.line(df, x='t/ms', y='U/mV', color='Cell')
```

```
100% |--------------------------------------------------|         6000ms
```

Network: simulate an IO network (pt 1)

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

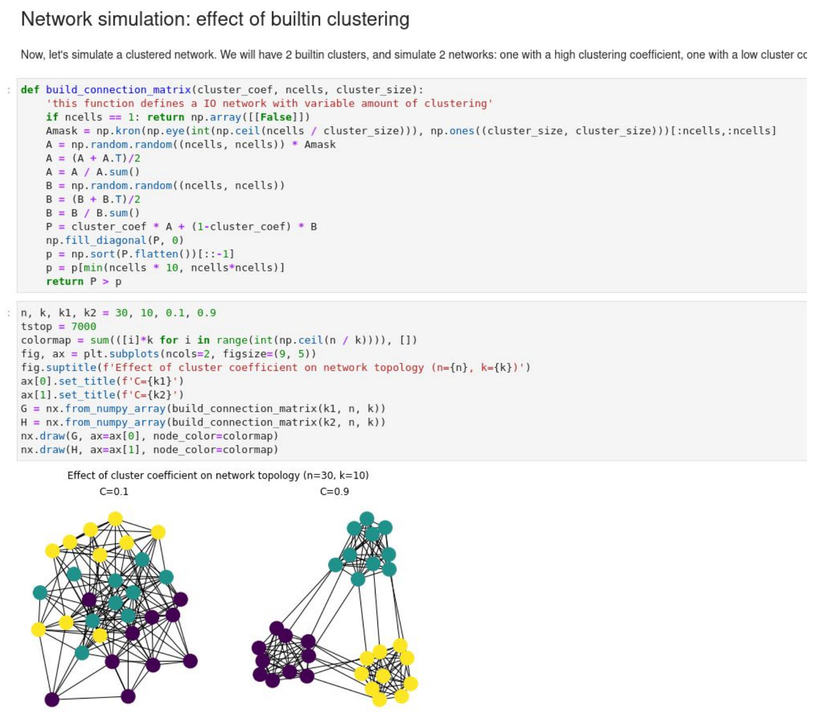
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Network simulation: effect of builtin clustering

Now, let's simulate a clustered network. We will have 2 builtin clusters, and simulate 2 networks: one with a high clustering coefficient, one with a low cluster cc

```
def build_connection_matrix(cluster_coef, ncells, cluster_size):
    'this function defines a IO network with variable amount of clustering'
    if ncells == 1: return np.array([[False]])
    Amask = np.kron(np.eye(int(np.ceil(ncells / cluster_size))), np.ones((cluster_size, cluster_size)))[:ncells,:ncells]
    A = np.random.random((ncells, ncells)) * Amask
    A = (A + A.T)/2
    A = A / A.sum()
    B = np.random.random((ncells, ncells))
    B = (B + B.T)/2
    B = B / B.sum()
    P = cluster_coef * A + (1-cluster_coef) * B
    np.fill_diagonal(P, 0)
    p = np.sort(P.flatten())[::-1]
    p = p[min(ncells * 10, ncells*ncells)]
    return P > p
```

```
n, k, k1, k2 = 30, 10, 0.1, 0.9
tstop = 7000
colormap = sum(([i]*k for i in range(int(np.ceil(n / k)))), [])
fig, ax = plt.subplots(ncols=2, figsize=(9, 5))
fig.suptitle(f'Effect of cluster coefficient on network topology (n={n}, k={k})')
ax[0].set_title(f'C={k1}')
ax[1].set_title(f'C={k2}')
G = nx.from_numpy_array(build_connection_matrix(k1, n, k))
H = nx.from_numpy_array(build_connection_matrix(k2, n, k))
nx.draw(G, ax=ax[0], node_color=colormap)
nx.draw(H, ax=ax[1], node_color=colormap)
```

Effect of cluster coefficient on network topology (n=30, k=10)
C=0.1
C=0.9

Network: simulate an IO network (pt 2)

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

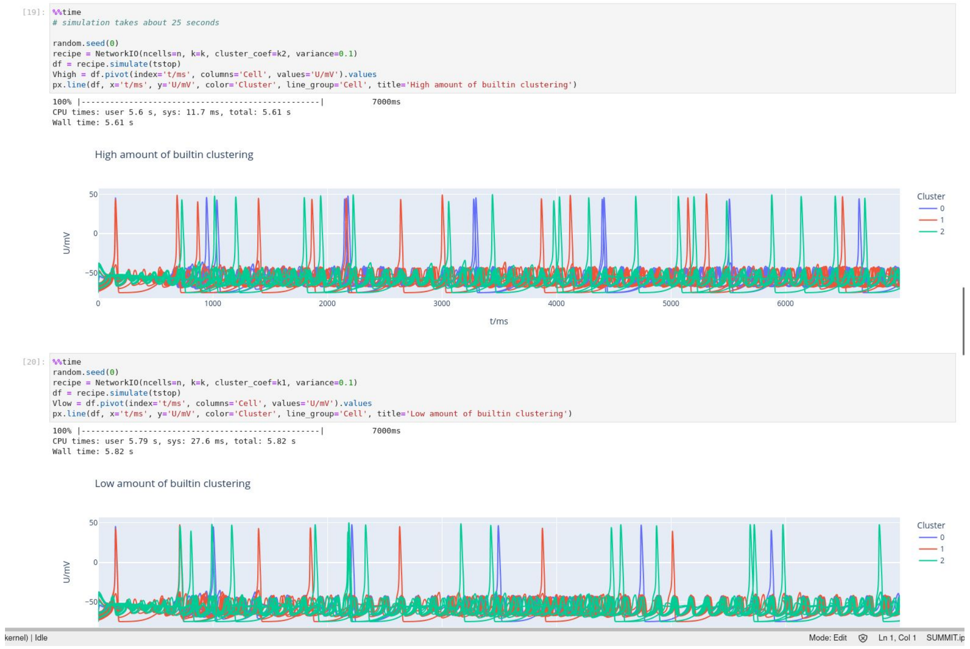
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



```
[19]: %%time
# simulation takes about 25 seconds

random.seed(0)
recipe = NetworkIO(ncells=n, k=k, cluster_coef=k2, variance=0.1)
df = recipe.simulate(tstop)
Vhigh = df.pivot(index='t/ms', columns='Cell', values='U/mV').values
px.line(df, x='t/ms', y='U/mV', color='Cluster', line_group='Cell', title='High amount of builtin clustering')
```

```
100% |--------------------------------------------------|          7000ms
CPU times: user 5.6 s, sys: 11.7 ms, total: 5.61 s
Wall time: 5.61 s
```

```
[20]: %%time
random.seed(0)
recipe = NetworkIO(ncells=n, k=k, cluster_coef=k1, variance=0.1)
df = recipe.simulate(tstop)
Vlow = df.pivot(index='t/ms', columns='Cell', values='U/mV').values
px.line(df, x='t/ms', y='U/mV', color='Cluster', line_group='Cell', title='Low amount of builtin clustering')
```

```
100% |--------------------------------------------------|          7000ms
CPU times: user 5.79 s, sys: 27.6 ms, total: 5.82 s
Wall time: 5.82 s
```

Visualize: visualize cluster dynamics (pt 1)

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

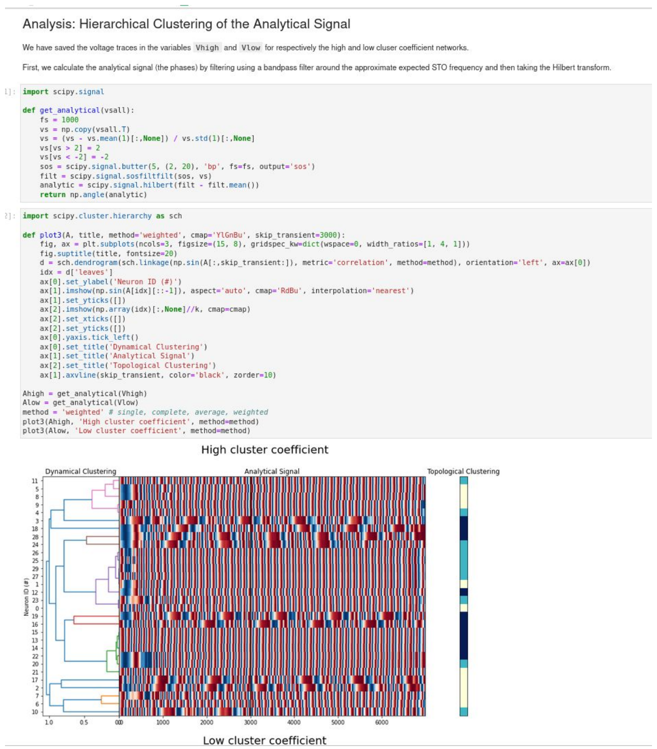
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Analysis: Hierarchical Clustering of the Analytical Signal

We have saved the voltage traces in the variables `Vhigh` and `Vlow` for respectively the high and low cluser coefficient networks.

First, we calculate the analytical signal (the phases) by filtering using a bandpass filter around the approximate expected STO frequency and then taking the Hilbert transform.

```
import scipy.signal

def get_analytical(vsall):
    fs = 1000
    vs = np.copy(vsall.T)
    vs = (vs - vs.mean(1)[:,None]) / vs.std(1)[:,None]
    vs[vs > 2] = 2
    vs[vs < -2] = -2
    sos = scipy.signal.butter(5, (2, 20), 'bp', fs=fs, output='sos')
    filt = scipy.signal.sosfiltfilt(sos, vs)
    analytic = scipy.signal.hilbert(filt - filt.mean())
    return np.angle(analytic)
```

```
import scipy.cluster.hierarchy as sch

def plot3(A, title, method='weighted', cmap='YlGnBu', skip_transient=3000):
    fig, ax = plt.subplots(ncols=3, figsize=(15, 8), gridspec_kw=dict(wspace=0, width_ratios=[1, 4, 1]))
    fig.suptitle(title, fontsize=20)
    d = sch.dendrogram(sch.linkage(np.sin(A[:,skip_transient:]), metric='correlation', method=method), orientation='left', ax=ax[0])
    idx = d['leaves']
    ax[0].set_ylabel('Neuron ID (#)')
    ax[1].imshow(np.sin(A[idx][::-1]), aspect='auto', cmap='RdBu', interpolation='nearest')
    ax[1].set_yticks([])
    ax[2].imshow(np.array(idx)[:,None]//k, cmap=cmap)
    ax[2].set_xticks([])
    ax[2].set_yticks([])
    ax[0].yaxis.tick_left()
    ax[0].set_title('Dynamical Clustering')
    ax[1].set_title('Analytical Signal')
    ax[2].set_title('Topological Clustering')
    ax[1].axvline(skip_transient, color='black', zorder=10)

Ahigh = get_analytical(Vhigh)
Alow = get_analytical(Vlow)
method = 'weighted' # single, complete, average, weighted
plot3(Ahigh, 'High cluster coefficient', method=method)
plot3(Alow, 'Low cluster coefficient', method=method)
```

Visualize: visualize cluster dynamics (pt 2)

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

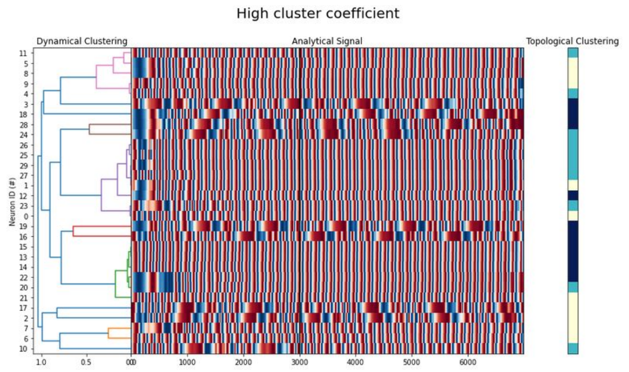
Single cell: Simulate a single voltage trace

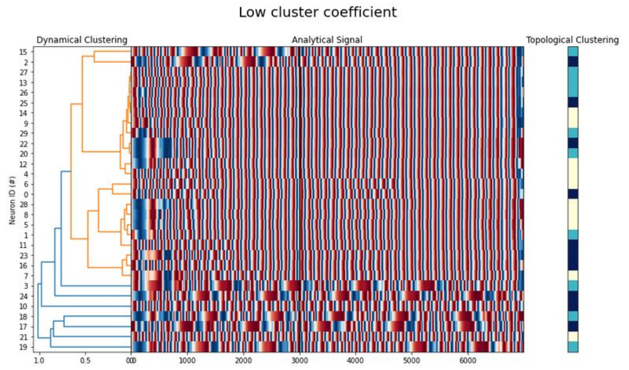
Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/





Quantify: effect topology on synchrony

Setup: Import and git clone

Mechanisms: Compile & load NMODL files

Decor: Define which mechanisms go where

Recipe: Putting everything together

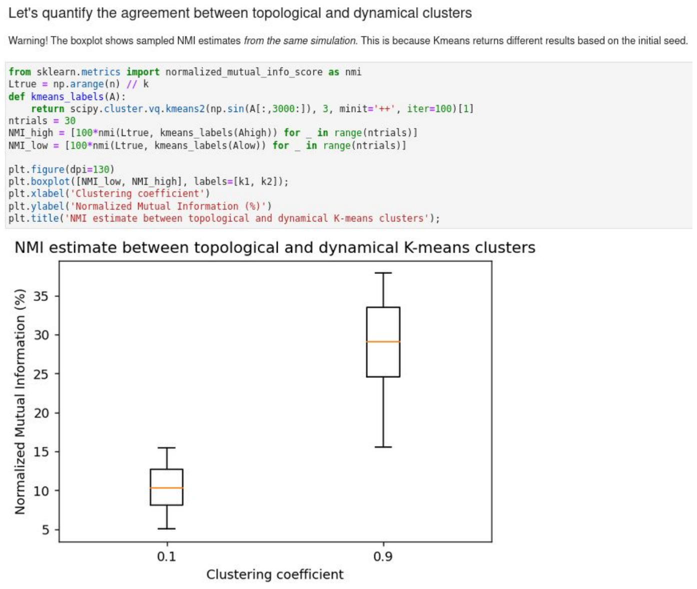
Single cell: Simulate a single voltage trace

Network: Simulate an IO network

Visualize: Visualize cluster dynamics

Quantify: Effect topology->synchrony

wiki.ebrains.eu/bin/view/Collabs/io-clusters/



Let's quantify the agreement between topological and dynamical clusters

Warning! The boxplot shows sampled NMI estimates *from the same simulation*. This is because Kmeans returns different results based on the initial seed.

```
from sklearn.metrics import normalized_mutual_info_score as nmi
Ltrue = np.arange(n) // k
def kmeans_labels(A):
    return scipy.cluster.vq.kmeans2(np.sin(A[:,3000:]), 3, minit='++', iter=100)[1]
ntrials = 30
NMI_high = [100*nmi(Ltrue, kmeans_labels(Ahigh)) for _ in range(ntrials)]
NMI_low = [100*nmi(Ltrue, kmeans_labels(Alow)) for _ in range(ntrials)]

plt.figure(dpi=130)
plt.boxplot([NMI_low, NMI_high], labels=[k1, k2]);
plt.xlabel('Clustering coefficient')
plt.ylabel('Normalized Mutual Information (%)')
plt.title('NMI estimate between topological and dynamical K-means clusters');
```

NMI estimate between topological and dynamical K-means clusters
Normalized Mutual Information (%)
35
30
25
20
15
10
5
0.1
0.9
Clustering coefficient

Questions? No time!

Come talk to me at my poster