The Nengo neural simulator

Trevor Bekolay

What is Nengo?

- A neural simulator (SNNs, LIFs, STDP, …)
- A machine learning platform (DNNs, Tanh, backprop, …)
- A neuromorphic hardware SDK (Loihi, SpiNNaker, …)
- A robot control SDK (MuJoCo sim, Kinova Jaco arm, …)
- …

Nengo’s goal is to use neural networks to perform intelligent functions efficiently

Nengo is an ecosystem of tools

Simulated robot arm control

Physical robot arm control

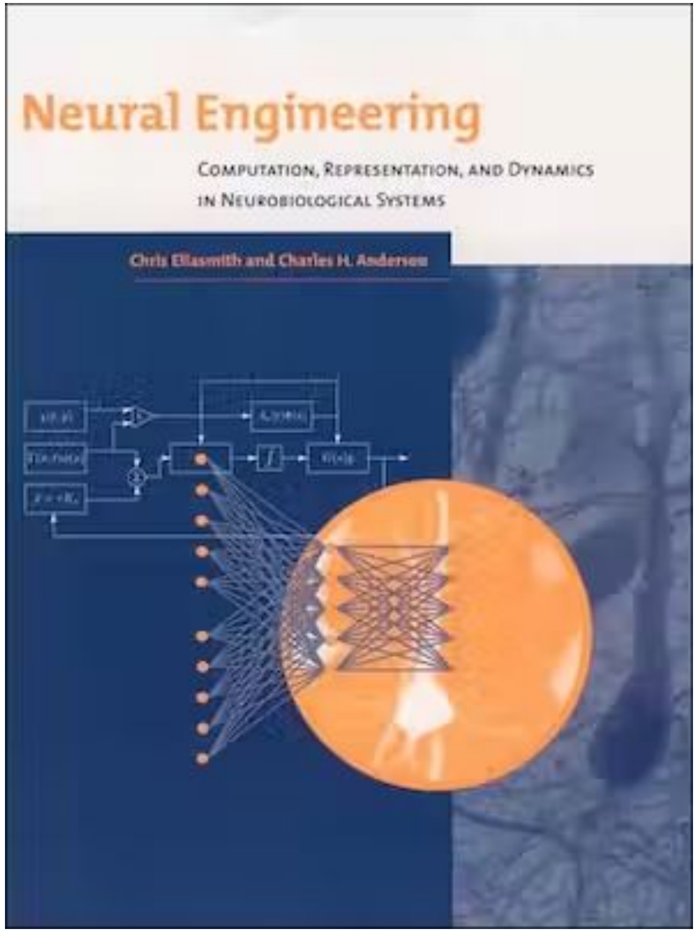
Autonomous drone control

Image classification

Keyword spotting

Interactive GUI

History: The Matlab years (2003)



Charles H. Anderson

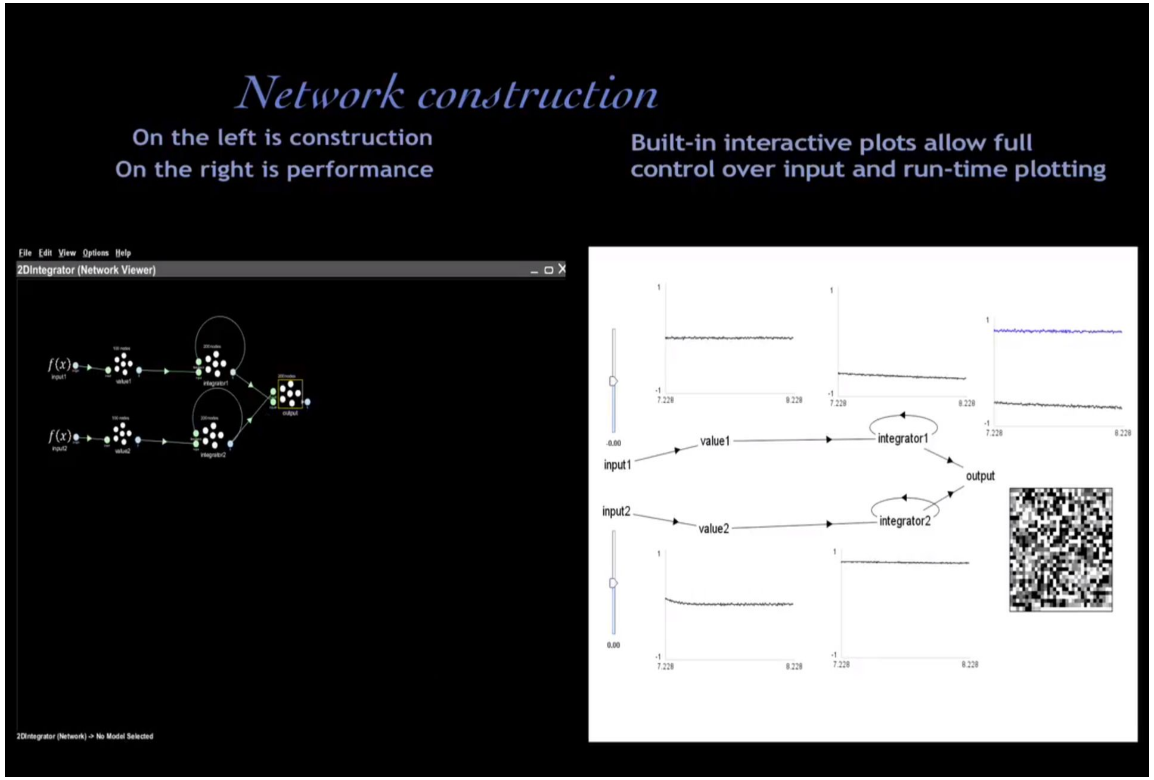
Chris Eliasmith

[NESim]

Bryan Tripp

[Nemo]

History: The Java years (2007)



[NEO → Nengo 1.4]

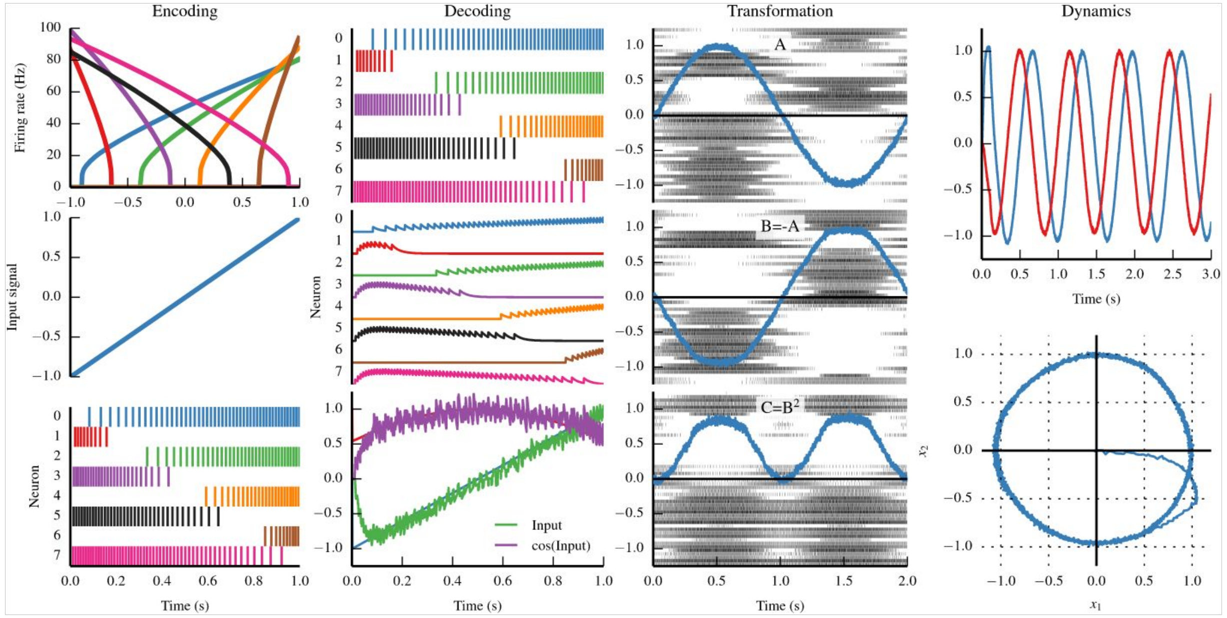
Bryan Tripp

Shu Wu

Terry Stewart

History: The Python years! (2013)

Nengo



[Nengo 2.0+]()

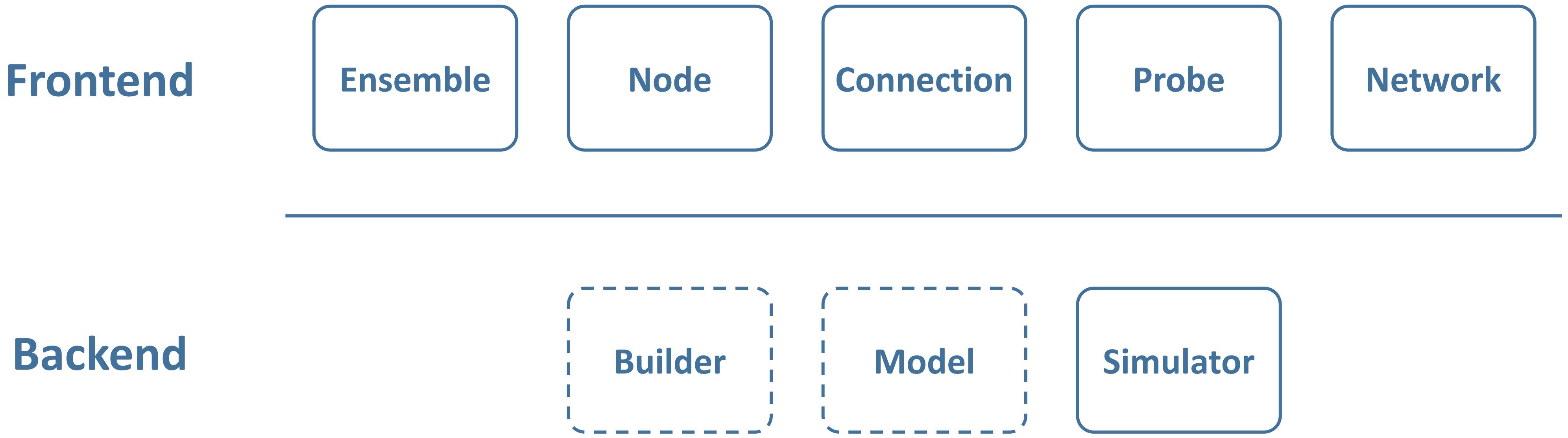
Trevor Bekolay 👋

James Bergstra

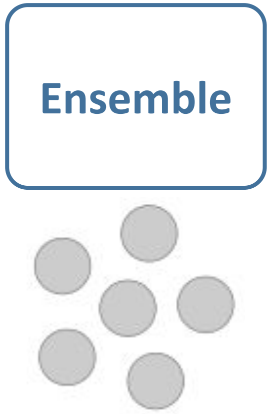
Eric Hunsberger

[…and many more]()

Nengo Architecture



Frontend: Ensemble

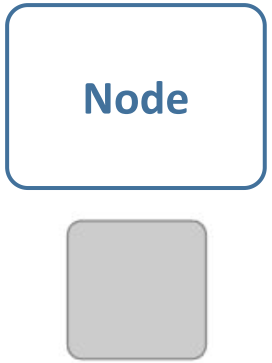


A population of neurons.

- `n_neurons`
- *`neuron_type`* `= LIF()`
- *`noise`* `= processes.WhiteNoise()`

```
lif = nengo.Ensemble(n_neurons=100, dimensions=1)
poisson = nengo.Ensemble(
    n_neurons=100,
    dimensions=1,
    neuron_type=nengo.PoissonSpiking(nengo.Tanh()),
)
```

Frontend: Node



Provide non-neural inputs, run non-neural functions, route signals, connect to external processes/devices.

- `output = None, array-like, function`
- *`size_in, size_out`*

```
const = nengo.Node([0, 0])
t_func = nengo.Node(lambda t: np.sin(t))
inp_func = nengo.Node (lambda t, x: x[0] * x[1])
passthrough = nengo.Node(None, size_in=3)
```

Frontend: Connection

Connects two object together.

- pre, post
- *synapse* = Lowpass(0.01), None
- *transform* = Dense, Sparse, Convolution

```
stim = nengo.Connection(node, ens.neurons[:2], transform=[1, -1])
nengo.Connection(ens.neurons, ens.neurons, synapse=0.2)
```

Frontend: Probe

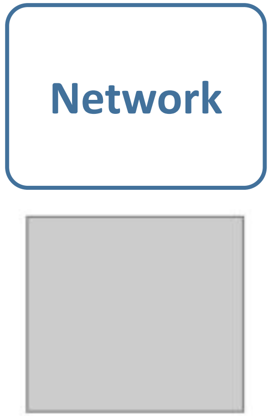
Probe

Collects data from a simulation.

- target
- *attr* = “input”, “weights”
- *sample_every* = 0.005
- *synapse* = Lowpass(0.01), None

```
probe = nengo.Probe(node, synapse=None)
filt_probe = nengo.Probe(ens.neurons)
conn_probe = nengo.Probe(conn, attr=”weights”, sample_every=0.1)
```

Frontend: Network



Container for frontend objects, including other networks.

- *label* = None, "M1"
- *seed* = 10

```
with nengo.Network(label="Vision") as vision:
    ...
```

Backend: Simulator

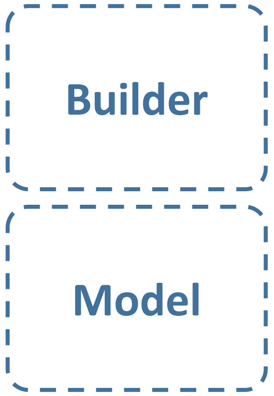
Simulator

Interface for running a simulation and collecting data.
Reference simulator uses NumPy.

- network
- *dt* = 0.001
- *seed* = 0.005

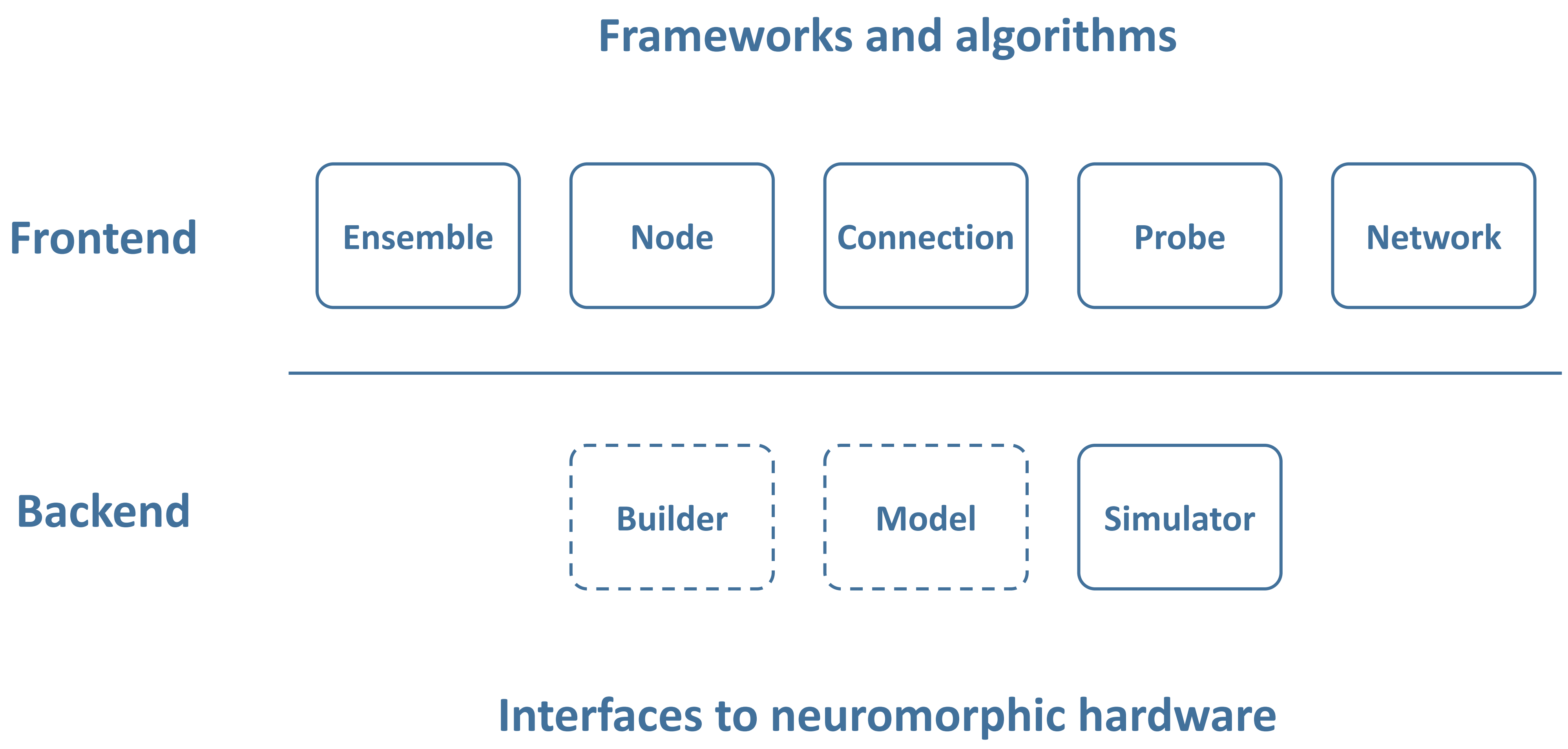
```
with nengo.Simulator(network) as sim:
    sim.run(0.1)
plt.plot(sim.trange(), sim.data[probe])
```

Backend: Model and Builder

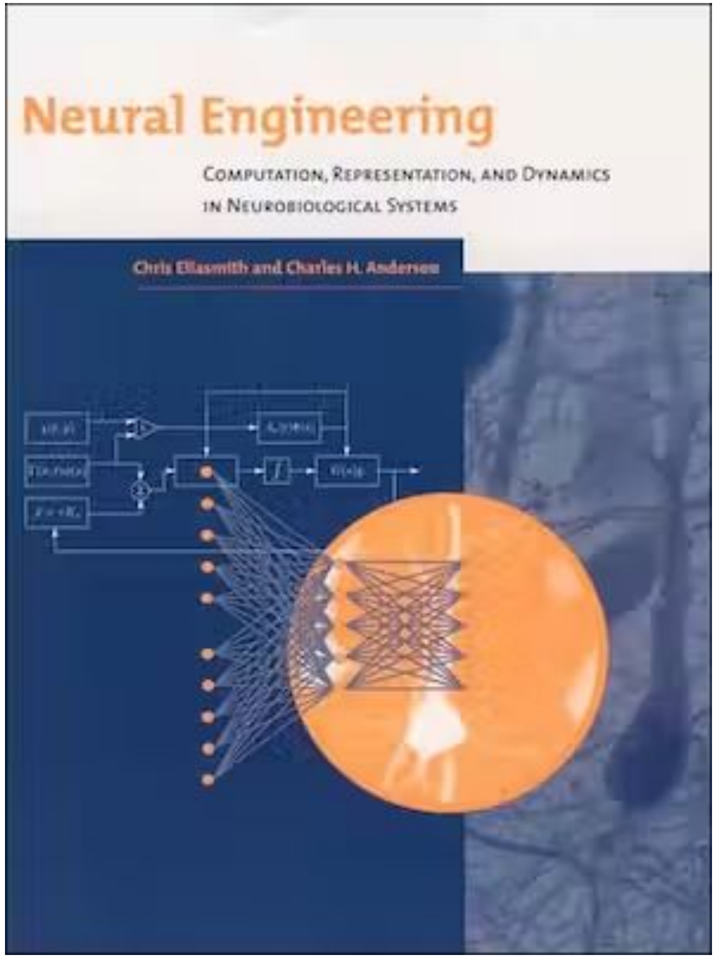


The reference build process generates a collection of Signals and Operations from the network.

```
@Builder.register(nengo.Ensemble)
def build_ensemble(model, ens):
    model.sig[ens]["in"] = Signal(shape=ens.n_neurons)
    model.add_op(Reset(model.sig[ens]["in"]))
    ...
```


Frameworks and algorithms
Frontend
Ensemble
Node
Connection
Probe
Network
Backend
Builder
Model
Simulator
Interfaces to neuromorphic hardware

The Neural Engineering Framework



**Nengo's goal is to
use neural networks
to perform intelligent functions
efficiently**

NEF Principle 1: Representation

Ensemble

A population of neurons represents a vector.

- `dimensions`
- *`radius`* `= 1`
- *`encoders`* `= Distribution, array-like`
- *`intercepts`* `= Distribution, array-like`
- *`max_rates`* `= Distribution, array-like`

Example: Many neurons

Aside: Neurons could be a first class object

NEF Principle 2: Transformation

Connection

Probe

Non-linear transformations of a vector can be decoded and projected to other neural populations.

- `function = lambda x: x[0]*x[1], array-like`
- `solver = solvers.LstsqL2()`
- `eval_points = int, array-like`

Example: Multiplication

NEF Principle 3: Dynamics

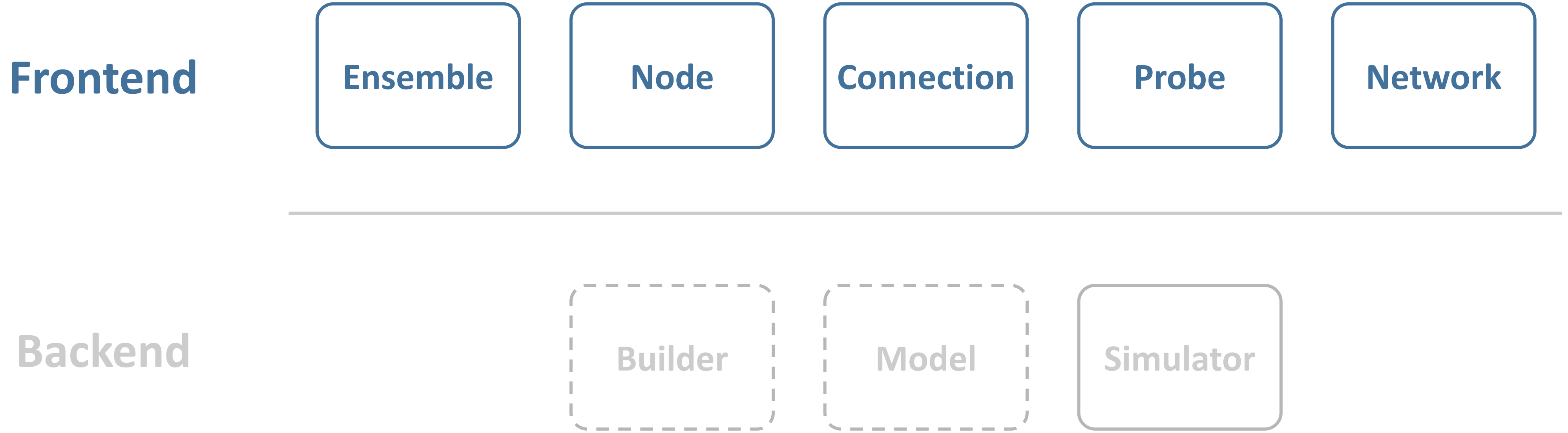
Connection

Non-linear dynamical systems can be implemented with recurrent connections.

Example: Memory (integrator)

Example: Oscillators

Frontend ecosystem



Included networks

EnsembleArray: Splits a high-dimensional ensemble into lower-dimensional sub-ensembles. (SPA parser example)

Product: Precisely computes the element-wise product of two equally sized vectors. (whitepaper)

nengo.ai/nengo/networks.html

NengoSPA

1. Symbols are associated with a high-dimensional vector (pointer)
2. Superposition: $P1 + P2$
3. Binding: $P1 \circledast P2 = P3$

   Unbinding: $P3 \circledast P1^{+} = P2 + \text{noise}$

[Spaun (2013)] [Spaun (2021)]

nengo.ai/nengo-spa

Gyrus

Build Nengo models with NumPy syntax

github.com/nengo-labs/nengo-gyrus

Gyrus

Outer product in Nengo

```
with nengo.Network() as model:
    stims = [nengo.Node(u_i) for u_i in u]
    probes = np.empty((len(u), len(u)), dtype=object)
    for i in range(len(u)):
        for j in range(len(u)):
            product = nengo.networks.Product(n_neurons=200, dimensions=1)
            nengo.Connection(stims[i], product.input_a, synapse=None)
            nengo.Connection(stims[j], product.input_b, synapse=None)
            probes[i, j] = nengo.Probe(product.output, synapse=0.005)
```

```
with nengo.Simulator(model) as sim:
    sim.run(0.1)

out = np.asarray(
    [
        [sim.data[probes[i, j]].squeeze(axis=-1) for j in range(len(u))]
        for i in range(len(u))
    ]
)
```

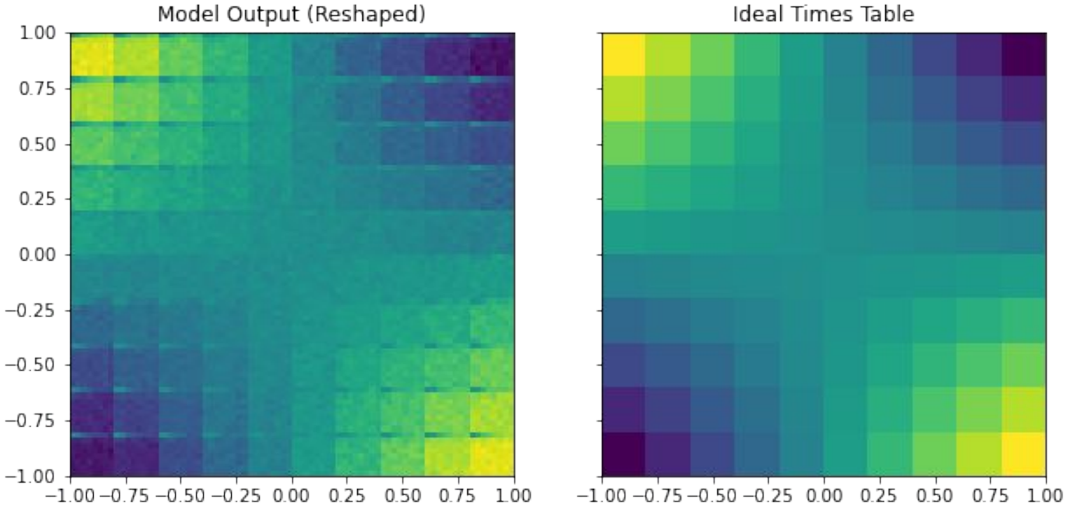
Gyrus

Outer product in NengoGyrus

```python
import gyrus

def times_table(u, tau=0.005):
    x = gyrus.stimuli(u)
    return np.outer(x, x).filter(tau)

out = np.asarray(times_table(u).run(0.1)).squeeze(axis=-1)
```

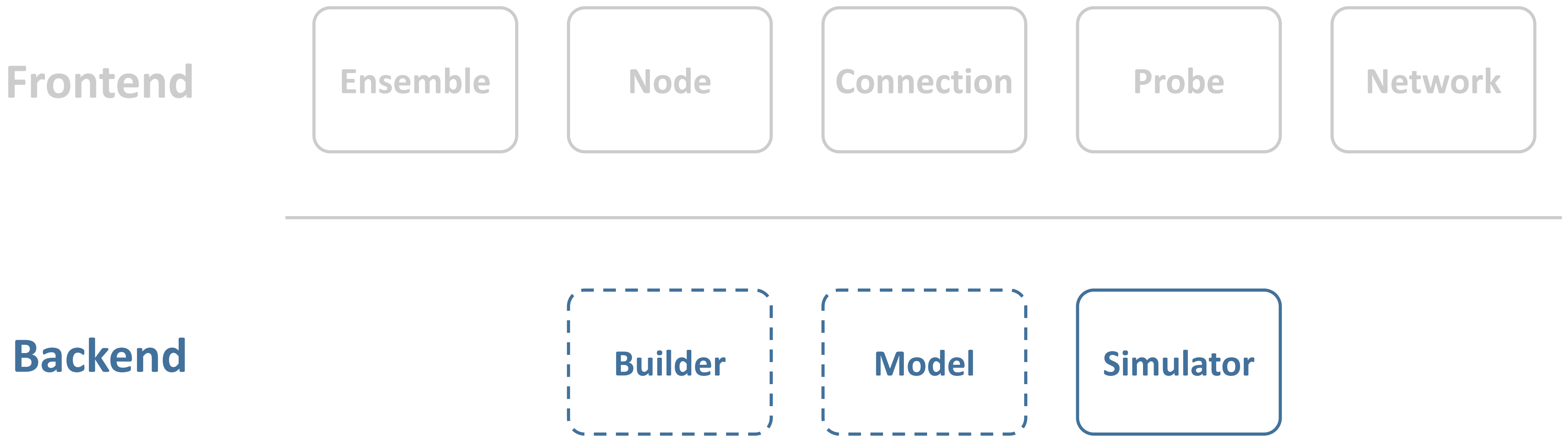


NengoExtras

nengolib

NengoExamples

Backend ecosystem



NengoLoihi

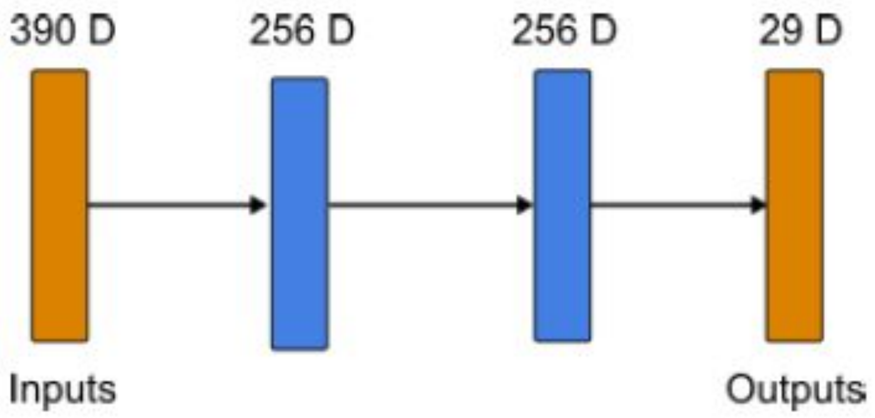

```
nengo.Simulator(model) → nengo_loihi.Simulator(model)
```

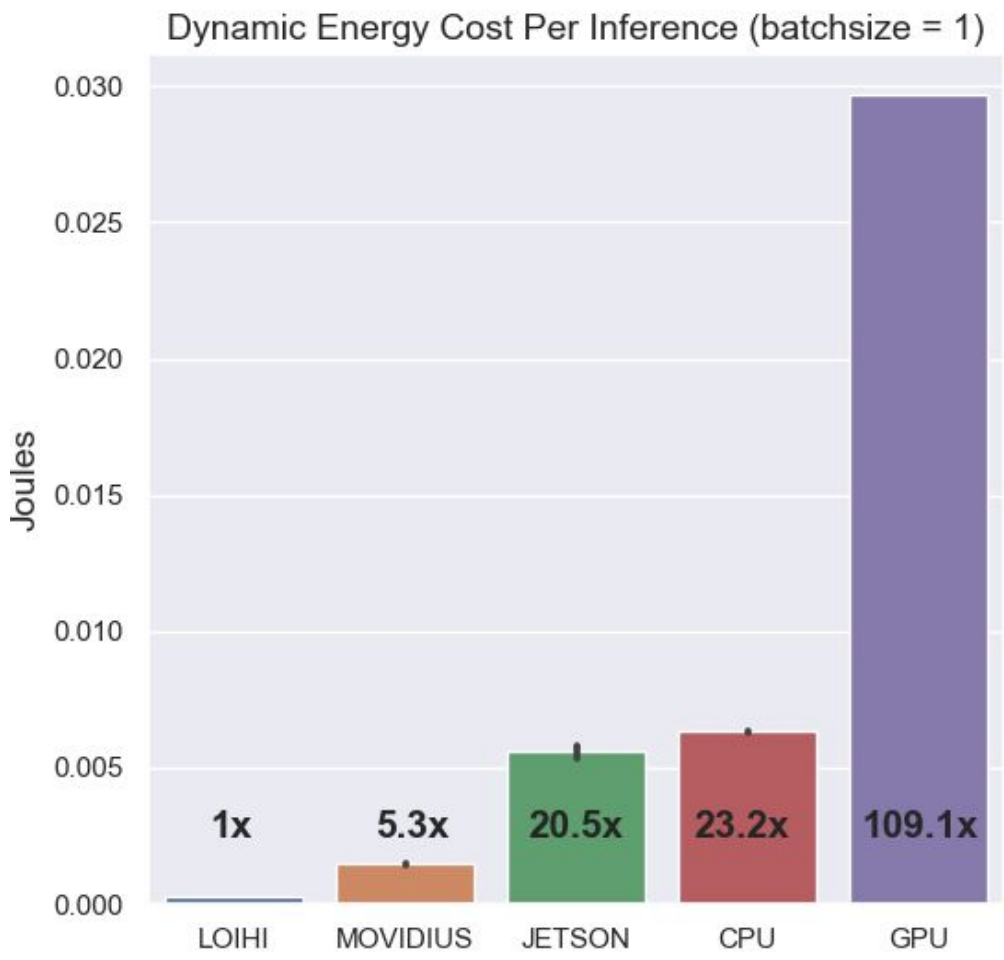
Can target real hardware or our included emulator

nengo.ai/nengo-loihi

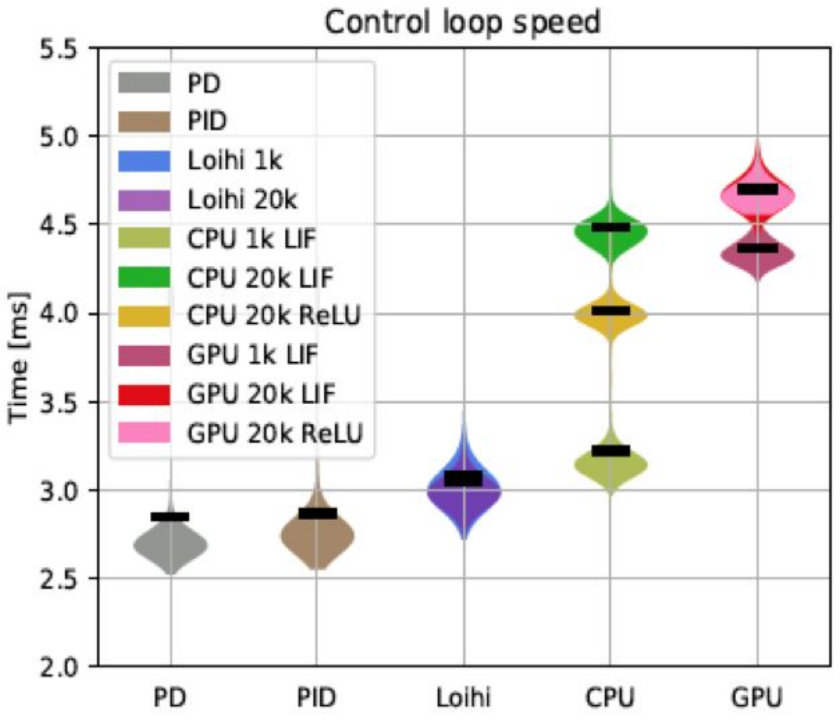
NengoLoihi

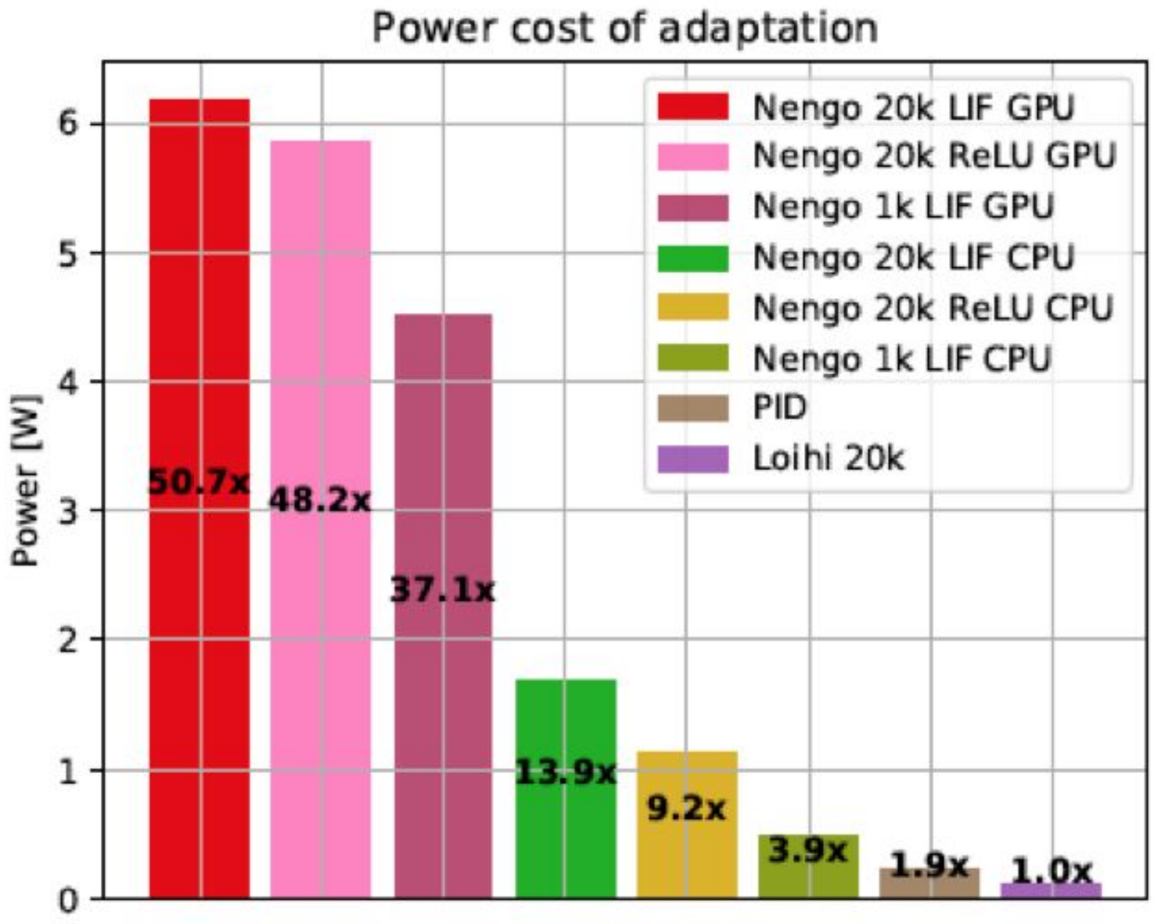
- **Benchmark [keyword spotting] model on CPU, GPU, Jetson, Movidius, and Loihi**
  - Identical data, network topologies
  - Non-spiking accuracy: 92.7%
  - Spiking accuracy: 93.8%



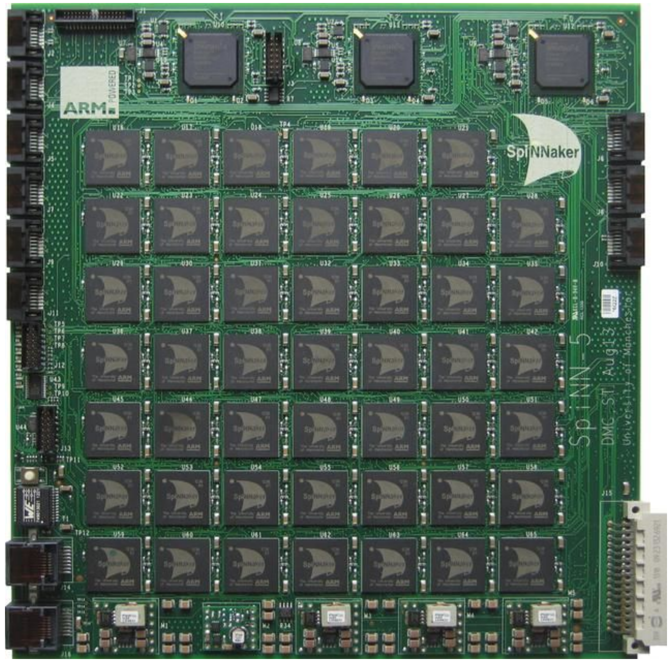


- **Benchmark arm control model**
  - 30% faster per timestep
  - 10-50x less power than CPU/GPU





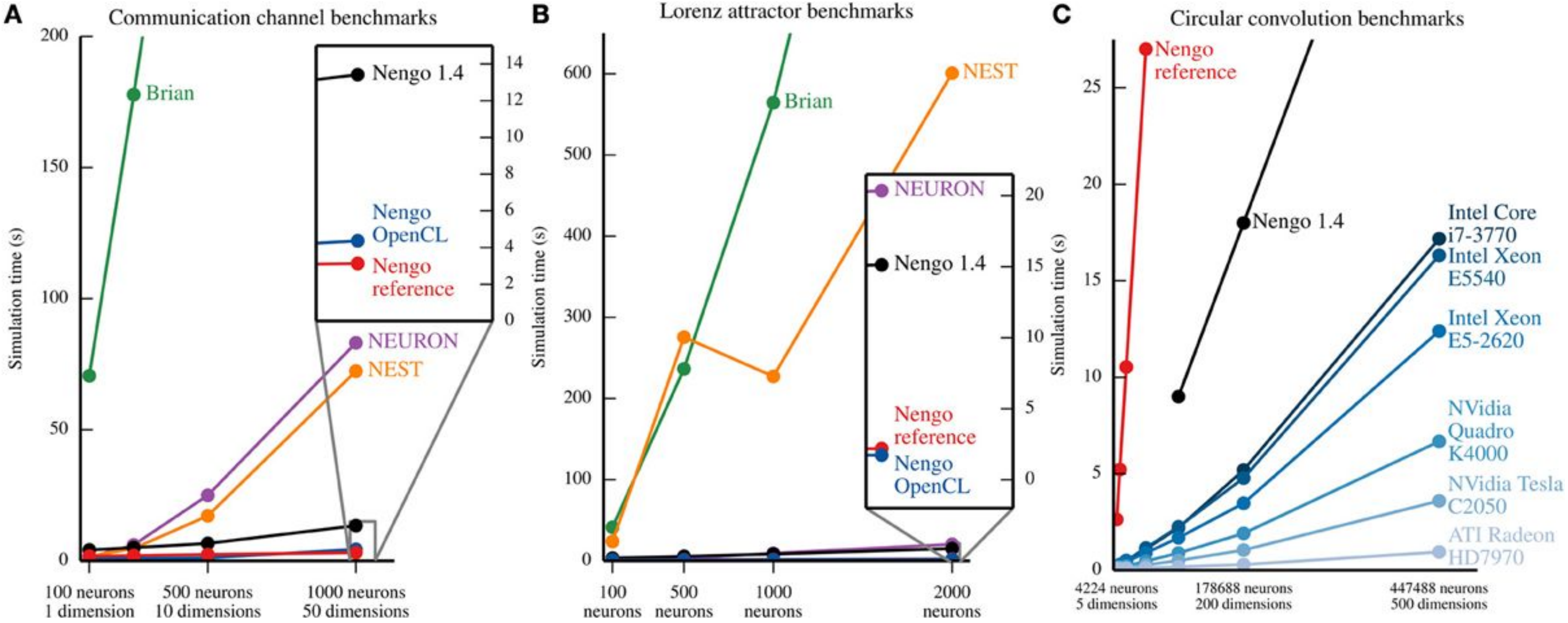
NengoSpiNNaker



`nengo.Simulator(model) → nengo_spinnaker.Simulator(model)`

github.com/project-rig/nengo_spinnaker

NengoOCL



`nengo.Simulator(model) → nengo_ocl.Simulator(model)`

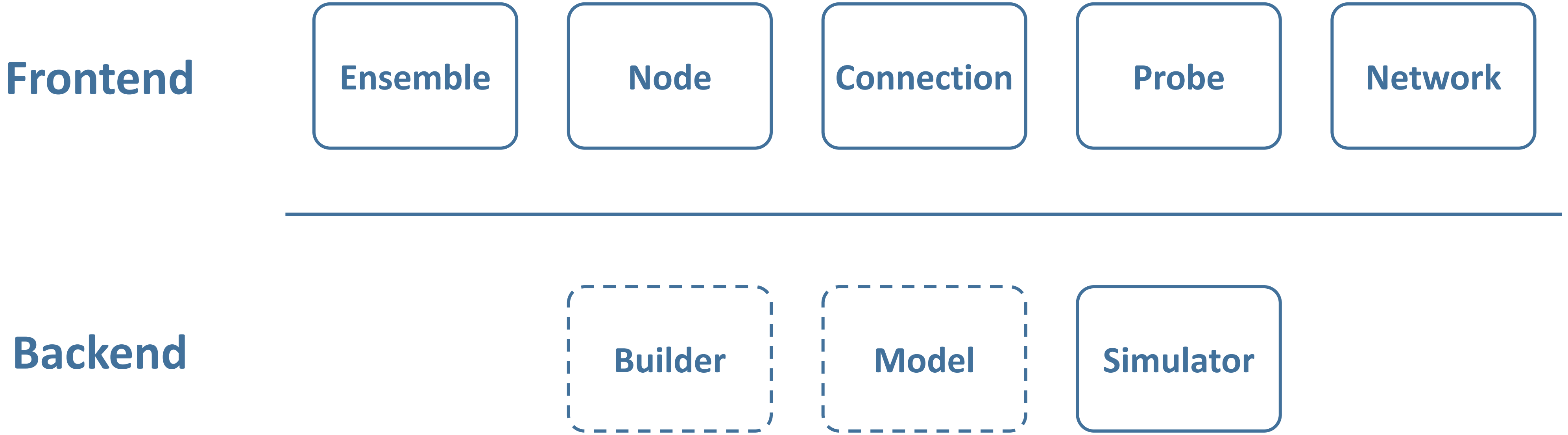
labs.nengo.ai/nengo-ocl

NengoFPGA

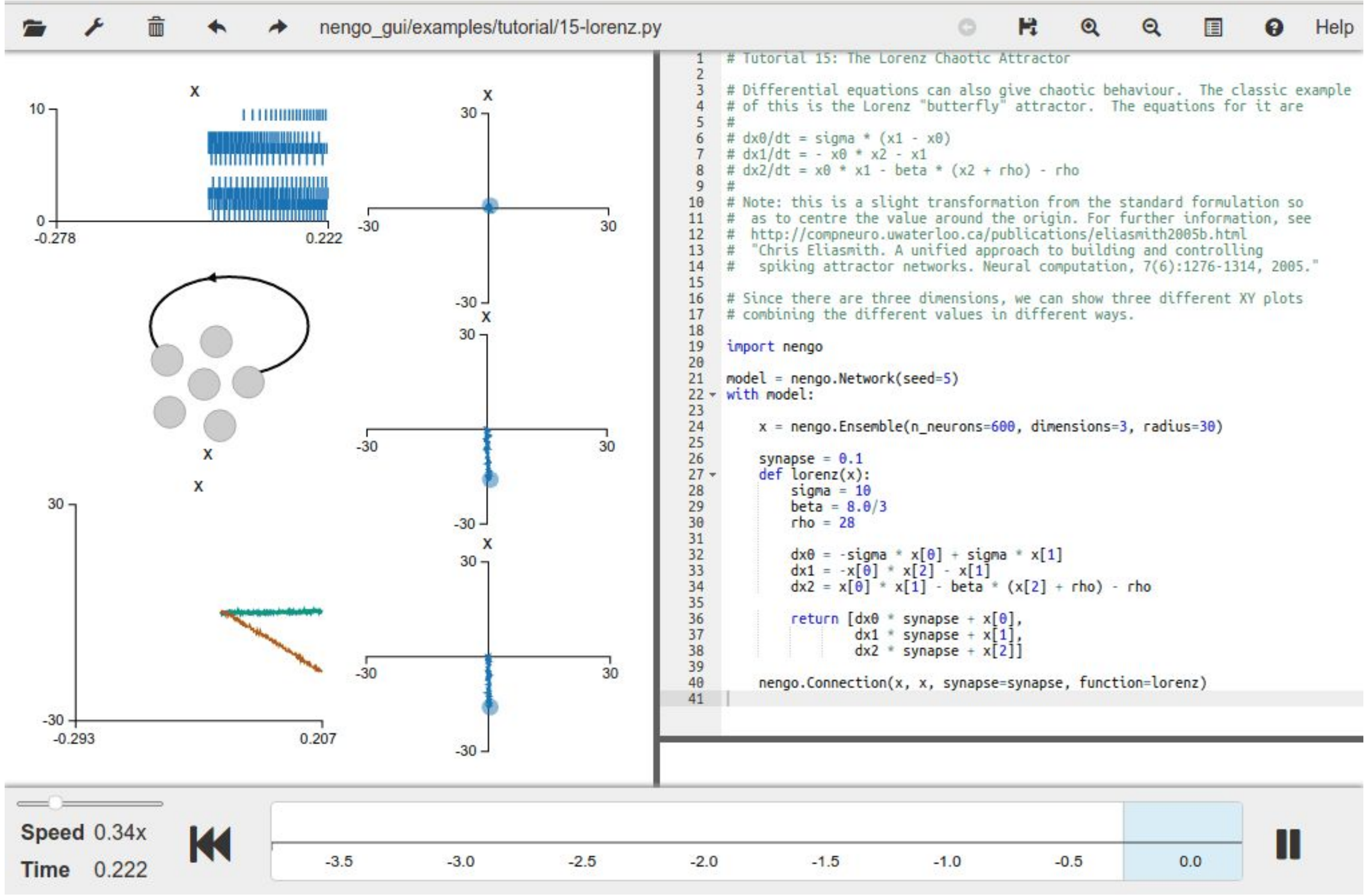
NengoBraindrop

NengoMPI

Other parts of the ecosystem

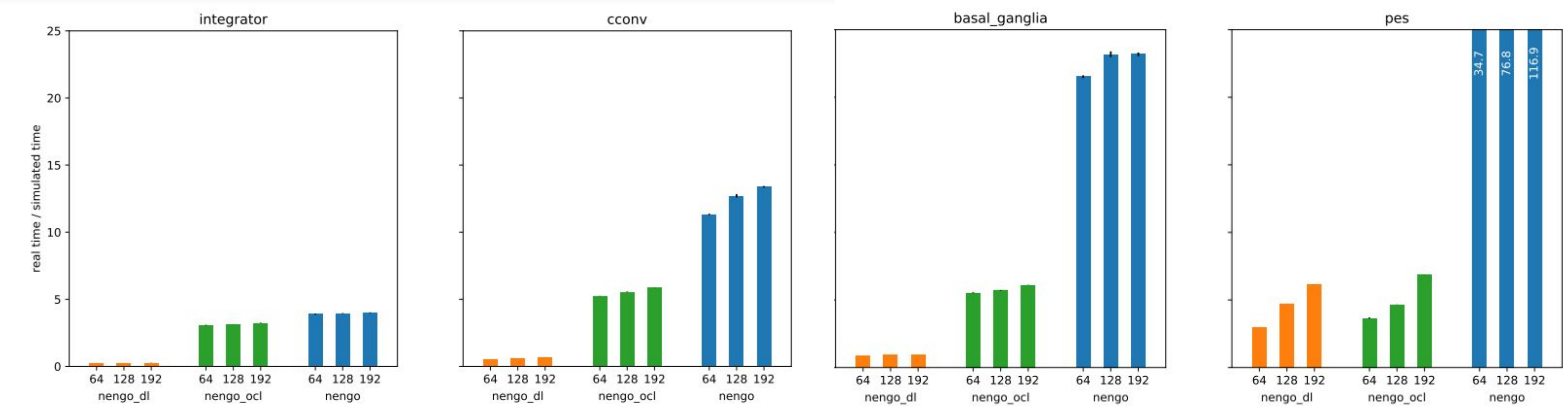


Nengo GUI



- Python server
- HTML / JS client
- Websockets
- D3.js

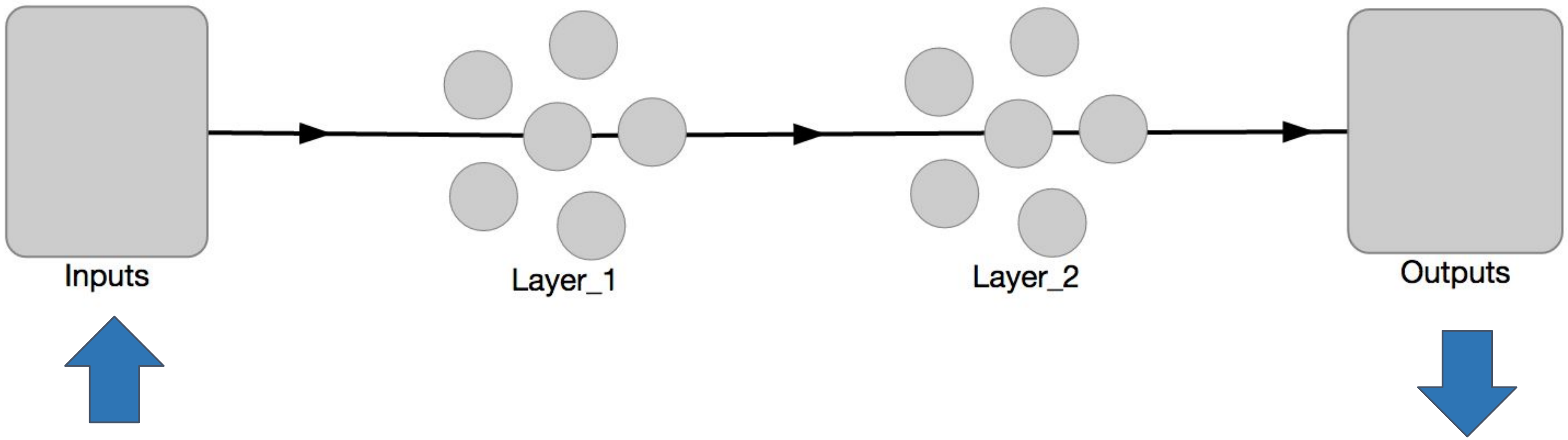
NengoDL



`nengo.Simulator(model) → nengo_dl.Simulator(model)`

nengo.ai/nengo-dl

NengoDL

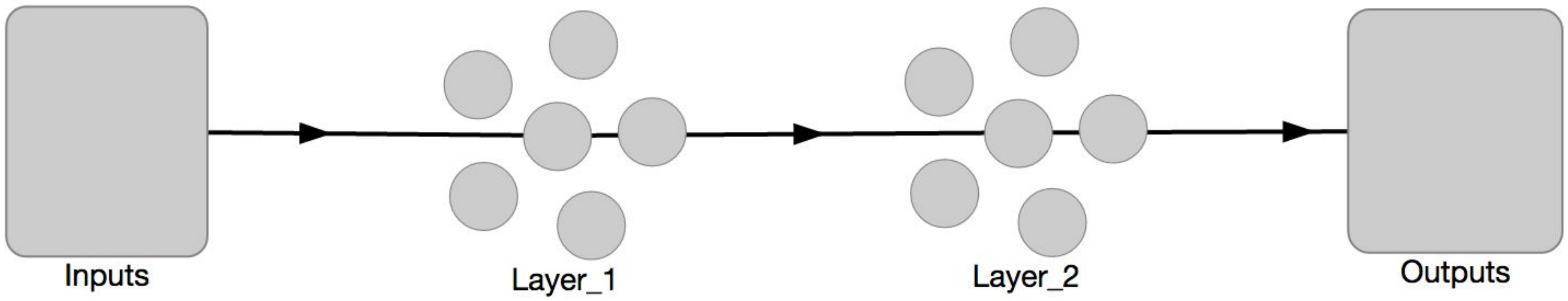


Feed features
to Nengo Node

Probe outputs and
compare to targets

Nengo DL will optimize all intermediate network parameters!

NengoDL



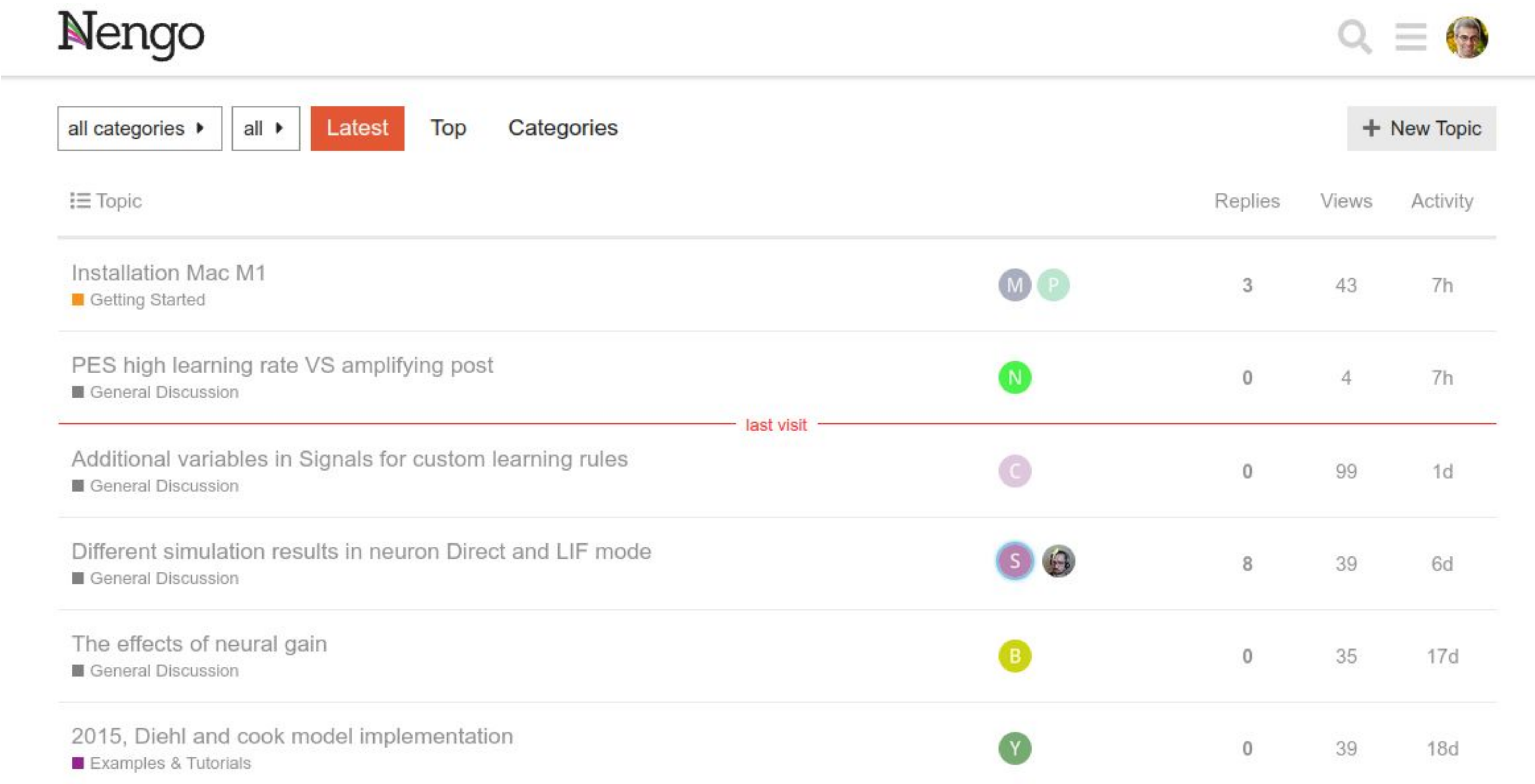

```python
import tensorflow as tf

with nengo_dl.Simulator(net, minibatch_size=10) as sim:
    sim.train(data={inputs: train_inputs, outputs: train_outputs},
              optimizer=tf.train.AdamOptimizer(),
              n_epochs=10, objective='mse')
```

NengoDL

[Embed a Keras model in a Nengo model with TensorNode]

[Convert a Keras model to Nengo objects with Converter]



forum.nengo.ai

June 4th - June 16th, 2023 at UWaterloo
Applications open! nengo.ai/summer-school

Licensing

- Nengo’s source is public
- Free for non-commercial use
- Commercial licenses can be purchased from ABR

Thanks! Questions?