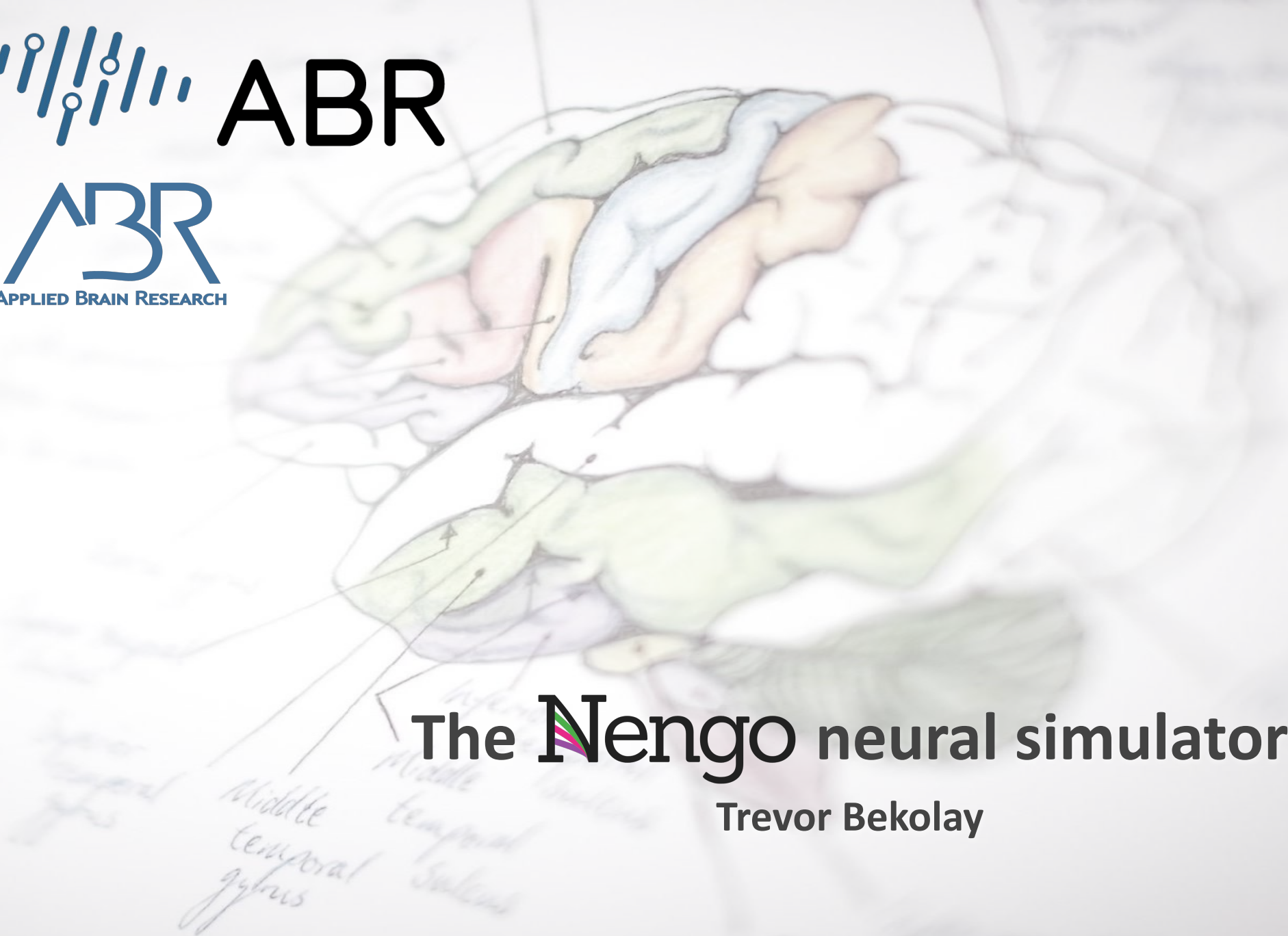




# ABR

**ABR**  
APPLIED BRAIN RESEARCH



# 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](#)

[Image classification](#)

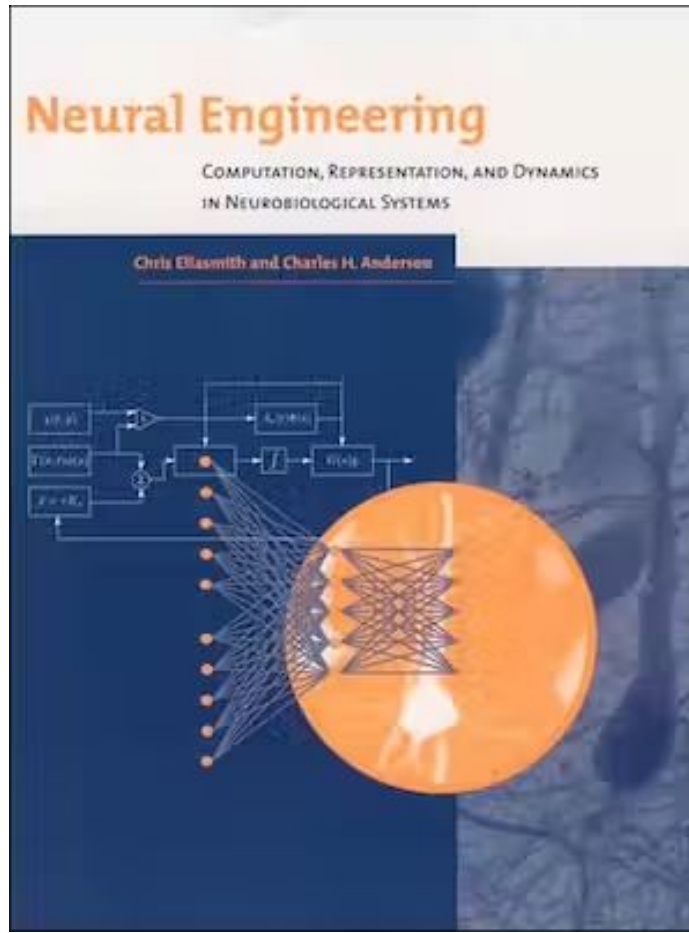
[Physical robot arm control](#)

[Keyword spotting](#)

[Autonomous drone control](#)

[Interactive GUI](#)

# History: The Matlab years (2003)



Charles H. Anderson



Chris Eliasmith

NESim



Bryan Tripp

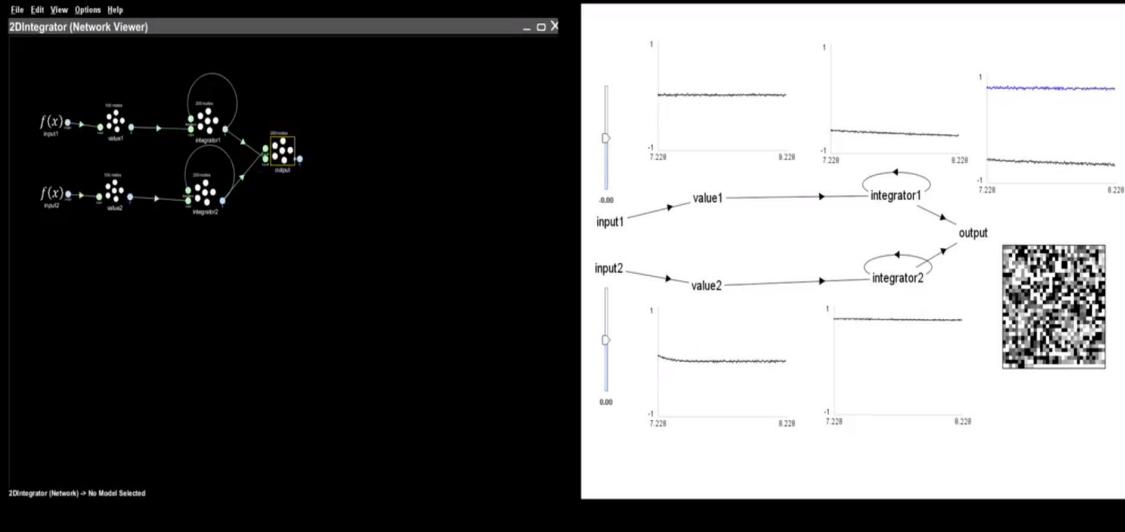
Nemo

# History: The Java years (2007)

*Network construction*

On the left is construction  
On the right is performance

Built-in interactive plots allow full control over input and run-time plotting



2DIntegrator (Network Viewer)

2DIntegrator (Network) - No Model Selected



Bryan Tripp



Shu Wu



Terry Stewart

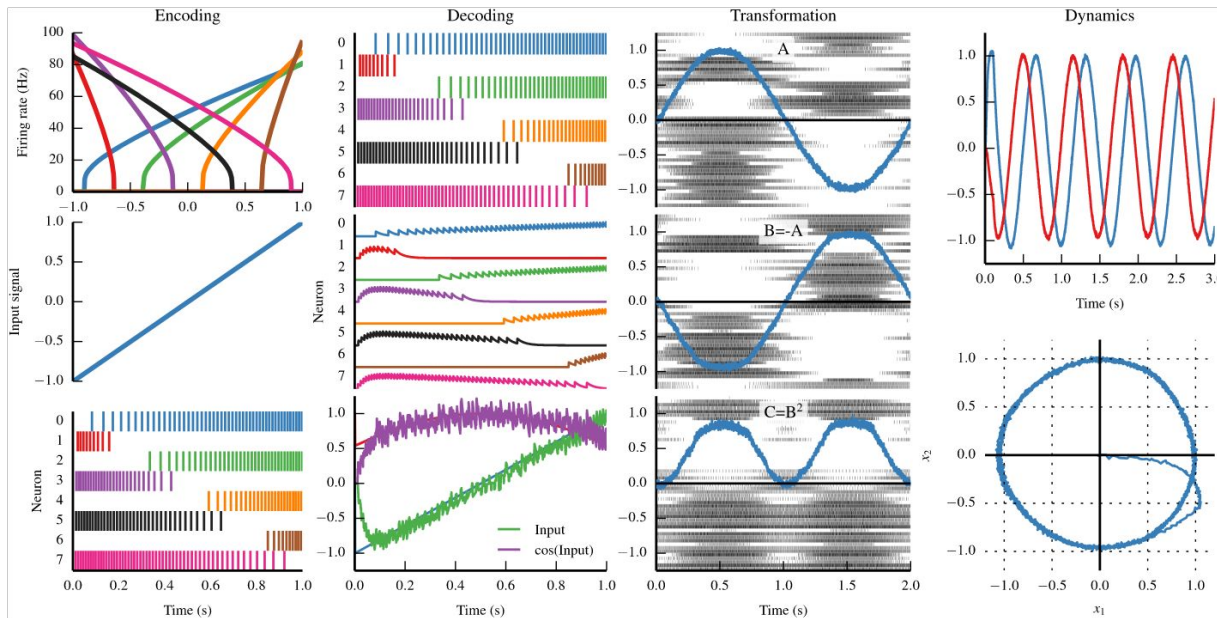
[NEO → Nengo 1.4](#)

# History: The Python years! (2013)

# Nengo



Trevor Bekolay 🙌



James Bergstra

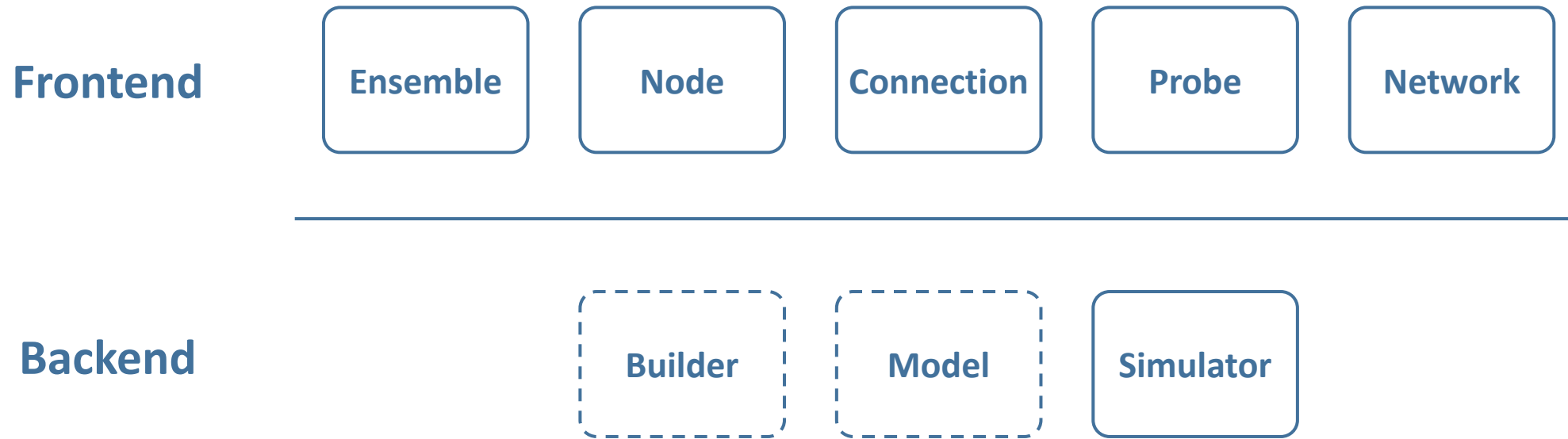


Eric Hunsberger

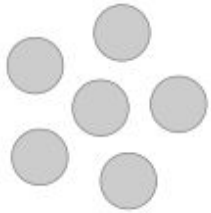
[Nengo 2.0+](#)

[...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

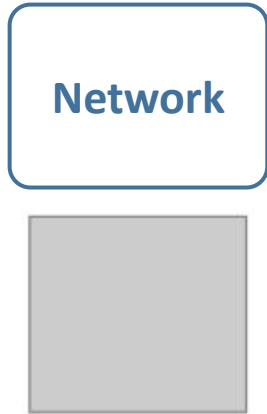
Probe

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

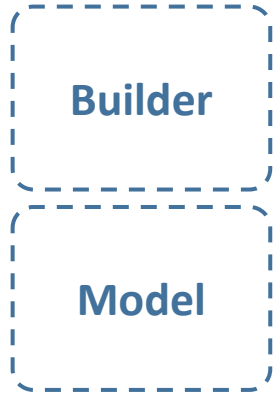
Simulator

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

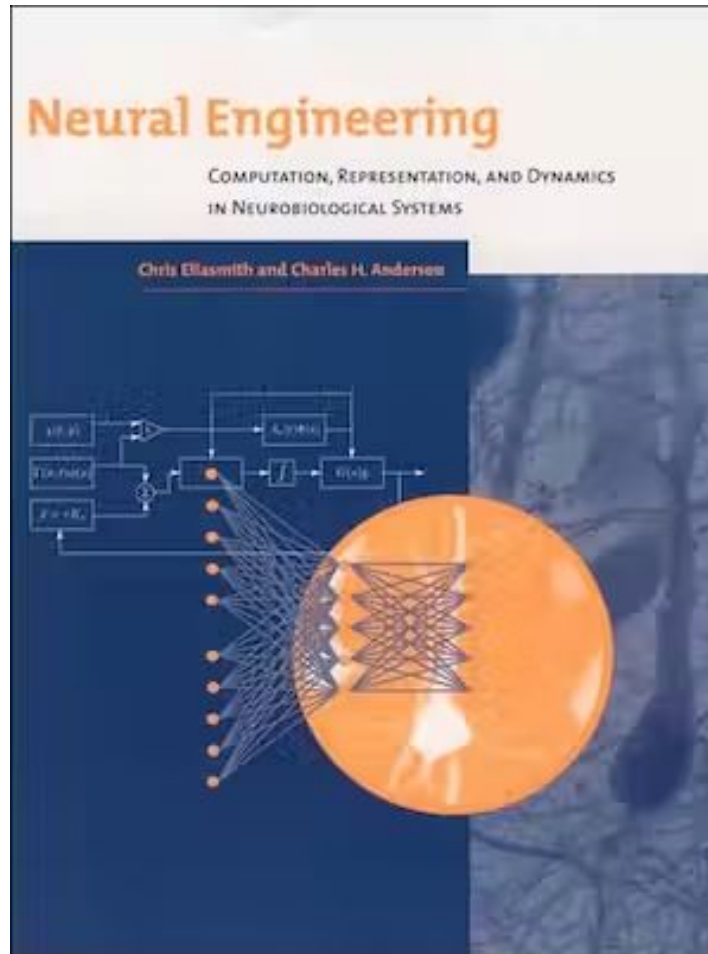
Builder

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

Frontend

Ensemble

Node

Connection

Probe

Network

Backend

Builder

Model

Simulator

# 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](http://nengo.ai/nengo/networks.html)



# NengoSPA

1. Symbols are associated with a high-dimensional vector (pointer)
2. Superposition:  $P1 + P2$
3. Binding:  $P1 \otimes P2 = P3$   
Unbinding:  $P3 \otimes P1^+ = P2 + \text{noise}$

[Spaun \(2013\)](#)

[Spaun \(2021\)](#)

[nengo.ai/nengo-spa](https://nengo.ai/nengo-spa)



Build Nengo models with NumPy syntax

[github.com/nengo-labs/nengo-gyrus](https://github.com/nengo-labs/nengo-gyrus)



## Outer product in Nengo

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

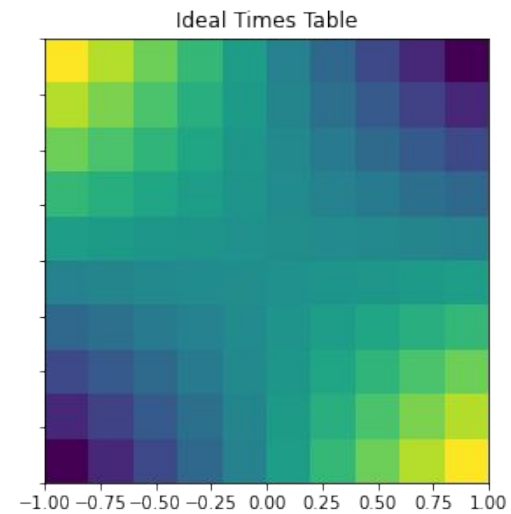
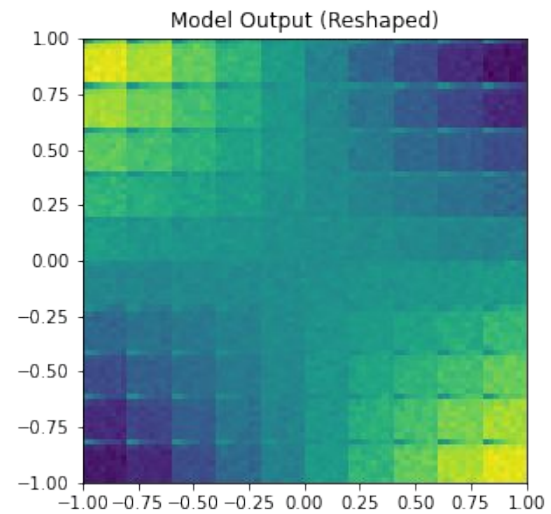
```
1 with nengo.Simulator(model) as sim:
2     sim.run(0.1)
3
4 out = np.asarray(
5     [
6         [sim.data[probes[i, j]].squeeze(axis=-1) for j in range(len(u))]
7         for i in range(len(u))
8     ]
9 )
```





## Outer product in NengoGyrus

```
1 import gyrus
2
3 def times_table(u, tau=0.005):
4     x = gyrus.stimuli(u)
5     return np.outer(x, x).filter(tau)
6
7 out = np.asarray(times_table(u).run(0.1)).squeeze(axis=-1)
```



NengoExtras

nengolib

NengoExamples

# Backend ecosystem

Frontend

Ensemble

Node

Connection

Probe

Network

Backend

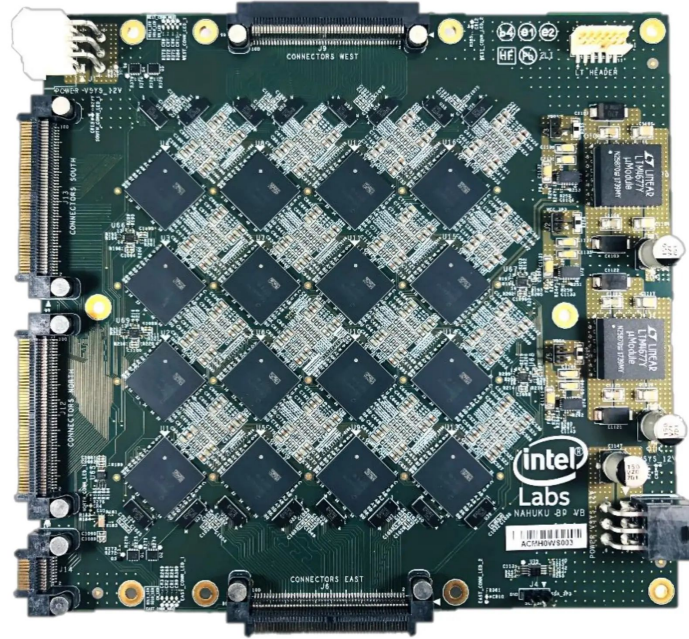
Builder

Model

Simulator



# NengoLoihi



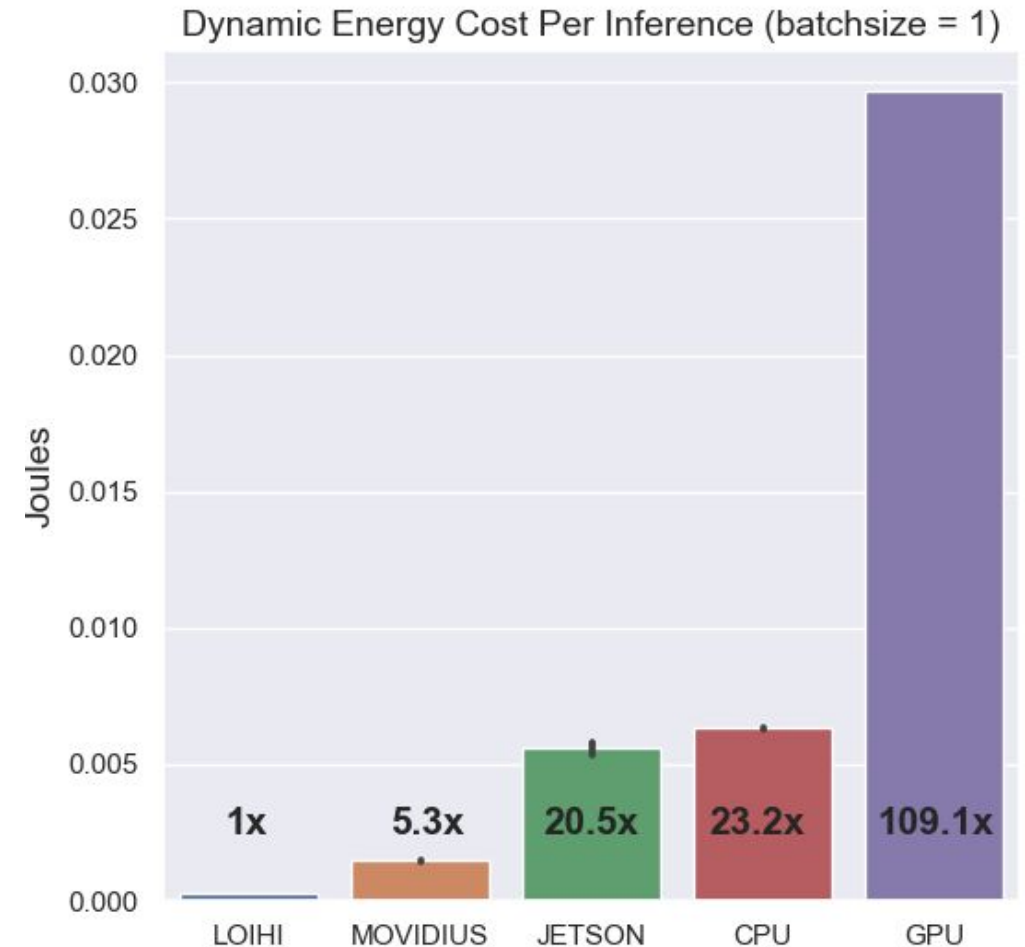
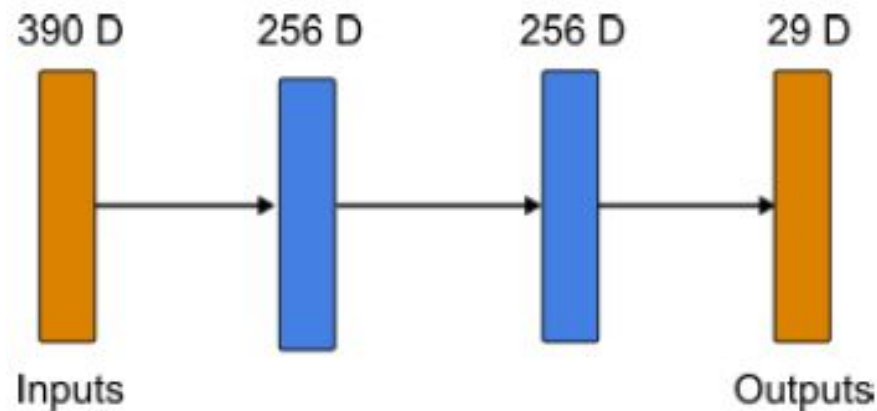
`nengo.Simulator(model) → nengo_loihi.Simulator(model)`

Can target real hardware or our included emulator

[nengo.ai/nengo-loihi](https://nengo.ai/nengo-loihi)

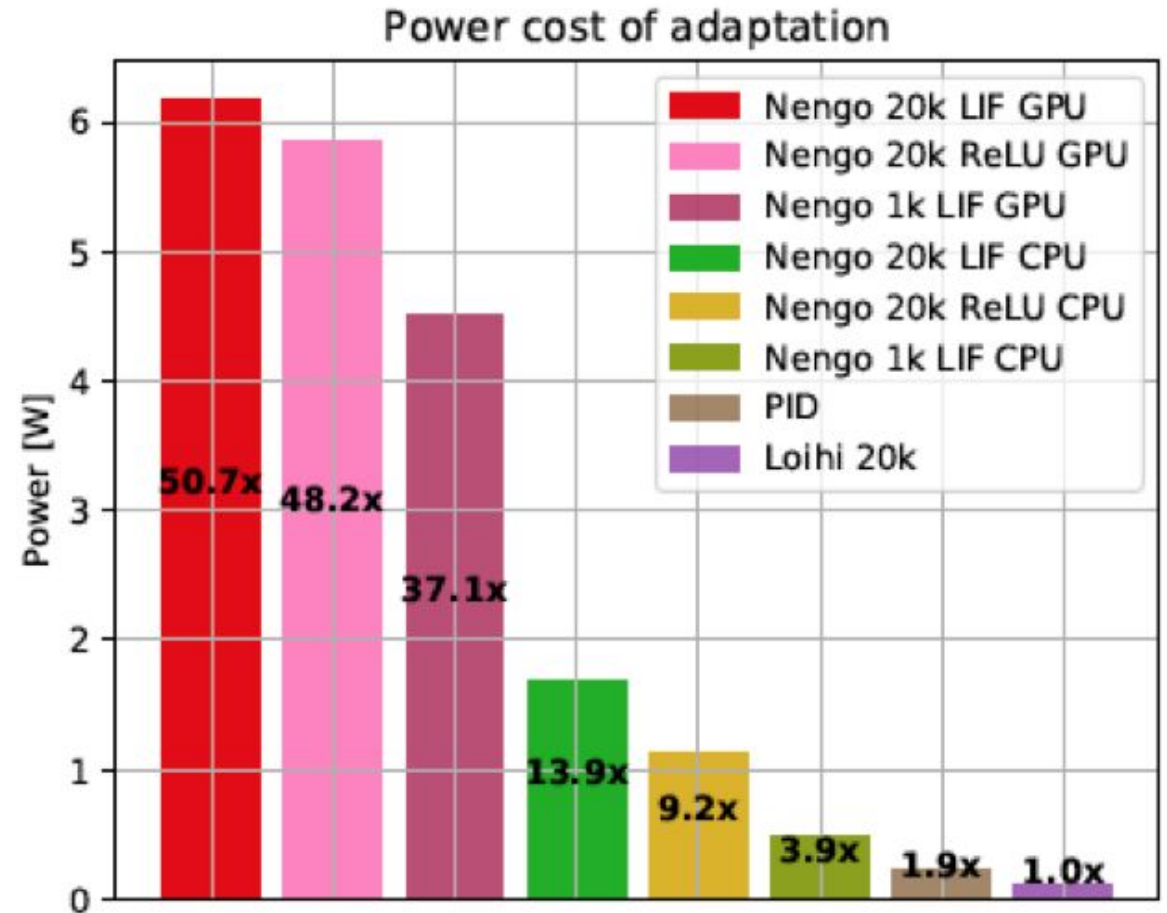
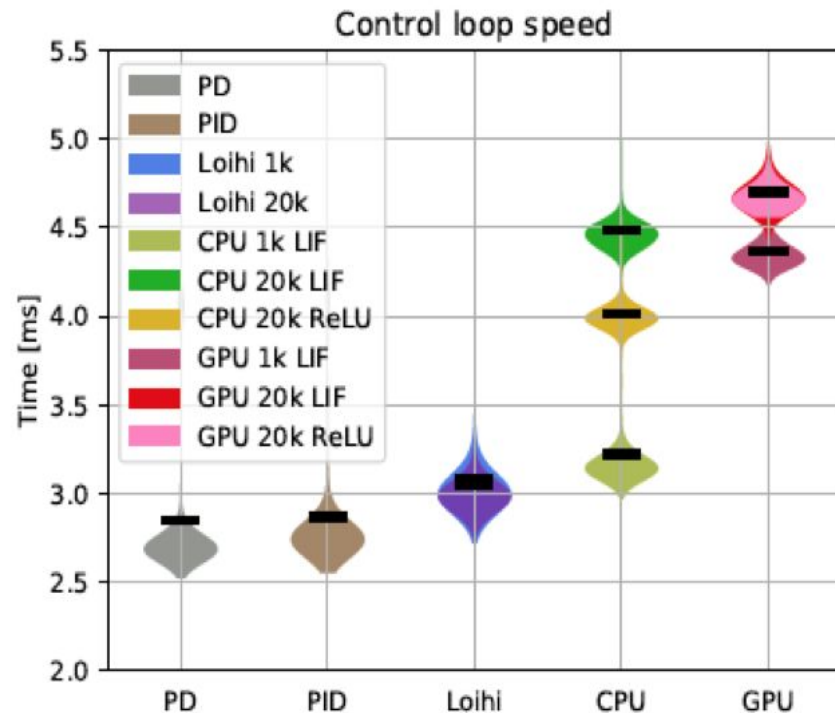
# NengoLoihi

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



# NengoLoihi

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





# Nengo SpiNNaker



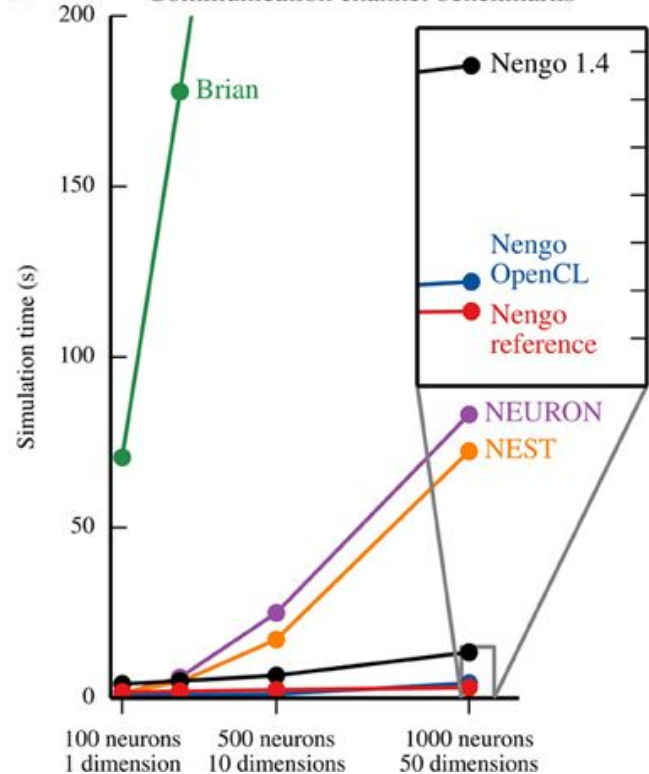
`nengo.Simulator(model) → nengo_spinnaker.Simulator(model)`

[github.com/project-rig/nengo\\_spinnaker](https://github.com/project-rig/nengo_spinnaker)

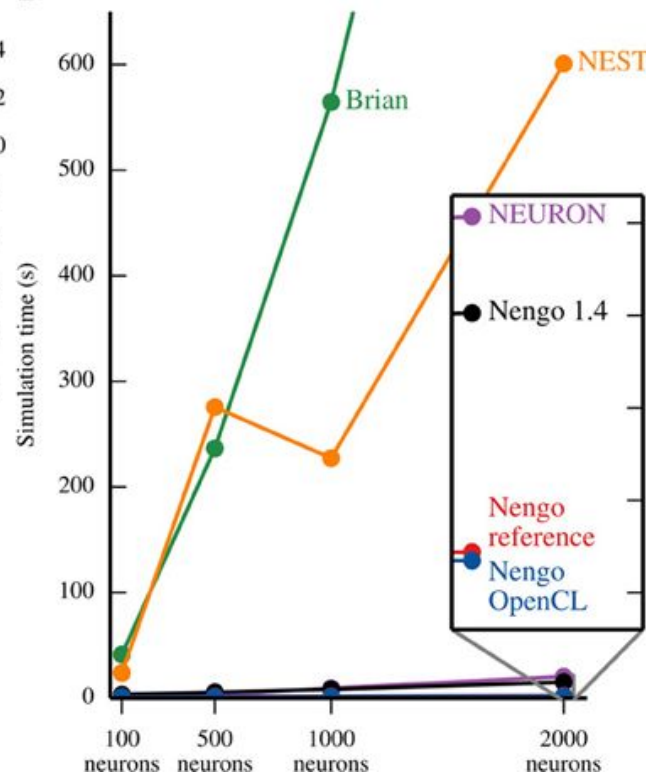


# Nengo OCL

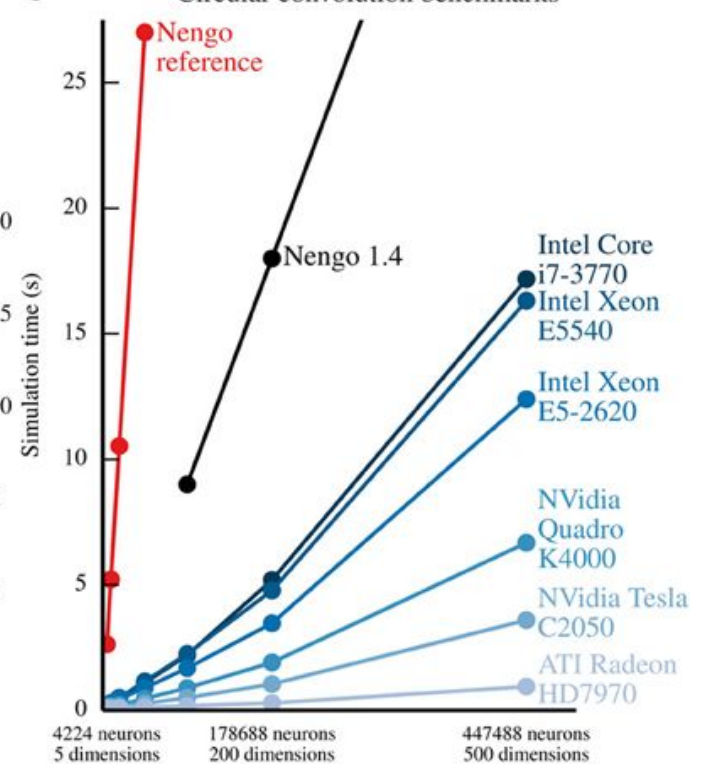
**A** Communication channel benchmarks



**B** Lorenz attractor benchmarks



**C** Circular convolution benchmarks



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

[labs.nengo.ai/nengo-ocl](http://labs.nengo.ai/nengo-ocl)



NengoFPGA

NengoBraindrop

[NengoMPI](#)

# Other parts of the ecosystem

**Frontend**

Ensemble

Node

Connection

Probe

Network

**Backend**

Builder

Model

Simulator



# Nengo GUI

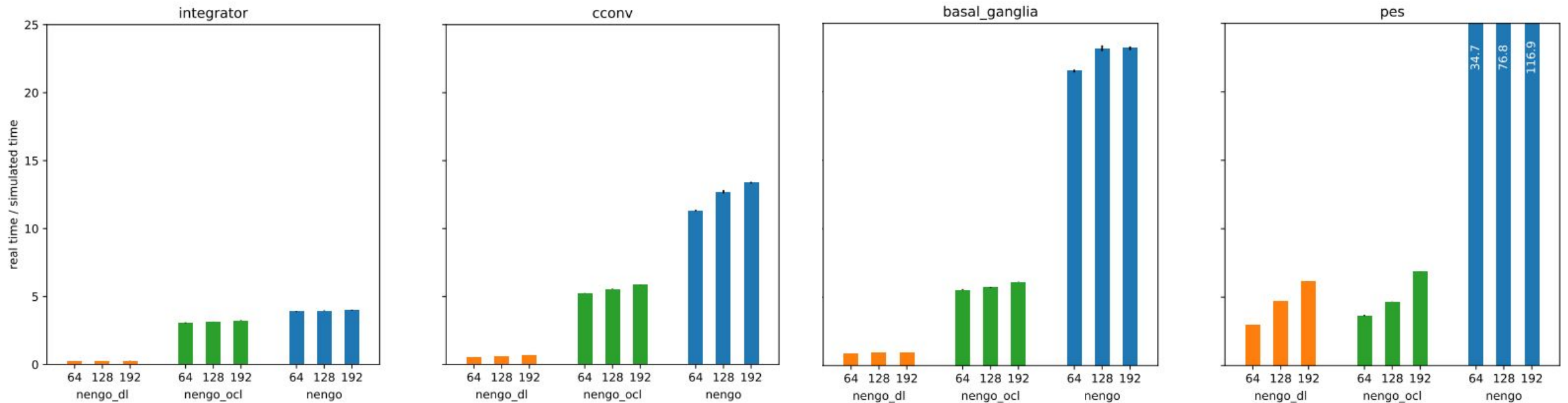
The screenshot displays the Nengo GUI interface. On the left, there are three plots: a top plot showing a spiking attractor with a blue horizontal band, a middle plot showing a network diagram with five gray nodes and a black loop, and a bottom plot showing a Lorenz attractor with a green and orange trajectory. On the right, a code editor shows the Python script for the Lorenz attractor. The script includes comments, imports, and the definition of the Lorenz function. At the bottom, there is a playback control bar with a speed slider set to 0.34x, a time display at 0.222, and a timeline from -3.5 to 0.0.

```
1 # Tutorial 15: The Lorenz Chaotic Attractor
2
3 # Differential equations can also give chaotic behaviour. The classic example
4 # of this is the Lorenz "butterfly" attractor. The equations for it are
5 #
6 # dx0/dt = sigma * (x1 - x0)
7 # dx1/dt = - x0 * x2 - x1
8 # dx2/dt = x0 * x1 - beta * (x2 + rho) - rho
9 #
10 # Note: this is a slight transformation from the standard formulation so
11 # as to centre the value around the origin. For further information, see
12 # http://compneuro.uwaterloo.ca/publications/eliasmith2005b.html
13 # "Chris Eliasmith. A unified approach to building and controlling
14 # spiking attractor networks. Neural computation, 7(6):1276-1314, 2005."
15
16 # Since there are three dimensions, we can show three different XY plots
17 # combining the different values in different ways.
18
19 import nengo
20
21 model = nengo.Network(seed=5)
22 with model:
23
24     x = nengo.Ensemble(n_neurons=600, dimensions=3, radius=30)
25
26     synapse = 0.1
27     def lorenz(x):
28         sigma = 10
29         beta = 8.0/3
30         rho = 28
31
32         dx0 = -sigma * x[0] + sigma * x[1]
33         dx1 = -x[0] * x[2] - x[1]
34         dx2 = x[0] * x[1] - beta * (x[2] + rho) - rho
35
36         return [dx0 * synapse + x[0],
37               dx1 * synapse + x[1],
38               dx2 * synapse + x[2]]
39
40     nengo.Connection(x, x, synapse=synapse, function=lorenz)
41
```

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



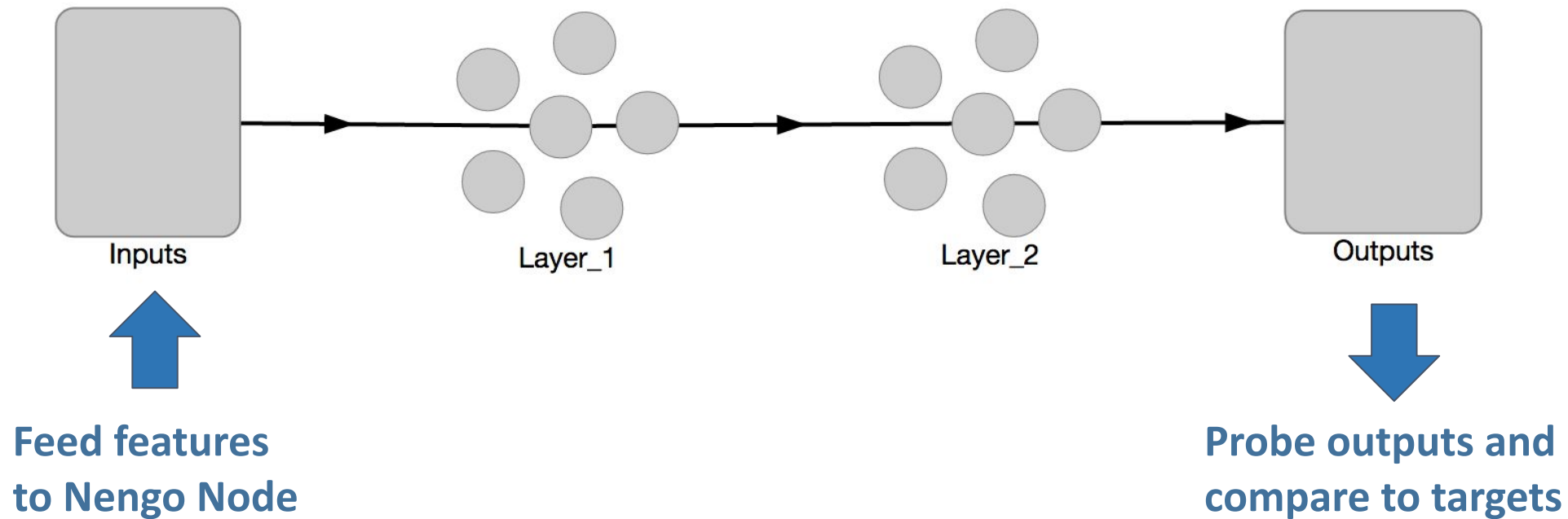
# NengoDL



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

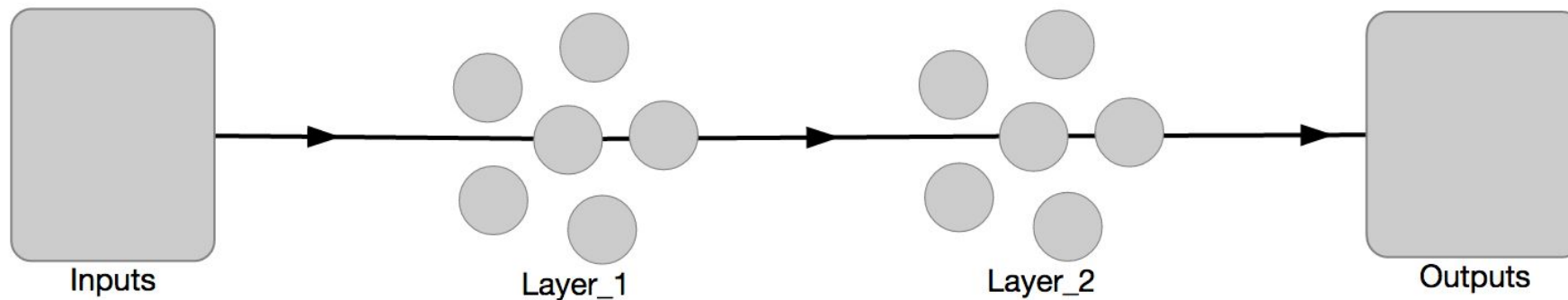
[nengo.ai/nengo-dl](https://nengo.ai/nengo-dl)

# NengoDL



**Nengo DL will optimize all intermediate network parameters!**

# NengoDL



```
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](#)

all categories ▾

all ▾

Latest

Top

Categories

+ New Topic

☰ Topic

Replies

Views

Activity

Installation Mac M1

■ Getting Started



3

43

7h

PES high learning rate VS amplifying post

■ General Discussion



0

4

7h

last visit

Additional variables in Signals for custom learning rules

■ General Discussion



0

99

1d

Different simulation results in neuron Direct and LIF mode

■ General Discussion



8

39

6d

The effects of neural gain

■ General Discussion



0

35

17d

2015, Diehl and cook model implementation

■ Examples & Tutorials



0

39

18d

[forum.nengo.ai](https://forum.nengo.ai)



A glowing blue brain is centered on a black background. The brain is rendered with a semi-transparent, glowing effect, showing its complex, folded surface. Overlaid on the brain is the text "Nengo Summer School" in a white, bold, sans-serif font.

# Nengo Summer School

June 4th - June 16th, 2023 at UWaterloo  
Applications open! [nengo.ai/summer-school](https://nengo.ai/summer-school)

# Licensing

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

Thanks! Questions?