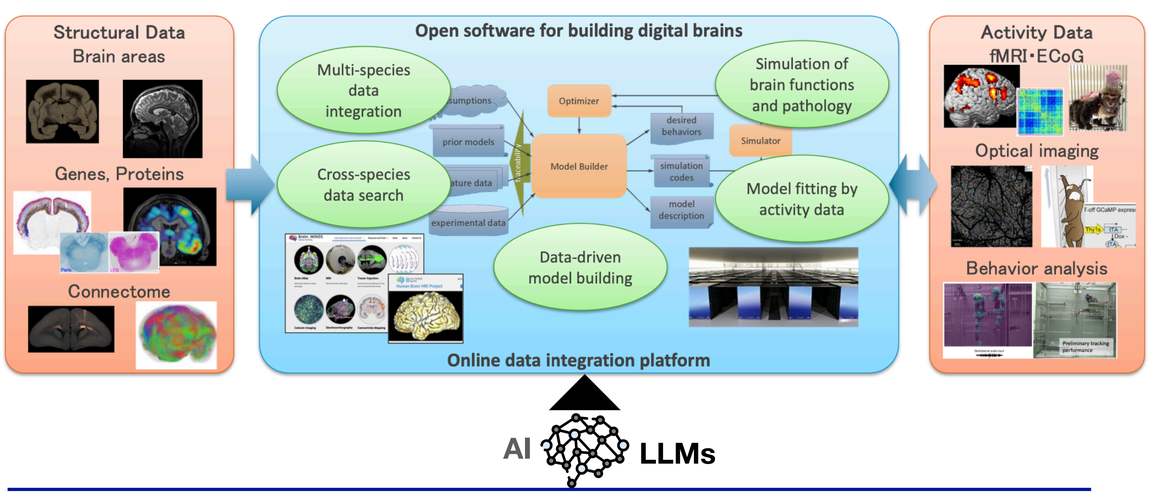
NeuroWorkflow: A python-based Graph Framework for Modular Brain Modeling Workflows

Carlos Enrique Gutierrez (1), Henrik Skibbe (2), Yukako Yamane (1), Kenji Doya (1)
(1) Neural Computation Unit, Okinawa Institute of Science of Technology Graduate University, Onna-son, Japan. (2) Brain Image Analysis Unit. RIKEN CBS. Wako, Japan

Brain/MINDS 2.0 - The Digital Brain Project

Goals: open software for building digital brains
-online platform for data integration, model building and simulations-

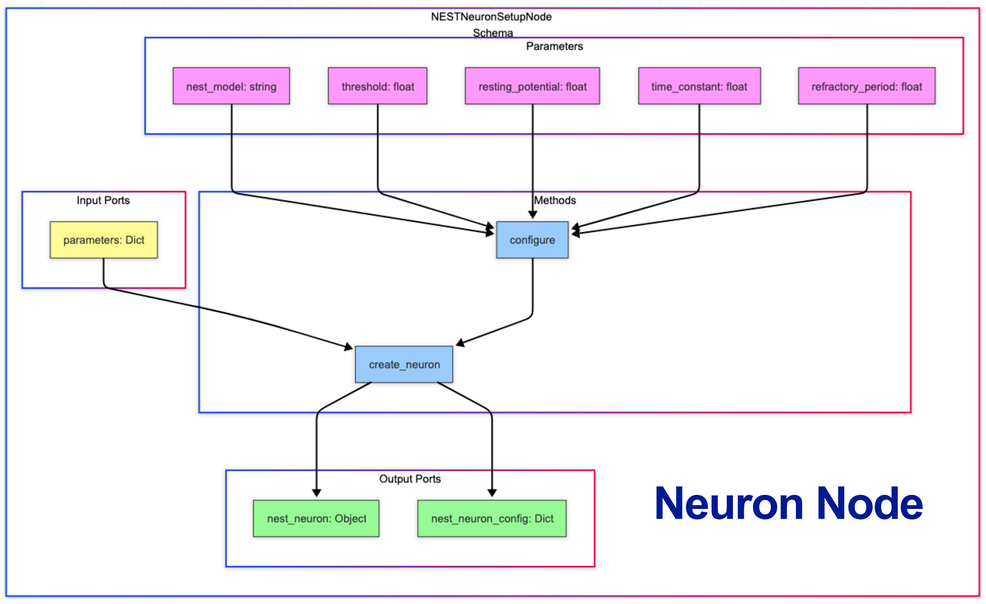


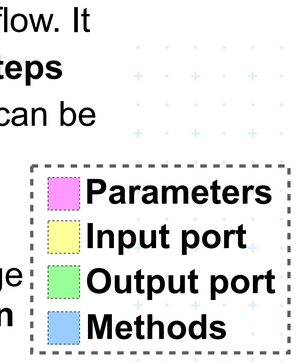
The Digital Brain as a graph-based framework

- The Brain/MINDS 2.0 Digital Brain aims to create a framework that supports **interoperability with open data and popular neuroscience tools** such as TVB, NEST, NEURON, and others, leveraging their Python interfaces.
- Modern brain modeling and simulation workflows often require complex sequences of data processing, model configuration, and analysis steps. While **powerful tools** exist, they frequently **demand advanced programming knowledge, limiting accessibility and collaboration potential**. Besides that, **shared models are frequently difficult to understand, execute and expand**.
- We present a python-based graph framework that **transforms complex scientific workflows into modular, interpretable, reusable components through a node-edge** architecture.

NeuroWorkflow Node's Core Components

- The node is the fundamental building block of NeuroWorkflow. It defines **inputs, outputs, parameters, and processing steps** (functions). It handles data flow and type validation and it can be **extended to create specialized/custom nodes**.
- The diagram below illustrates the structure of a NEST **Neuron Setup Node** class in the **NeuroWorkflow** package which **defines the node as a python class with a certain schema**. Here's a breakdown of the components:





Exploring workflows for model's optimization

- This diagram illustrates a **preliminary workflow for the optimization of a single neuron parameters**, showing how different **nodes interact via ports (edges)**.

In Python a Node is defined as a Class:

NESTNeuronSetupNode Schema

Node Structure Table

Component	Details
Node Type	`nest_neuron_setup`
Description	Sets up NEST neurons for neural simulations

Parameters

Parameter Name	Type	Description	Default Value
`nest_model`	string	NEST neuron model type	'iaf_psc_alpha'
`threshold`	float	Firing threshold (mV)	-55.0
`resting_potential`	float	Resting membrane potential (mV)	-70.0
`time_constant`	float	Membrane time constant (ms)	20.0
`refractory_period`	float	Refractory period after spiking (ms)	2.0

Input Ports

Port Name	Type	Description	Optional
`parameters`	Dict	Parameter updates from optimization	Yes

Output Ports

Port Name	Type	Description
`nest_neuron`	Object	NEST neuron object
`nest_neuron_config`	Dict	Configuration of the NEST neuron

Methods

Method Name	Description	Inputs	Outputs
`configure`	Sets up the node with initial parameters	nest_model, threshold, resting_potential, time_constant, refractory_period	None
`create_neuron`	Creates a NEST neuron with the specified parameters	parameters (from input port)	nest_neuron, nest_neuron_config

Process Steps

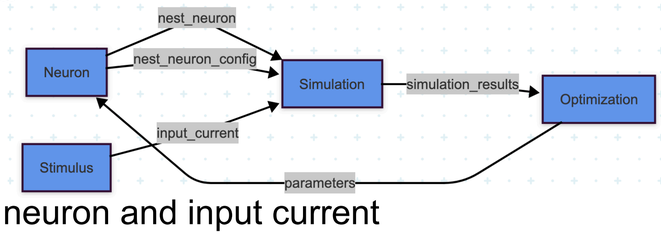
1. Receive parameters (if any) from optimization node
2. Execute `create_neuron` method
3. Output the NEST neuron object and configuration

```python
class NESTNeuronSetupNode(Node):
    """Node for creating and configuring neuron models.
    This class represents a neuron model with configurable parameters.
    In a real implementation, this could be a more complex model or
    interface with external neural simulators."""

    NODE_DEFINITION = NodeDefinitionSchema(
        type='neuron_setup',
        description='Creates and configures a neuron model in NEST',
        parameters={
            'nest_model': ParameterDefinition(
                default_value='iaf_psc_alpha',
                description='neuron model name in NEST'
            ),
            'threshold': ParameterDefinition(
                default_value=-55.0,
                description='Firing threshold (mV)',
                constraints={'min': -70.0, 'max': -40.0},
                optimizable=True,
                optimization_range=[-65.0, -45.0]
            ),
            'resting_potential': ParameterDefinition(
                default_value=-70.0,
                description='Resting membrane potential (mV)',
                constraints={'min': -80.0, 'max': -60.0},
            ),
            'time_constant': ParameterDefinition(
                default_value=20.0,
                description='Membrane time constant (ms)',
                constraints={'min': 5.0, 'max': 50.0},
                optimizable=True,
                optimization_range=[10.0, 30.0]
            ),
            'refractory_period': ParameterDefinition(
                default_value=2.0,
                description='Refractory period (ms)',
                constraints={'min': 1.0, 'max': 5.0}
            )
        },
        inputs={
            'parameters': PortDefinition(
                type=PortType.DICT,
                description='Parameters to configure the neuron',
```

```python
def create_neuron(self, parameters: Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
    """Create and configure a neuron model.

    Args:
        parameters: Optional parameters to override defaults

    Returns:
        Dictionary with neuron model and configuration
    """
    # If parameters are provided, configure the node
    if parameters:
        self.configure(**parameters)

    # Create nest neuron object
    neuro_params = {
        "V_th": self._parameters['threshold'],
        "E_L": self._parameters['resting_potential'],
        "tau_m": self._parameters['time_constant'],
        "t_ref": self._parameters['refractory_period'],
    }

    nest.set_verbosity("M_ERROR")
    nest.ResetKernel()
    neuron = nest.Create(self._parameters['nest_model'], params=neuro_params)
```

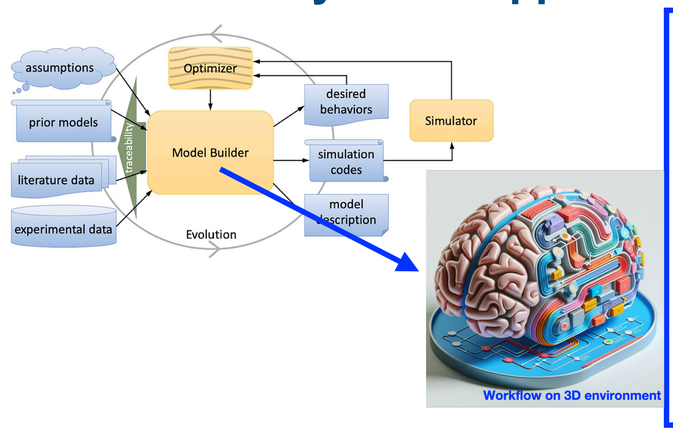
Workflow Process

- Initialization:
  - Nodes are configured with initial parameters
- Execution Loop:
  - Neuron Node creates a neuron model
  - Stimulus Node generates an input current
  - Simulation Node runs the simulation with the neuron and input current
  - Optimization Node evaluates the results and suggests new parameters
  - The new parameters are fed back to Neuron Node
  - The loop continues until optimization criteria are met



This simple workflow **demonstrates the power of the node-based architecture**, where each node handles a specific aspect of the neural simulation process, and data flows between nodes through well-defined ports.

Community-based approach for building digital brains



- The model builder aims to be **composed by several custom nodes built by BM2** researchers and collaborators.
- Several **easy-to-understand complex brain models** and workflows may be built and **deployed easily**.
- Data integration and data analysis workflows can be deployed as well.
- Machine learning / AI dedicated nodes can also be created and integrated.

Acknowledgments: This work was supported by Brain/MINDS 2.0 project (AMED), Development of the "digital brain" and related research platforms utilizing mathematical models.