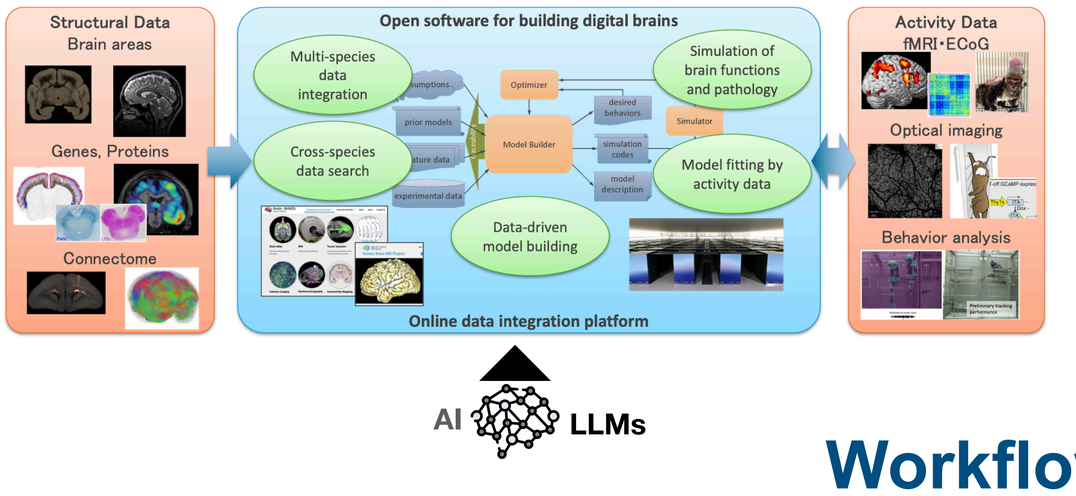
A Graph-Based, In-Memory Workflow Library for Brain/MINDS 2.0 – The Japan Digital Brain Project

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-



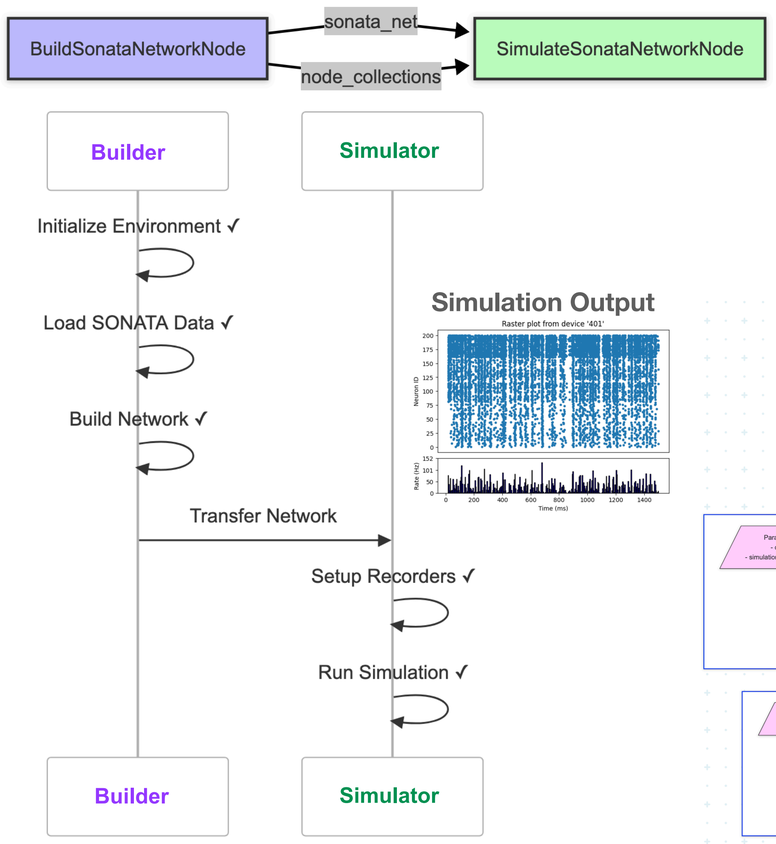
The Digital Brain as a graph-based framework

- 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.
- We present a python-based graph framework that **transforms complex scientific workflows into modular, interpretable, reusable components through a node-edge** architecture.
- The node is the fundamental building block of our library. It defines **inputs, outputs, parameters, and processing steps** (methods).
  A node is defined as a **python class** that follows a schema.

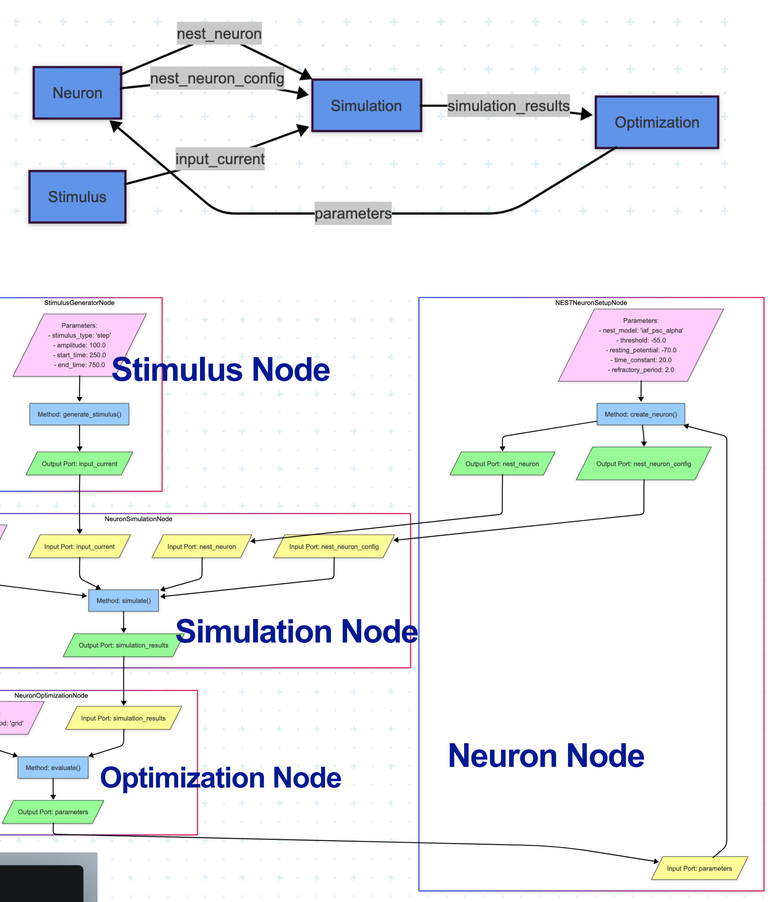
Parameters
Input port
Output port
Methods

Workflow examples

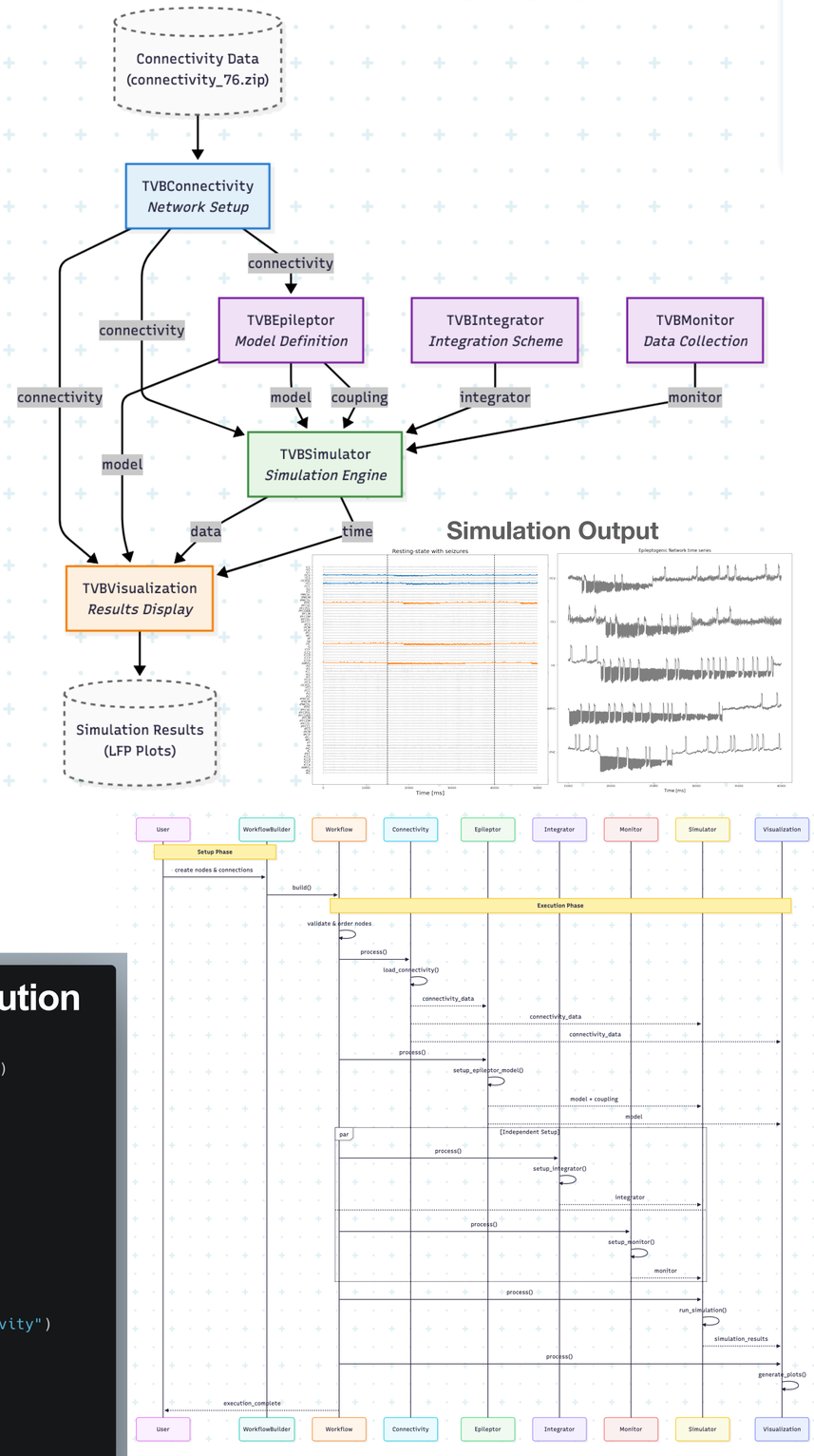
1.**SONATA-NEST** workflow implementation



2.**NEST neuron** optimization workflow



3.**TVB Resting-State in Epilepsy** workflow



Node definition

```python
from neuroworkflow.core.node import Node
from neuroworkflow.core.schema import NodeDefinitionSchema, PortDefinition, ParameterDefinition,
MethodDefinition
from neuroworkflow.core.port import PortType

class MyNode(Node):
    """Brief description of node functionality."""

    NODE_DEFINITION = NodeDefinitionSchema(
        type='node_category',
        description='What this node does',

        parameters={
            'param1': ParameterDefinition(
                default_value=1.0,
                description='Parameter description',
                optimizable=True,
                optimization_range=[0.1, 10.0]
            ),
        },

        inputs={
            'input_data': PortDefinition(
                type=PortType.OBJECT,
                description='Input description'
            ),
        },

        outputs={
            'output_data': PortDefinition(
                type=PortType.OBJECT,
                description='Output description'
            ),
        },

        methods={
            'process_method': MethodDefinition(
                description='Main processing method',
                inputs=['input_data'],
                outputs=['output_data']
            ),
        }
    )

    def __init__(self, name: str):
        super().__init__(name)
        self._define_process_steps()

    def _define_process_steps(self):
        self.add_process_step(
            "process_method",
            self.process_method,
            method_key="process_method"
        )

    def process_method(self, input_data):
        """Main processing logic."""
        # Access parameters: self._parameters['param1']
        # Your processing logic here
        result = input_data  # Replace with actual processing

        return {'output_data': result}
```

Parameters
Input port
Output port
Methods
Methods

Workflow building & execution

```python
# 1. Create & Configure Nodes
load_sc =
TVBConnectivitySetUpNode("TVBConnectivity").configure(connectivity_file='connectivity_76.zip')
model = TVBEpileptorNode("TVBEpileptor").configure()
integrator = TVBIntegratorNode("TVBIntegrator").configure()
monitor = TVBMonitorNode("TVBMonitor").configure()
simulation = TVBSimulatorNode("TVBSimulator").configure()
plots = TVBVisualizationNode("TVBVisualization").configure()

# 2. Build Workflow
workflow_builder = WorkflowBuilder("Resting-State in Epilepsy")
workflow_builder.add_node(load_sc).add_node(model).add_node(integrator)
                .add_node(monitor).add_node(simulation).add_node(plots)

# 3. Connect Nodes (Data Flow)
workflow_builder.connect("TVBConnectivity", "tvb_connectivity", "TVBEpileptor", "tvb_connectivity")
                .connect("TVBEpileptor", "tvb_model", "TVBSimulator", "tvb_model")
                .connect("TVBIntegrator", "tvb_integrator", "TVBSimulator", "tvb_integrator")
                .connect("TVBSimulator", "tvb_simdata", "TVBVisualization", "data_series")

# 4. Execute
workflow = workflow_builder.build()
success = workflow.execute()
```


Community-based approach for building digital brains

- The model builder aims to be **composed by several custom nodes built by collaboration**.
- Several **easy-to-understand complex brain models** and workflows may be built and **deployed easily**.

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