WHEN TRUST MATTERS

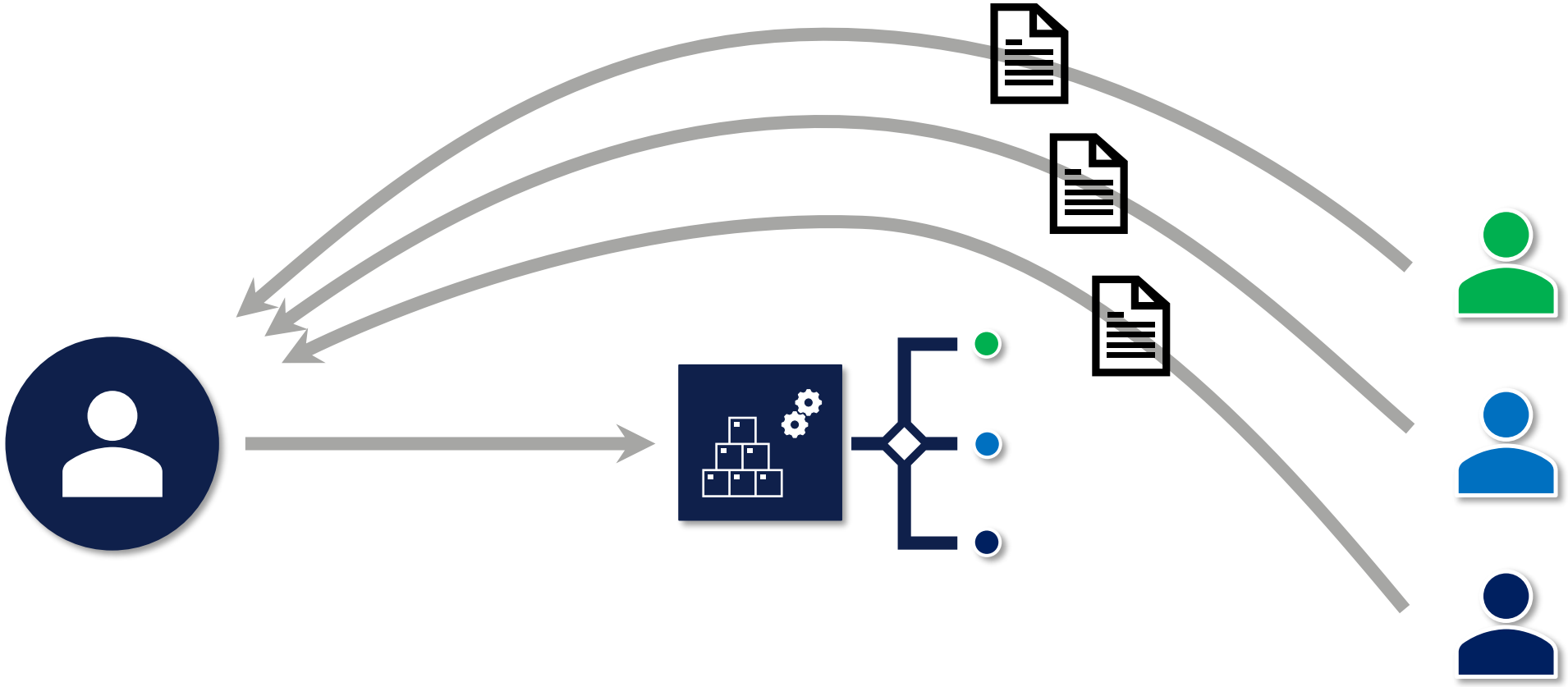
MLFMU: An easy-to-use tool for converting ML models to FMUs

Kristoffer Skare, Stephanie Kemna
Jorge Mendez, Melih Akdağ, Hee Jong Park, Claas Rostock

Open Simulation Platform conference

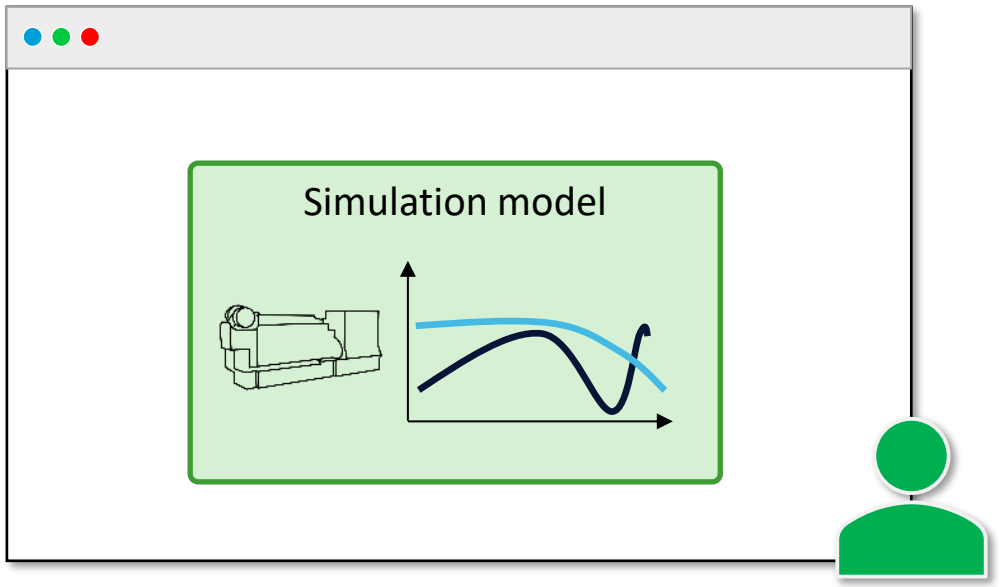
Tuesday, November 12, 2024

System integration: collaboration beyond data sheets

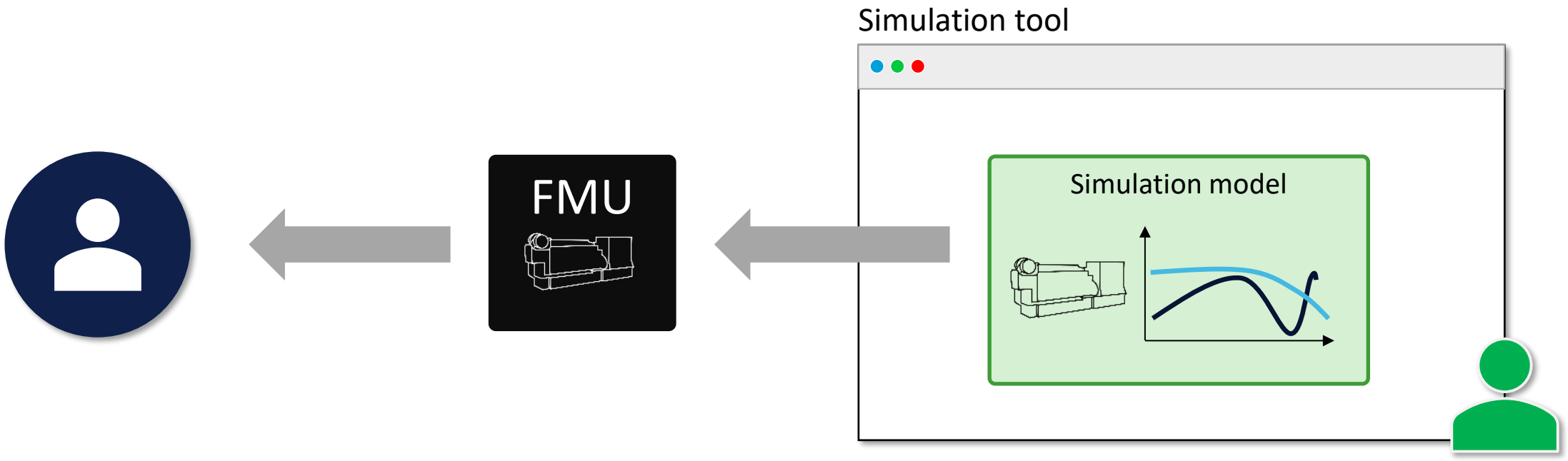


fmi: Functional Mockup Interface

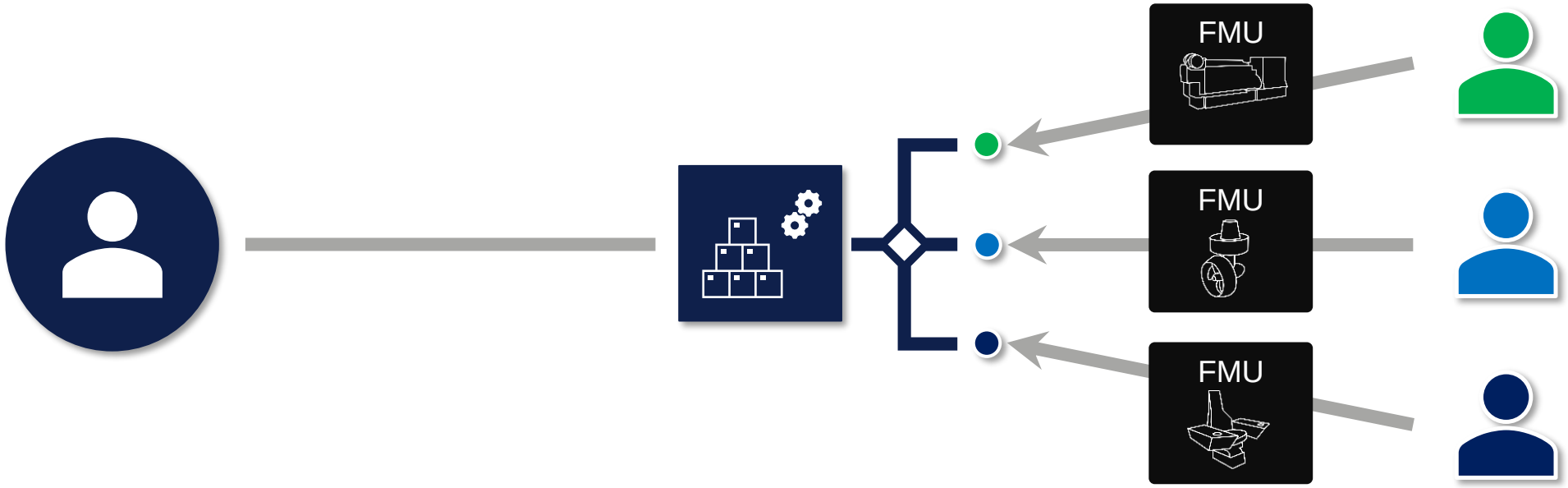
Functional
Mock-Up
Interface



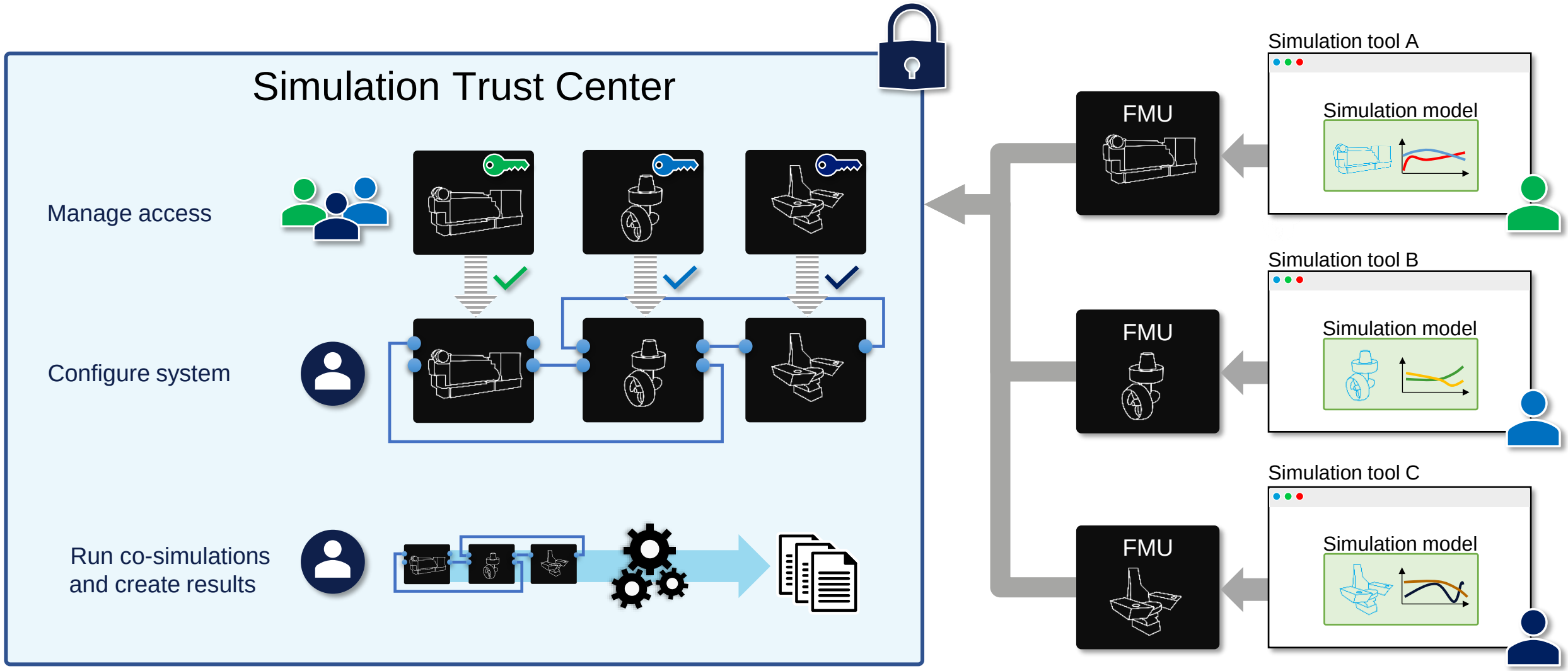
FMU: Functional Mockup Unit



System Integration



STC: Secure Collaboration Space



https://store.veracity.com/simulation-trust-center

Why Machine Learning (ML) models?

Advent of AI: It is common now to create ML models from data

Don’t have an accurate simulation model (yet) but need a model ASAP

Need a model that can run faster than the physics-based high-fidelity model

How to create FMUs from ML models?

Premise:
We want to be able to use powerful tools, e.g. PyTorch or TensorFlow, to build ML models.

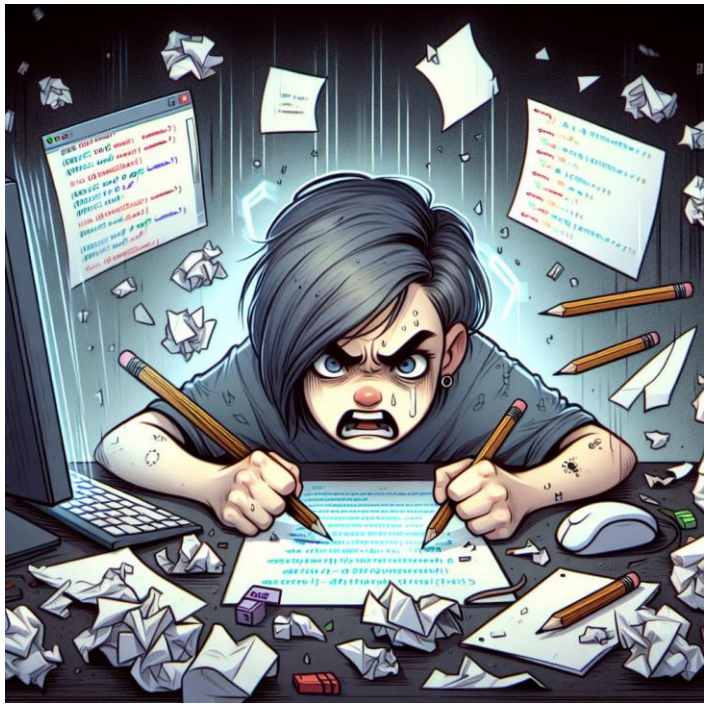
Current options:

a. Use PythonFMU and download/include (for example) PyTorch in FMU

b. Export Tensorflow lite to C code, then wrap C code as FMU

c. Export ML model for import to Matlab/Simulink, then export from there to FMU

None of these are great options..



ONNX: Open Neural Network Exchange

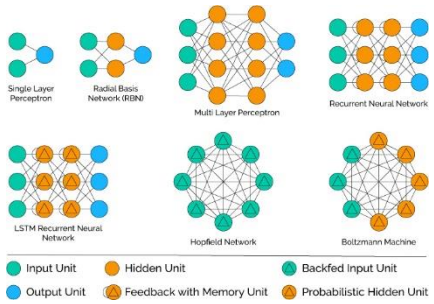
- ONNX is an open format built to represent ML models.
- ONNX defines a common set of operators - the building blocks of ML and DL models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.
- ONNX Runtime: library to optimize and accelerate ML inferencing
  - Take model, export ML model to ONNX format, do inferencing in many different languages.

Commonly used in ML community.

https://onnx.ai/ / https://onnxruntime.ai/

From data to FMU:

Create your ML model → Save as ONNX file → Prepare FMU interface specification file (.json) → Compile FMU from ONNX and json files → Run FMU



ONNX

{JSON}

FMU

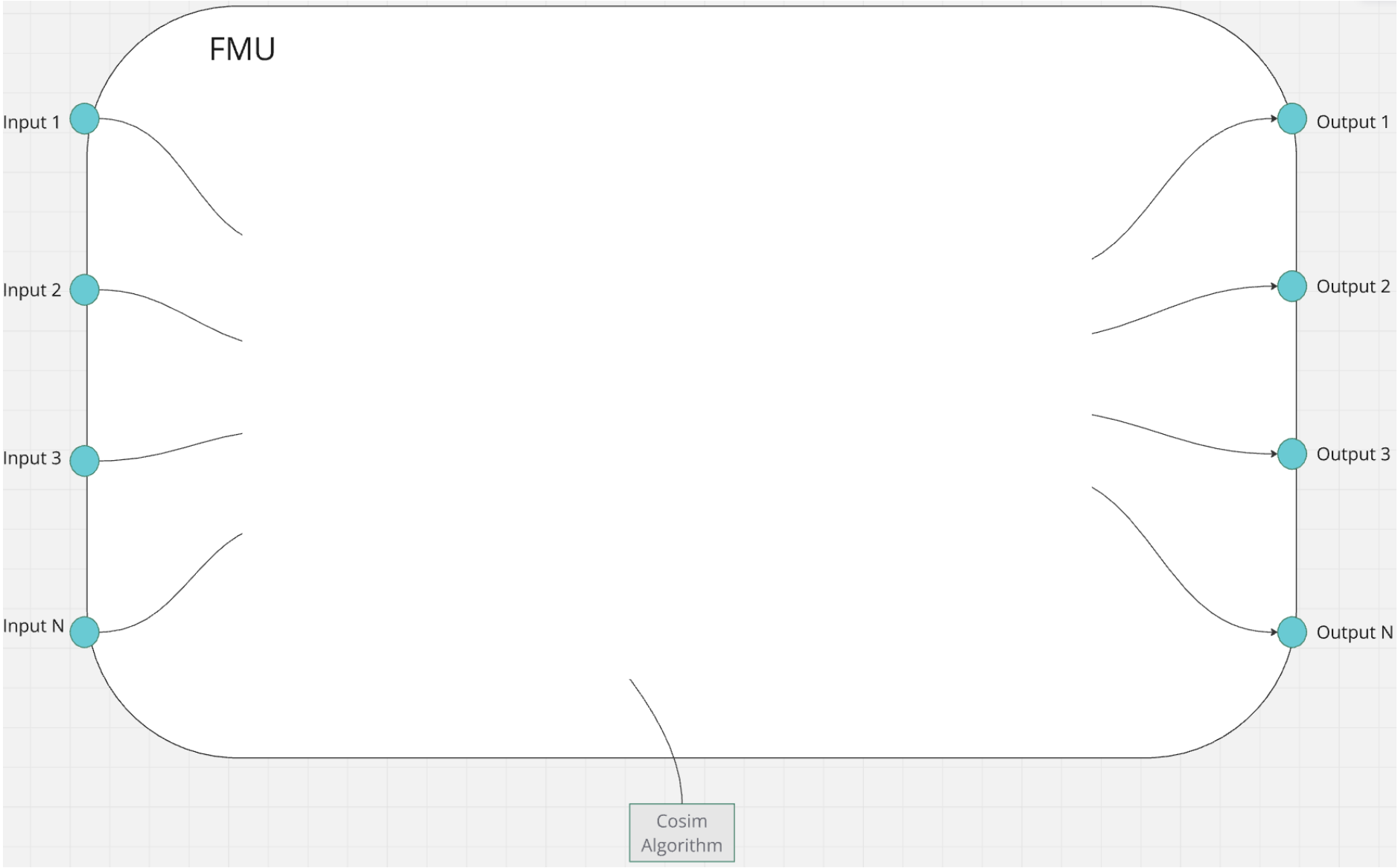
fmi

DNV

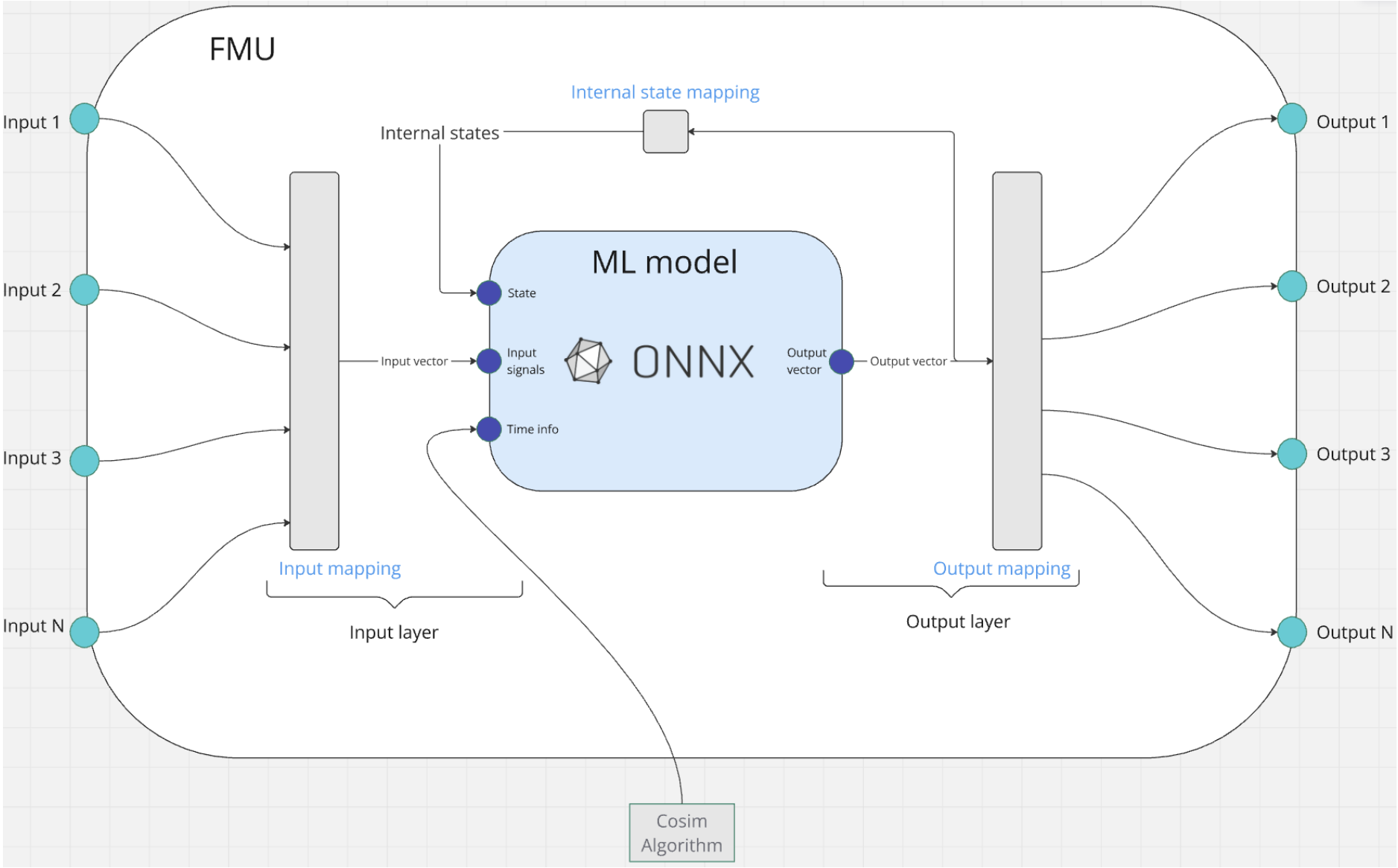
MLFMU:
From ONNX to FMU

DNV

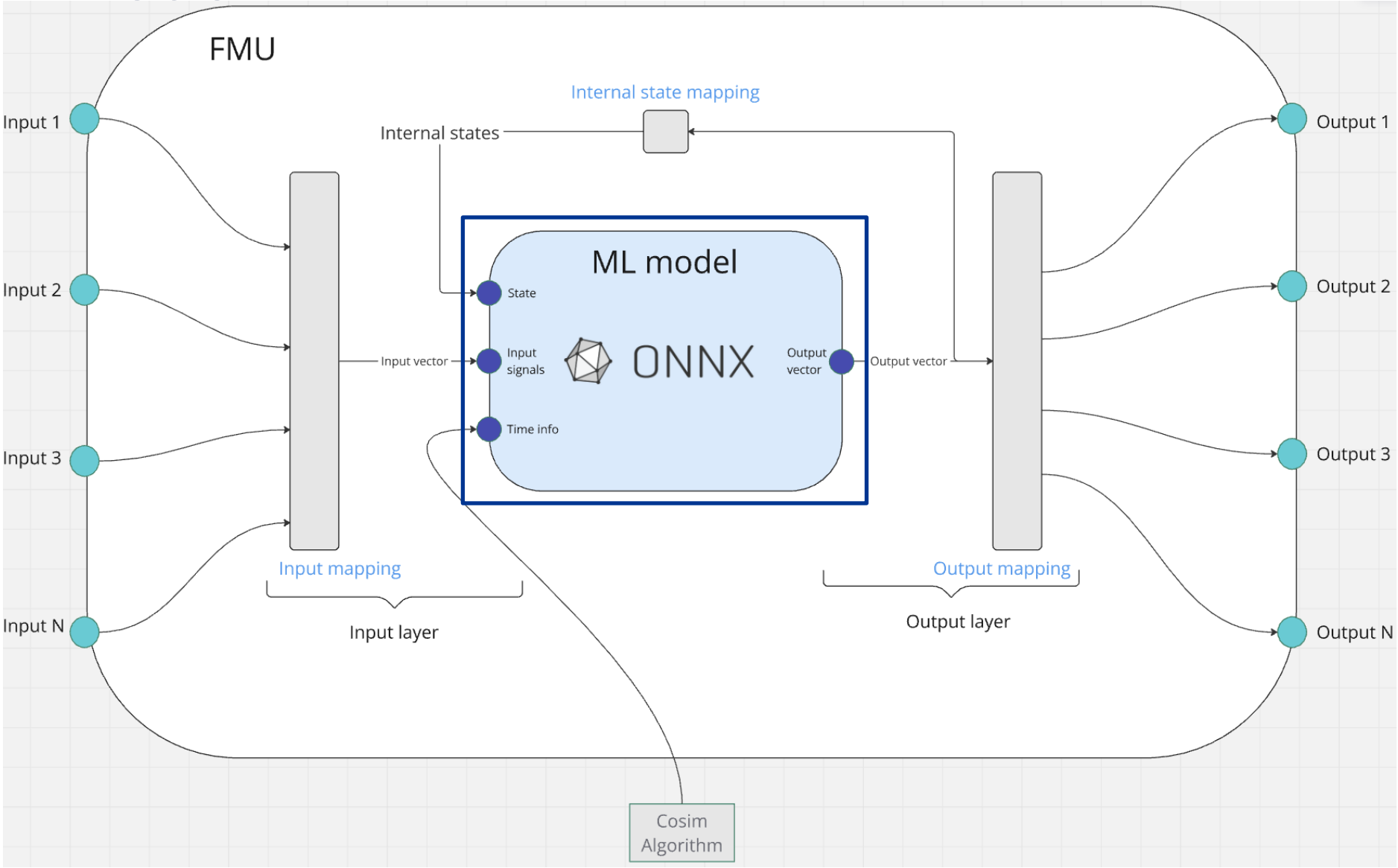
FMU


FMU
Input 1
Input 2
Input 3
Input N
Output 1
Output 2
Output 3
Output N
Cosim
Algorithm

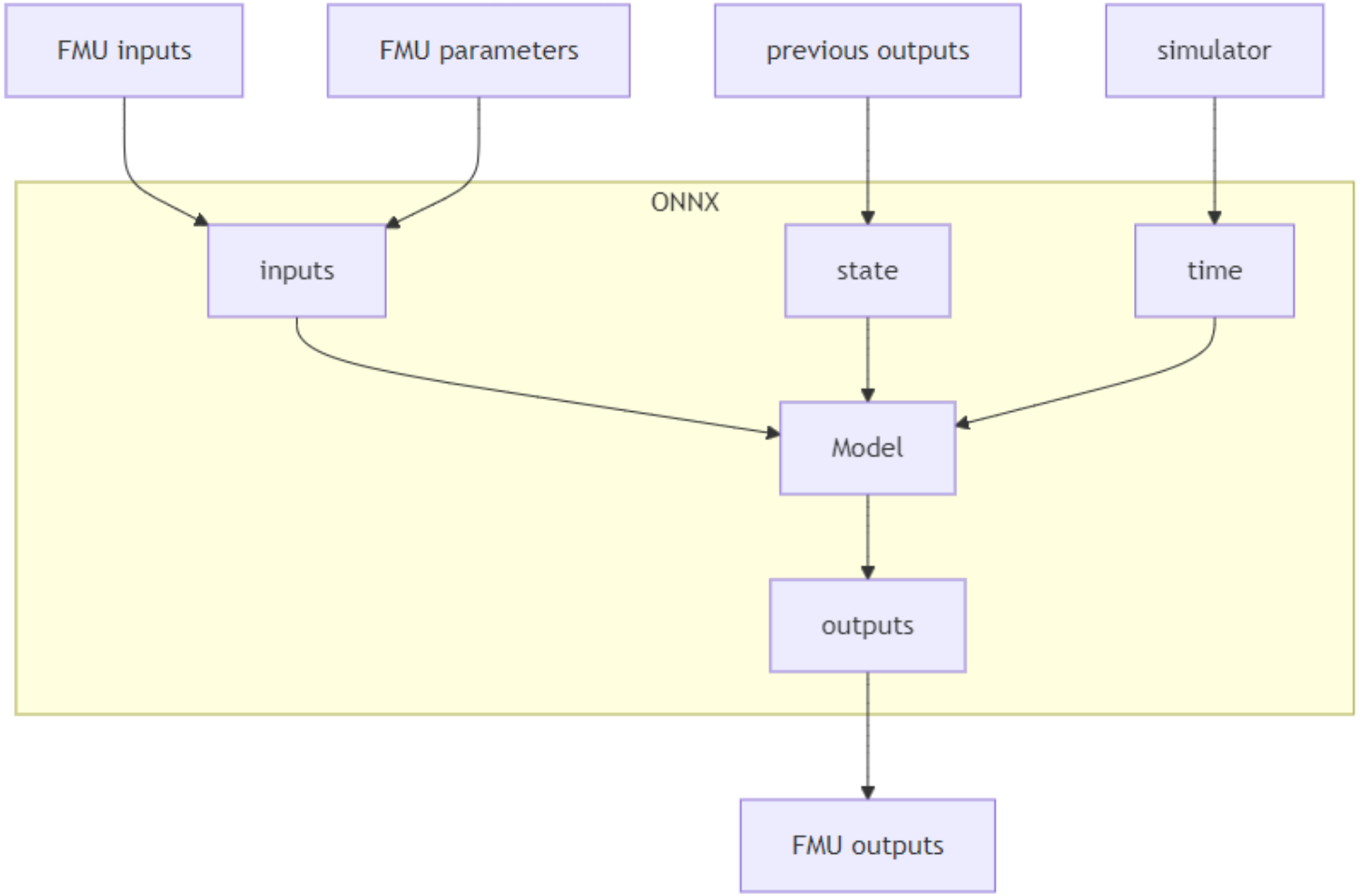
FMU


FMU
Internal state mapping
Internal states
Input 1
Input 2
Input 3
Input N
ML model
State
Input signals
Time info
ONNX
Output vector
Input vector
Output vector
Output 1
Output 2
Output 3
Output N
Input mapping
Input layer
Output mapping
Output layer
Cosim Algorithm

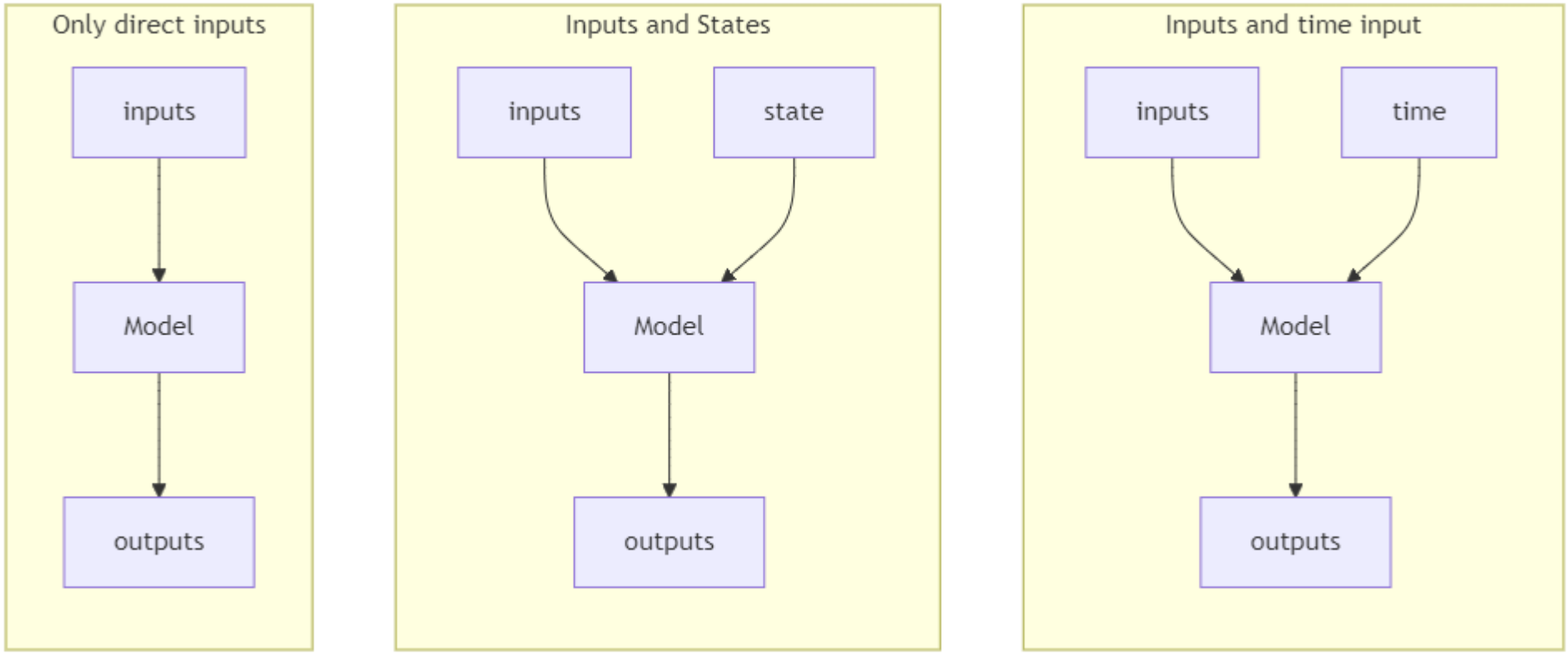
ONNX model


FMU
Internal state mapping
Internal states
Input 1
Input 2
Input 3
Input N
ML model
State
Input signals
ONNX
Output vector
Time info
Input vector
Output vector
Output 1
Output 2
Output 3
Output N
Input mapping
Input layer
Output mapping
Output layer
Cosim Algorithm

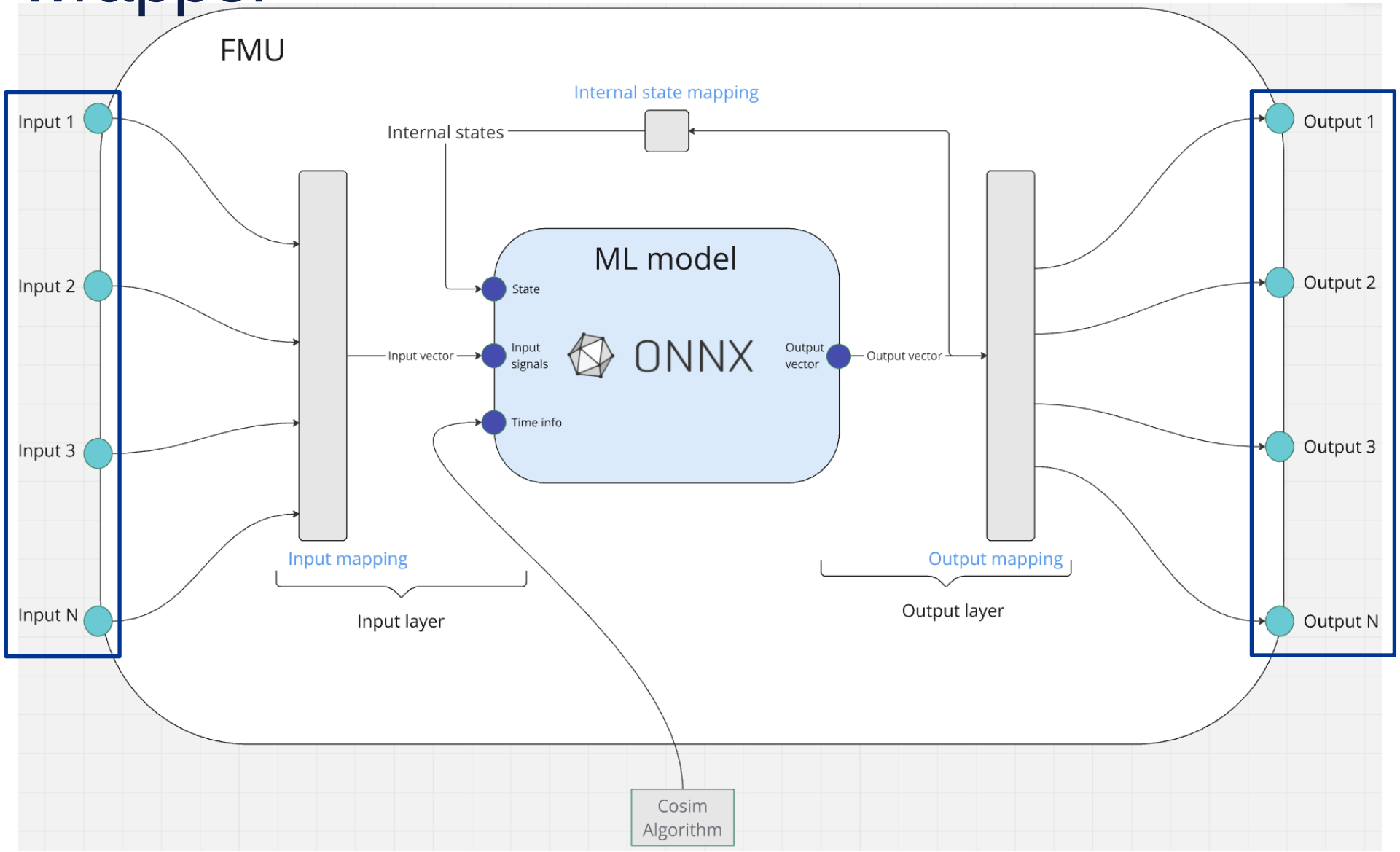
Specifying inputs and outputs



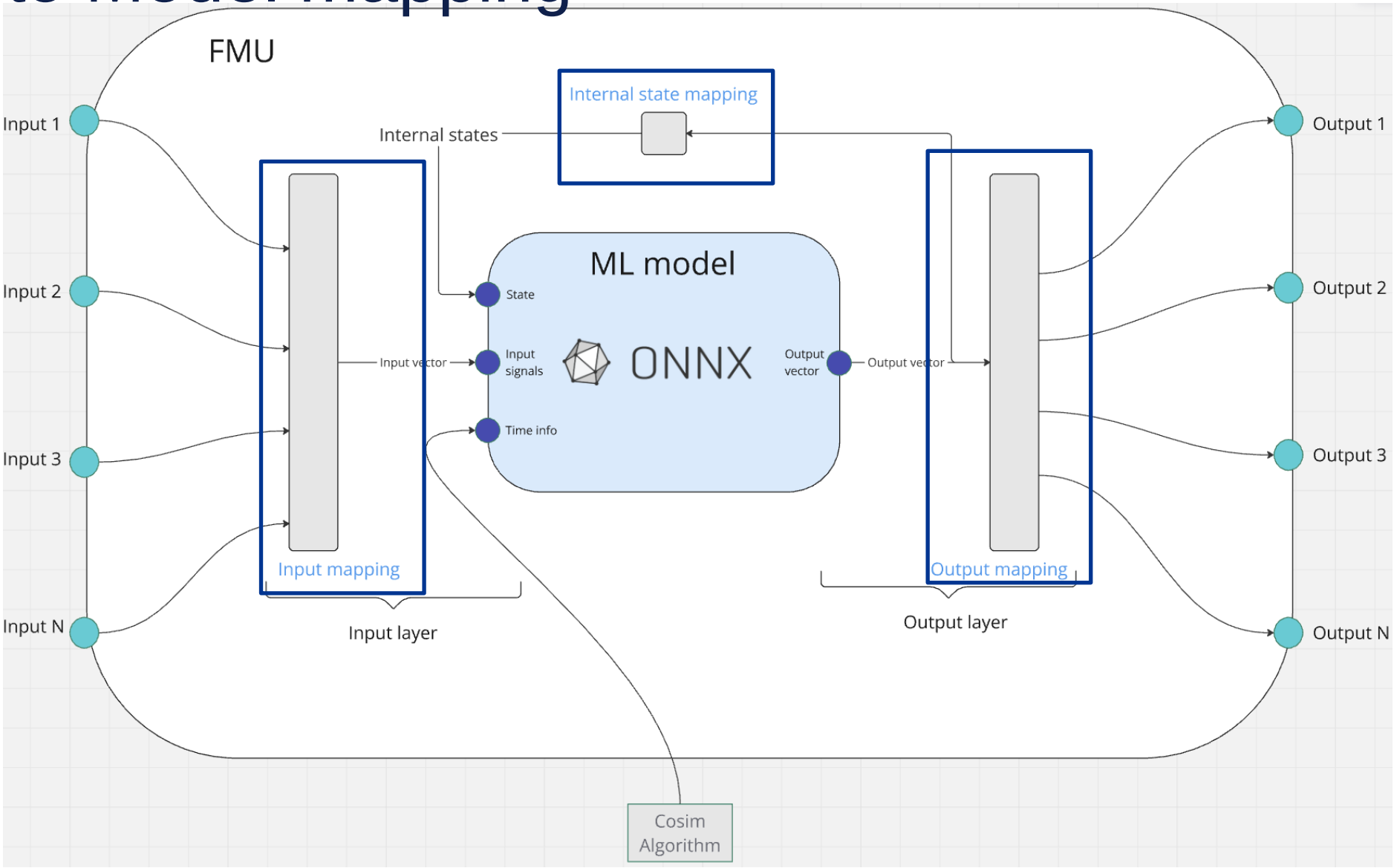
Input options



FMU wrapper


FMU
Internal state mapping
Internal states
Input 1
Input 2
Input 3
Input N
ML model
State
Input signals
Input vector
ONNX
Output vector
Output vector
Time info
Output 1
Output 2
Output 3
Output N
Input mapping
Input layer
Output mapping
Output layer
Cosim Algorithm

FMU to Model mapping



Interface specification file

```
{
    "name": "WindToPower",
    "description": "A Machine Learning based FMU that outputs the estimated power output of a wind turbine given the wind speed and direction.",
    "inputs": [
        {
            "name": "windSpeed",
            "description": "The speed of the wind",
        },
        {
            "name": "windDirection",
            "description": "The direction of the wind",
        }
    ],
    "parameters": [],
    "outputs": [
        {
            "name": "power",
            "description": "The estimated wind turbine power output",
        }
    ],
    "states": []
}
```

Interface specification file


```
{
    "name": "WindToPower",
    "description": "A Machine Learning based FMU that outputs the estimated power output of a wind turbine given the wind speed and direction.",
    "inputs": [
        {
            "name": "windSpeed",
            "description": "The speed of the wind",
            "agentInputIndexes": [
                "0"
            ]
        },
        {
            "name": "windDirection",
            "description": "The direction of the wind",
            "agentInputIndexes": [
                "1"
            ]
        }
    ],
    "parameters": [],
    "outputs": [
        {
            "name": "power",
            "description": "The estimated wind turbine power output",
            "agentOutputIndexes": [
                "0"
            ]
        }
    ],
    "states": []
}
```

Using arrays

```
"inputs": [
    {
        "name": "position",
        "description": "position with [x, y, z] coordinates",
        "agentInputIndexes": [
            "0:3"
        ],
        "is_array": true,
        "length": 3
    },
    {
        "name": "velocity",
        "description": "velocity in the [x, y, z] directions",
        "agentInputIndexes": [
            "3:6"
        ],
        "is_array": true,
        "length": 3
    }
],
```

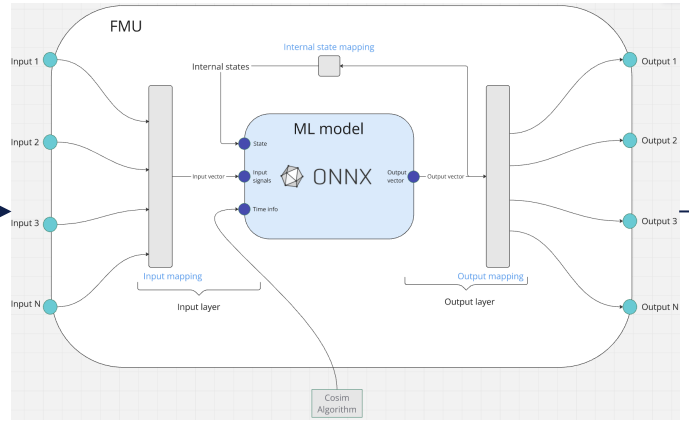

DNV

MLFMU: Compile FMU from .onnx & .json

model.onnx

interface.json

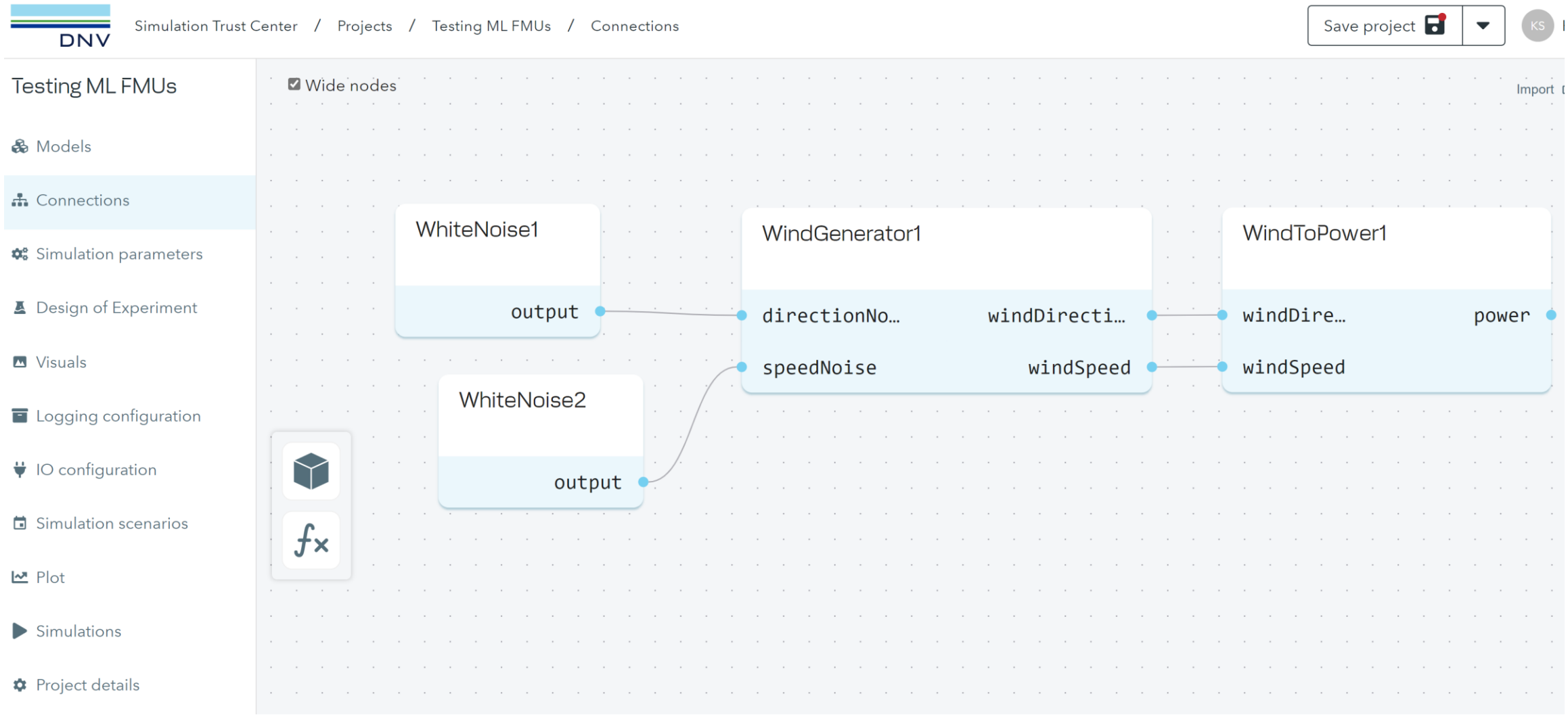
```
> mlfmu build
```



model.fmu

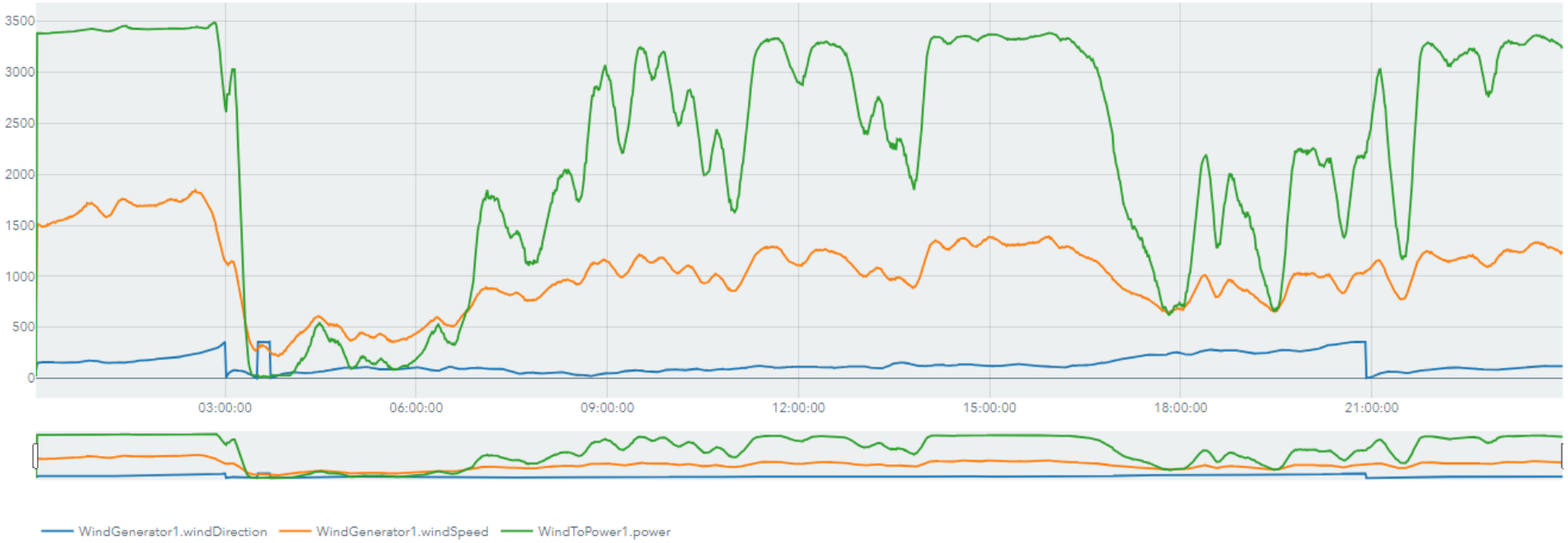
Using the ML FMUs

Example: Run FMUs in the Simulation Trust Center



DNV

STC results



Start time [s] 0

End time [s] 86400 Set max

Window period 60s

Load Data

Variable	Current	Mean	Max	Min	Std dev	Scaling
WindGenerator1.windDirection	117.25	135.29	358.48	0.21	78.31	None
WindGenerator1.windSpeed	1232.38	1039.20	1844.73	217.02	359.91	x 100
WindToPower1.power	3253.65	2242.37	3489.56	11.06	1131.60	None

Recorded models

- WhiteNoise2
- WhiteNoise1
- WindGenerator1
- WindToPower1

Additional functionality

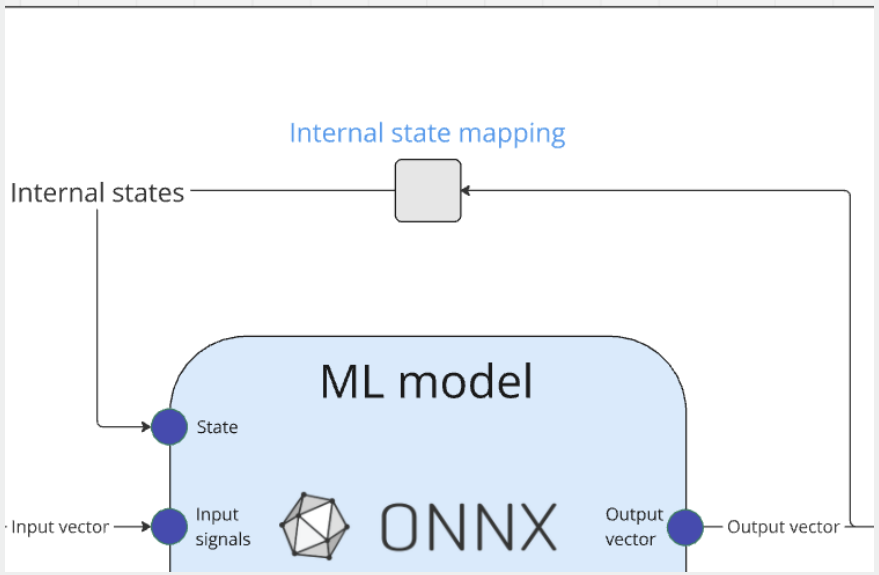
Using state

Why?

- Remember signals in the past
- Internal states to the ML model

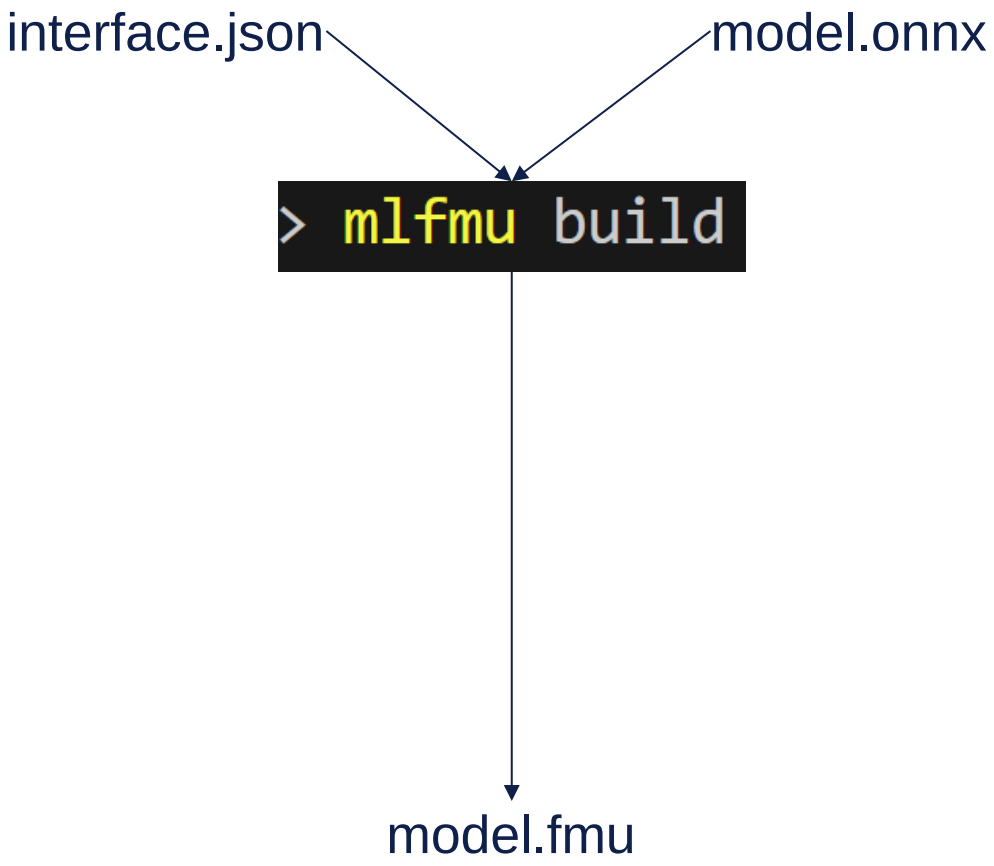
How?

- Needs to be outputted from the ML model
  - Some used for FMU outputs
  - Others used as states
  - Or both
- Modify existing ML model with a wrapper

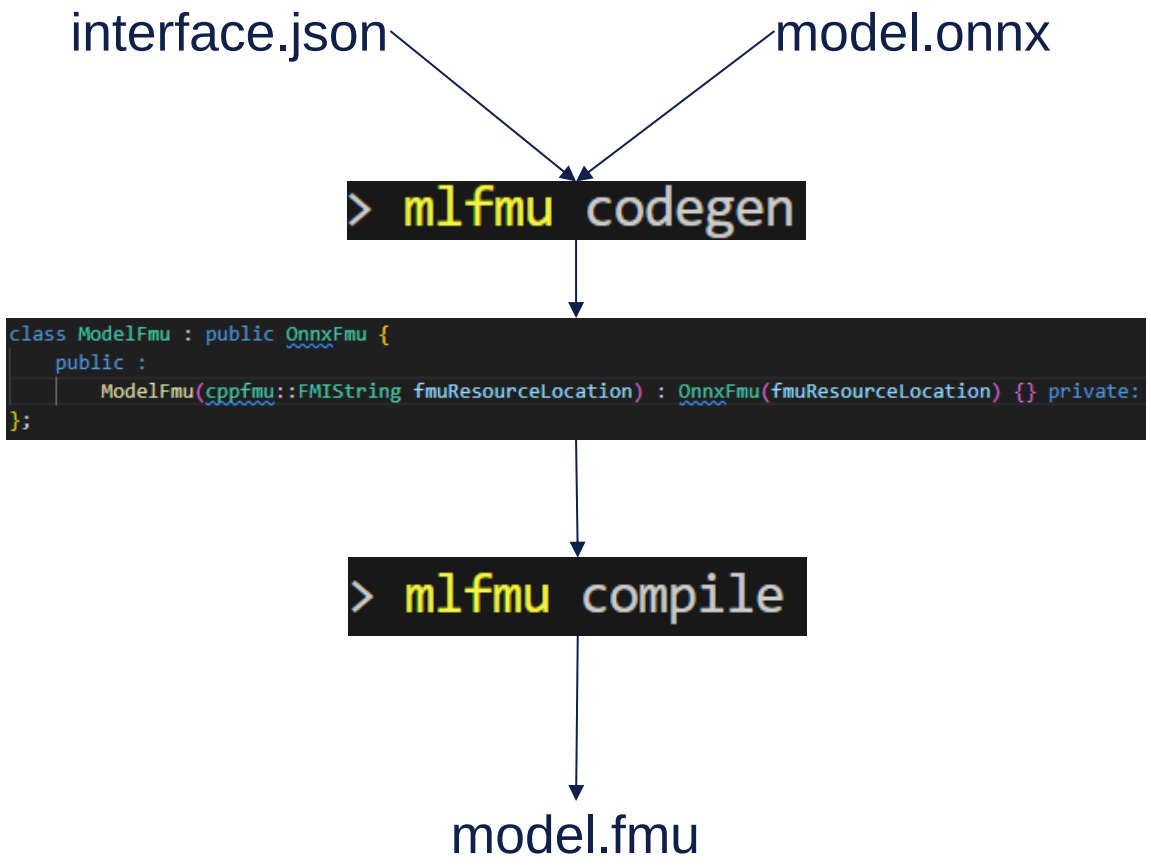



```
"states": [
    {
        "name": "previousOutput",
        "agentOutputIndexes": [
            "0:3"
        ]
    },
    {
        "name": "previousPosition",
        "agentOutputIndexes": [
            "3:6"
        ]
    },
    {
        "name": "previousVelocity",
        "agentOutputIndexes": [
            "6:9"
        ]
    },
    {
        "name": "statesForModel",
        "agentOutputIndexes": [
            "9:42"
        ]
    }
```

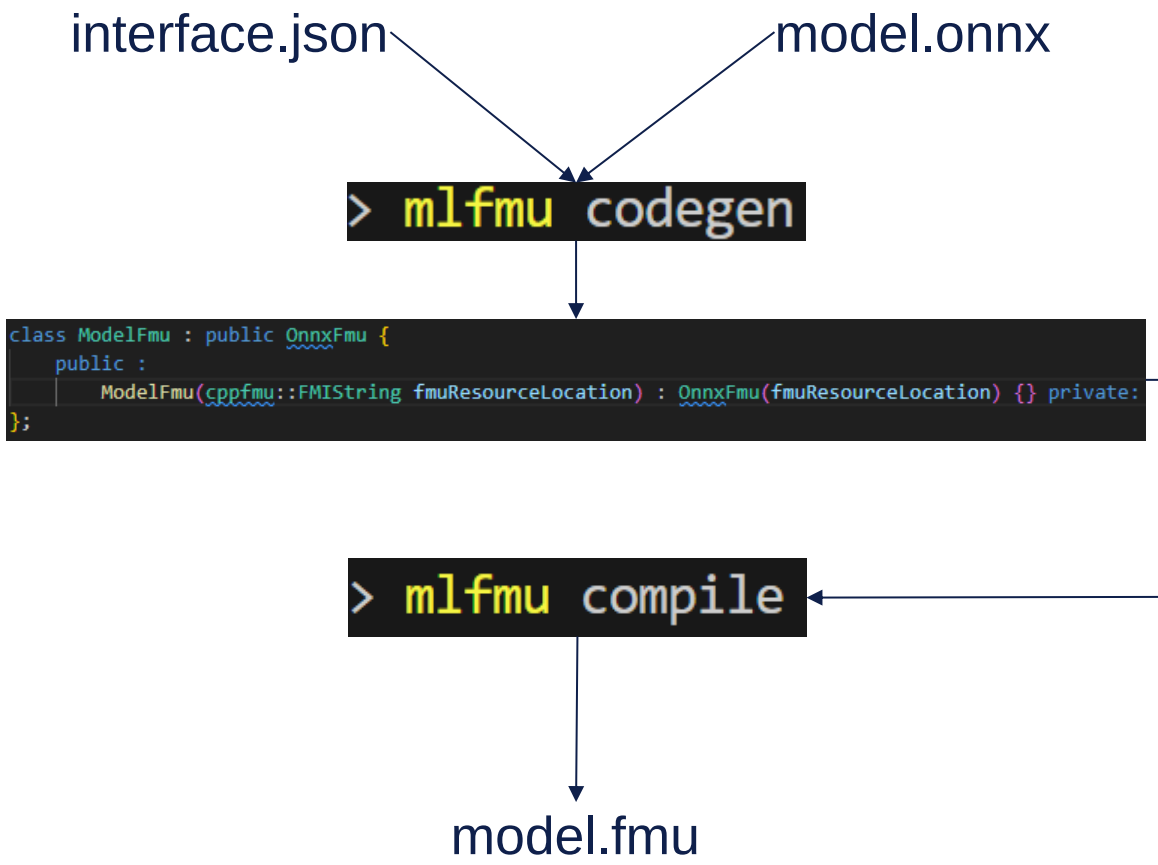
Altering C++ source code



Split up build command



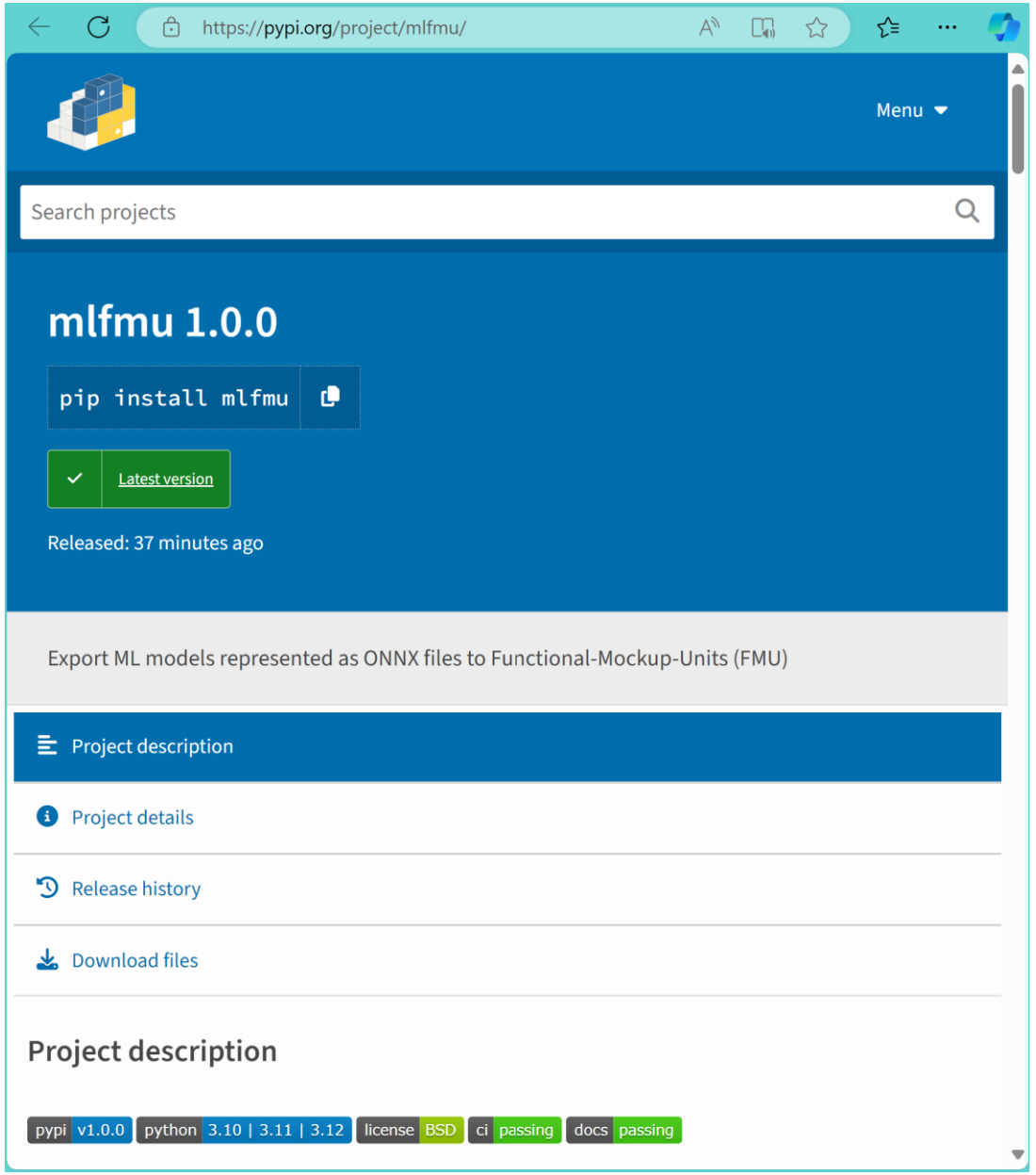
Implement custom behaviour before and after DoStep




```cpp
class ModelFmu : public OnnxFmu
{
public:
    ModelFmu(cppfmu::FMIString fmuResourceLocation)
        : OnnxFmu(fmuResourceLocation)
    { }

    bool DoStep(
        cppfmu::FMIReal currentCommunicationPoint,
        cppfmu::FMIReal dt,
        cppfmu::FMIBoolean newStep,
        cppfmu::FMIReal& endOfStep) override
    {
        // Implement custom behaviour here ...

        // Modify inputs and parameters

        // Call the OnnxFmu::DoStep function
        bool onnxDoStepSuccessful = OnnxFmu::DoStep(
            currentCommunicationPoint,
            dt,
            newStep,
            endOfStep);

        if (!onnxDoStepSuccessful) {
            return false;
        }

        // Modify outputs

        return true;
    }

private:
};
```

Future developments

- It is published as a package: mlfmu · PyPI
  - `pip install mlfmu`
- The code is open source: https://www.github.com/dnv-opensource/mlfmu
- Version 1.0 is not the final version
  - No “one size fits all”
  - We tried to make it generic and easy to use.
  - Just try to use it, find what is missing, and let us know!



Thank you!

https://pypi.org/project/mlfmu/
https://www.github.com/dnv-opensource/mlfmu

Kristoffer Skare <kristoffer.skare@dnv.com>
Stephanie Kemna <stephanie.kemna@dnv.com>
Jorge Luis Mendez <jorge.luis.mendez@dnv.com>
Cesar Ramos de Carvalho <cesar.de.carvalho@dnv.com>

Simulation Technologies team, DNV

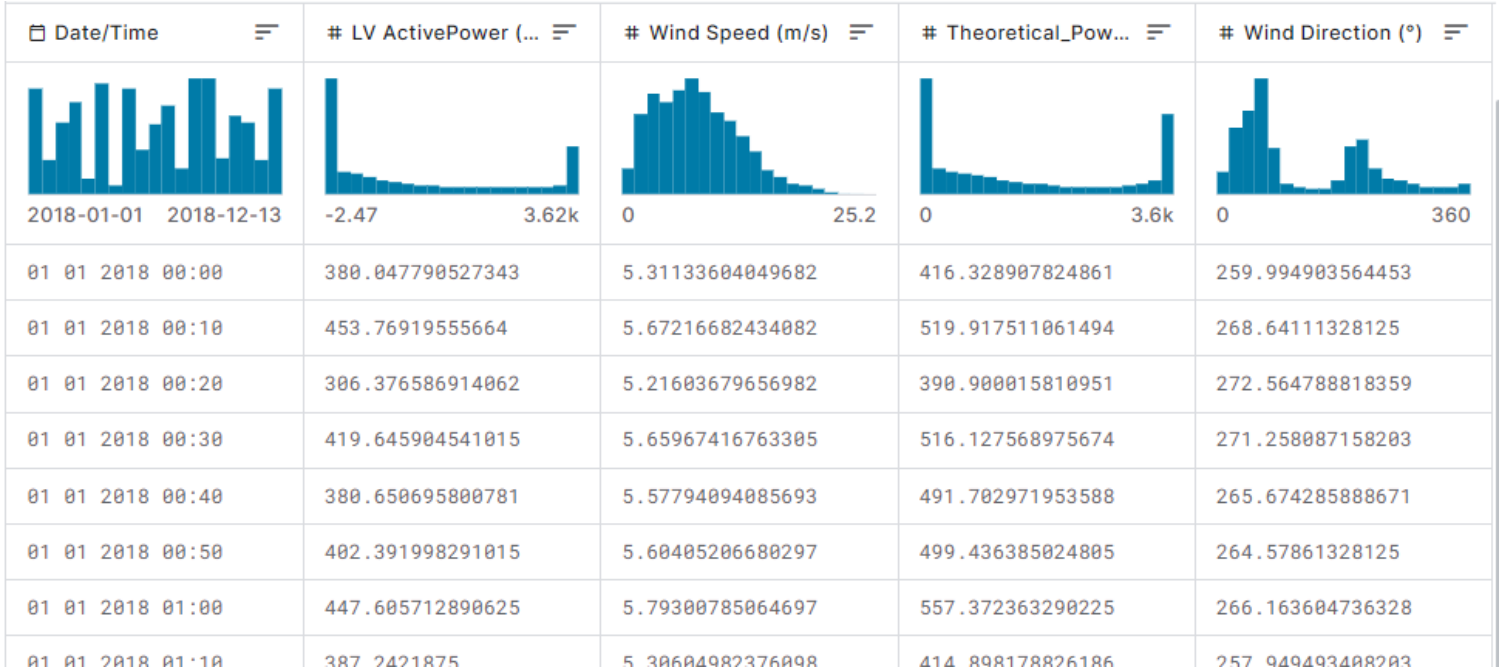
www.dnv.com

Example: The data

Example: https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset/data

Data:

- recorded date/time,
- active power generated by the turbine for that moment,
- wind speed,
- theoretical power curve values that the turbine generates given the wind speed,
- wind direction.



Date/Time	# LV ActivePower (...	# Wind Speed (m/s)	# Theoretical_Pow...	# Wind Direction (°)
2018-01-01 2018-12-13	-2.47 3.62k	0 25.2	0 3.6k	0 360
01 01 2018 00:00	380.047790527343	5.31133604049682	416.328907824861	259.994903564453
01 01 2018 00:10	453.76919555664	5.67216682434082	519.917511061494	268.64111328125
01 01 2018 00:20	306.376586914062	5.21603679656982	390.900015810951	272.564788818359
01 01 2018 00:30	419.645904541015	5.65967416763305	516.127568975674	271.258087158203
01 01 2018 00:40	380.650695800781	5.57794094085693	491.702971953588	265.674285888671
01 01 2018 00:50	402.391998291015	5.60405206680297	499.436385024805	264.57861328125
01 01 2018 01:00	447.605712890625	5.79300785064697	557.372363290225	266.163604736328
01 01 2018 01:10	387.2421875	5.30604982376098	414.898178826186	257.949493408203

Example: Create a simple ML model

Goal: train a simple neural network to predict the active power production (model output), given a certain wind speed and wind direction (model input).

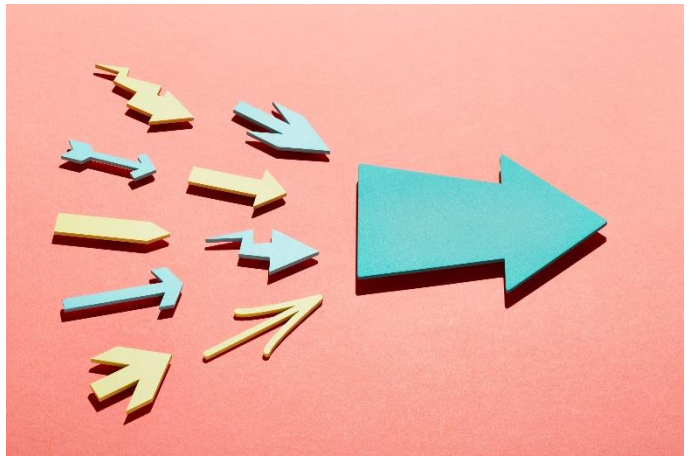
```
# 2 hidden layers, 32 units each
self.dense1 = tf.keras.layers.Dense(32, activation=tf.nn.relu)
self.dense2 = tf.keras.layers.Dense(32, activation=tf.nn.relu)
# output layer
self.dense3 = tf.keras.layers.Dense(1, activation=None)

power_model.fit(train_power, validation_data=val_power, epochs=num_epochs_power)
```


DNV

Example: Convert to ONNX

Use the appropriate converting library:

- sklearn-onnx: converts models from scikit-learn,
- tf2onnx: converts models from tensorflow,
- onnxmltools: converts models from lightgbm, xgboost, pyspark, libsvm
- torch.onnx: converts model from pytorch.



DNV

Altering C++ source code - Detailed example

interface.json

model.onnx

```
> mlfmu codegen
```

```
class ModelFmu : public OnnxFmu {
    public :
        ModelFmu(cppfmu::FMIString fmuResourceLocation) : OnnxFmu(fmuResourceLocation) {} private:
};
```

```
> mlfmu compile
```

model.fmu

```
class ModelFmu : public OnnxFmu
{
public:
    ModelFmu(cppfmu::FMIString fmuResourceLocation)
        : OnnxFmu(fmuResourceLocation)
    { }

    bool DoStep(cppfmu::FMIReal currentCommunicationPoint, cppfmu::FMIReal dt, cppfmu::FMIBoolean newStep,
        cppfmu::FMIReal& endOfStep) override
    {
        // Modify the inputs or parameters before the onnx model is run

        // Get the values needed for modification
        const cppfmu::FMIReal values[1] = {0.0};
        const cppfmu::FMIValueReference value_references[1] = {0};
        GetReal(value_references, 1, values);

        // Modify the values as needed

        // Set the modified values
        SetReal(value_references, 1, values);

        bool onnxDoStepSuccess = OnnxFmu::DoStep(currentCommunicationPoint, dt, newStep, endOfStep);
        if (!onnxDoStepSuccess) {
            return false;
        }

        // Modify the outputs or states after the onnx model is run

        // Get the values needed for modification
        const cppfmu::FMIValueReference output_value_references[1] = {1};
        cppfmu::FMIReal output_values[1] = {0.0};
        GetReal(output_value_references, 1, output_values);

        // Modify the values as needed

        // Set the modified values
        SetReal(output_value_references, 1, output_values);

        // Complete the modification by saving any changes to the states as well
        SetOnnxStates();

        return true;
    }

private:
};
```


DNV