Hybrid Simulation Models for Embedded Applications: A Modelica and eFMI approach

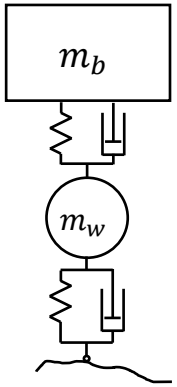
Tobias Kamp[1], Christoff Bürger[2], Johannes Rein[1], Jonathan Brembeck[1]

[1]DLR - Institute of Vehicle Concepts, Germany

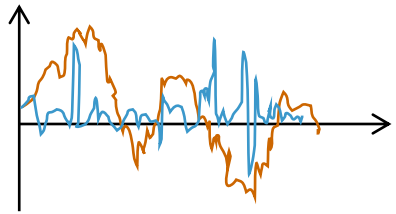
[2]Dassault Systèmes AB, Sweden, Christoff.Buerger@3ds.com

Hybrid Simulation Models
Concept

System

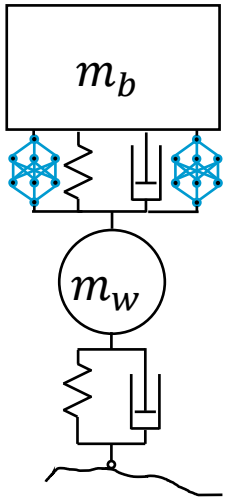


ODE



Neural ODE

Hybrid Neural ODE Model



$$\dot{x} = f(x, u, t, \Theta)$$
$$y = g(x, u, t, \Theta)$$

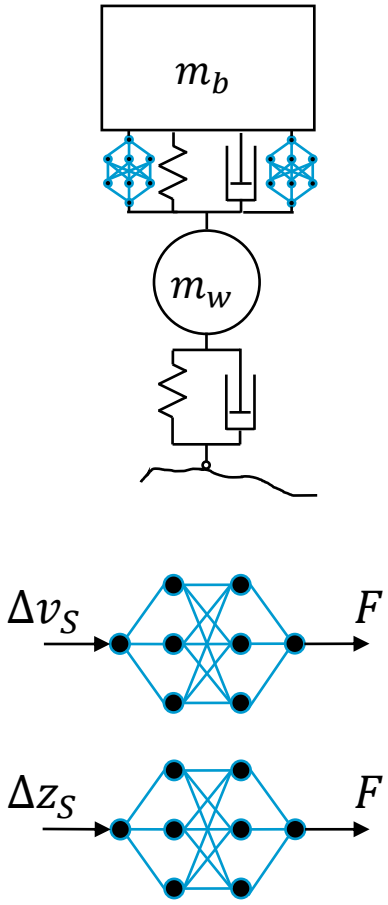
Why?

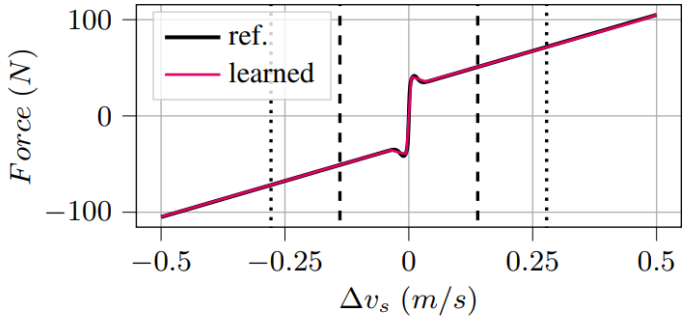
- **Enhance predictions**
- **Enhance performance**

Case-study
Hybrid Quarter Vehicle Model

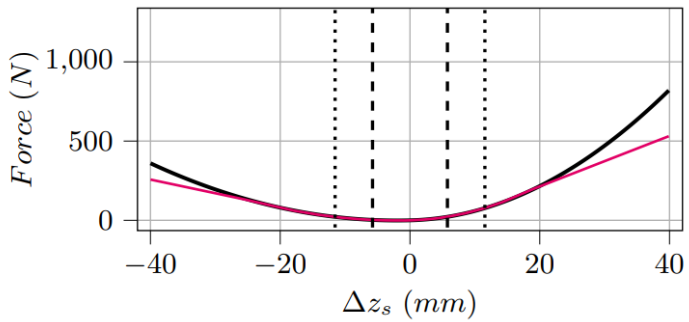
Previous work:

- Linear Quarter Vehicle Model + neural networks that learn missing nonlinear effects
- ***Physics-enhanced Neural ODE (PeN-ODE):*** meaningful combination of physics and neural networks





(a) Friction Force



(b) Spring Characteristic

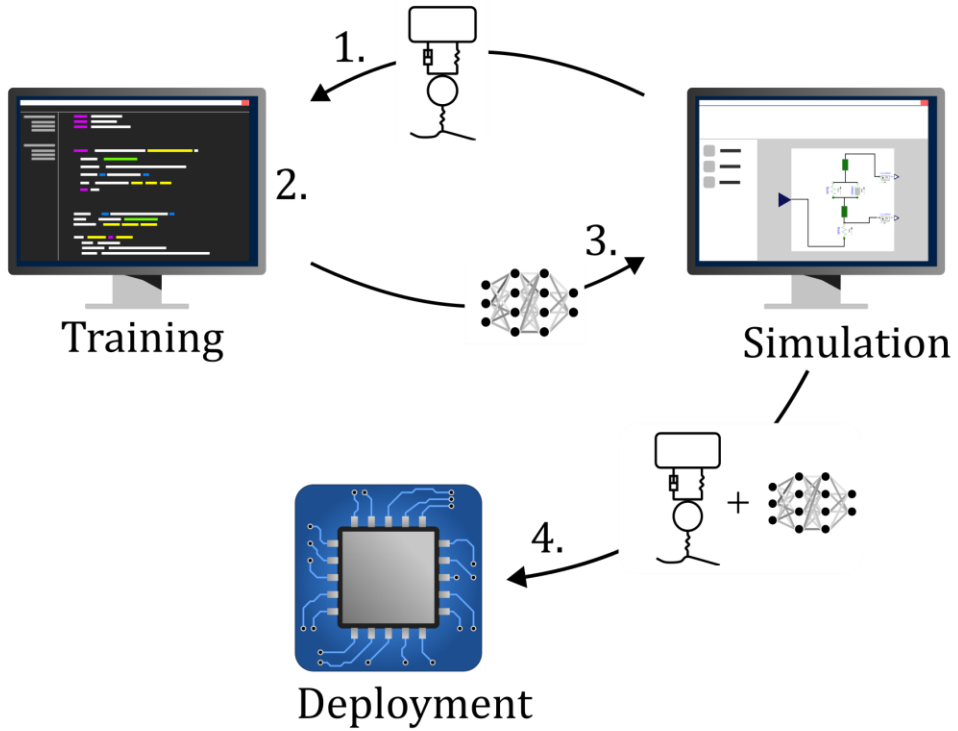
Trained forces as part of the PeN-ODE.
Kamp et al. 2023[1]

[1]https://elib.dlr.de/200100/

Neural ODE Workflow

Related Work

1. Transfer of the physics model to the training framework
   - using FMI[1], code generation, …
2. Training of the NODE with measurement data
   - Julia[2], Pytorch[3], …
3. Transfer trained components to the simulation environment
   - FMI[1], SmartIInt[4], …
4. Deployment of the Neural ODE on embedded target (control applications)
   - (this work)



[1]https://github.com/ThummeTo/FMI.jl

[2]https://julialang.org/

[3]https://pytorch.org/

[4]https://github.com/xrg-simulation/SMArtIInt

Requirements of embedded applications

- MISRA:C 2023 compliance (The MISRA Consortium 2023)
- **Restricted dependencies** on libraries and frameworks
- Worst **execution-time** and **memory-consumption** guarantees
- Self-dependent implementation, with **inline-integrated ODEs** such that only linear solver calls are required
- **Error-handling concepts**, especially in case of unexpected positive or negative Infinity and NaN results of floating point operations
- **Software-, Processor- and Hardware-in-the-Loop tests** which are derived from Model in the Loop tests in the simulation environment

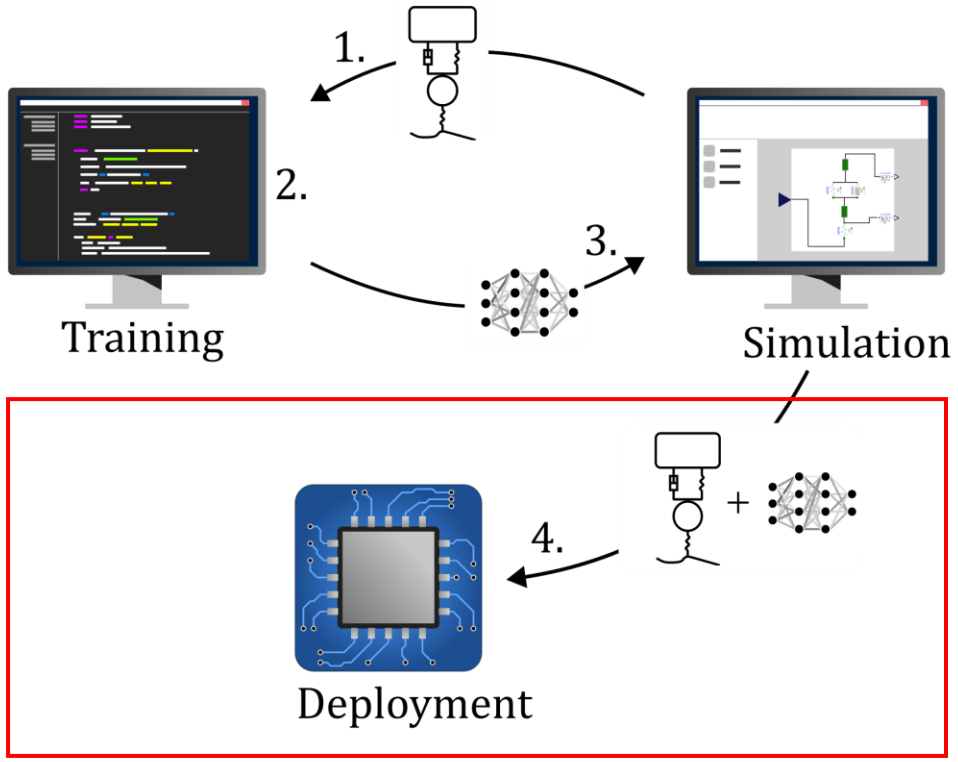
Contribution

This work focuses on the application of Neural ODEs **after** the training

- Simulation with Modelica
- Deployment on embedded hardware – and meeting the aforementioned requirements
- Recalibration of neural networks during runtime

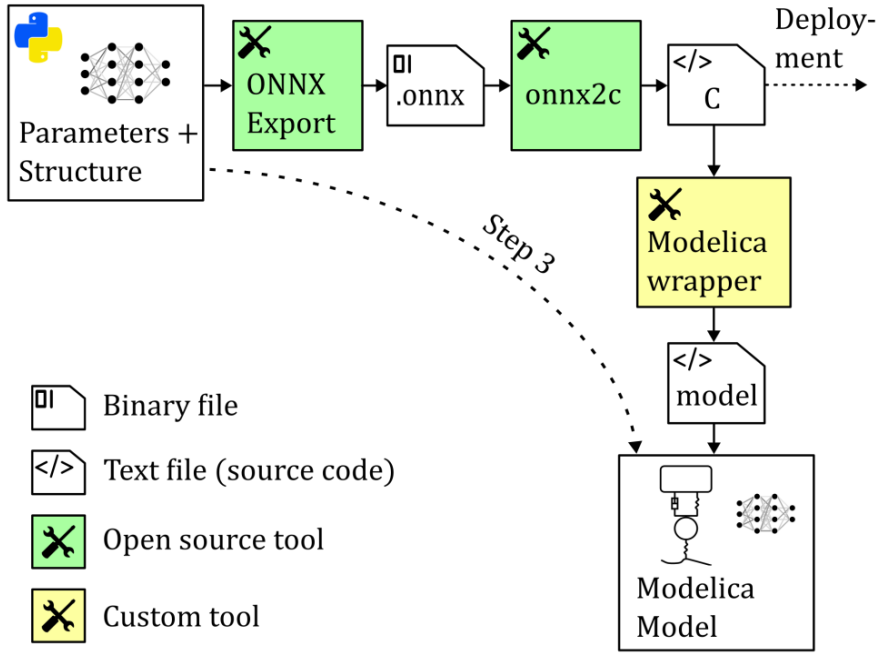
We propose two workflows:

1. C embedding via ONNX
2. „Native Modelica“



C embedding via ONNX
Concept

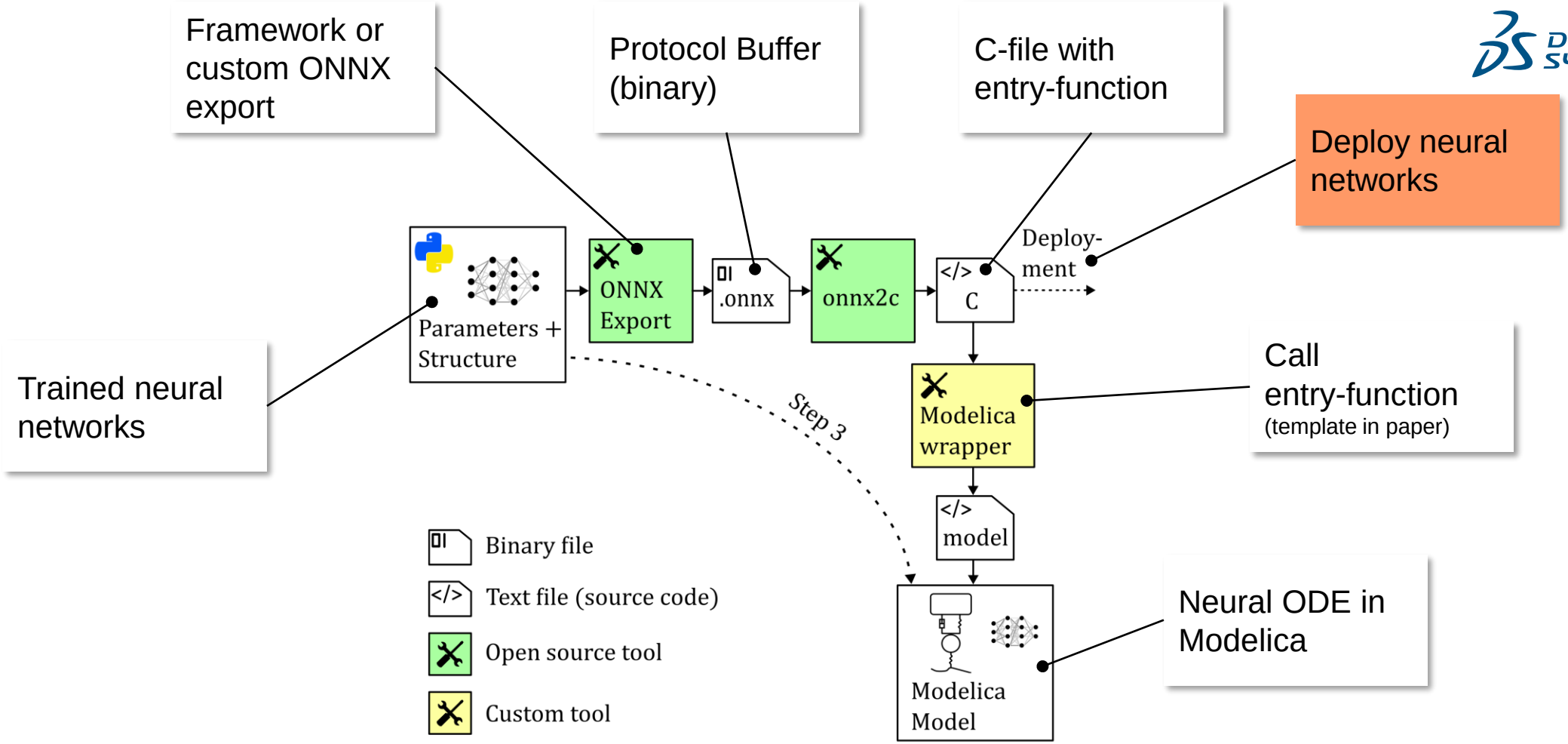
- Leverage ONNX[1] (Open Neural Network Exchange) to export trained neural networks
- Generate „hardware-ready“ C code of the neural networks with onnx2c[2]
- Wrap the generated code into a Modelica package via the *External C* interface



[1]https://onnx.ai/

[2]https://github.com/kraiskil/onnx2c

C embedding via ONNX
Workflow



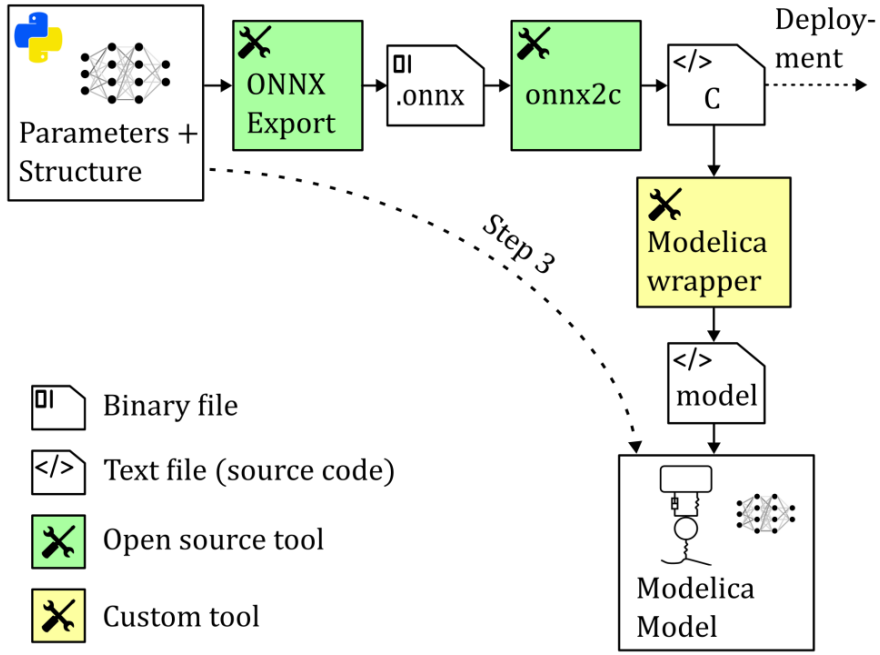
C embedding via ONNX
Evaluation

Benefits

- ONNX is already widely supported
- You obtain plain C code of the NNs
- Incorporation in Modelica is easy

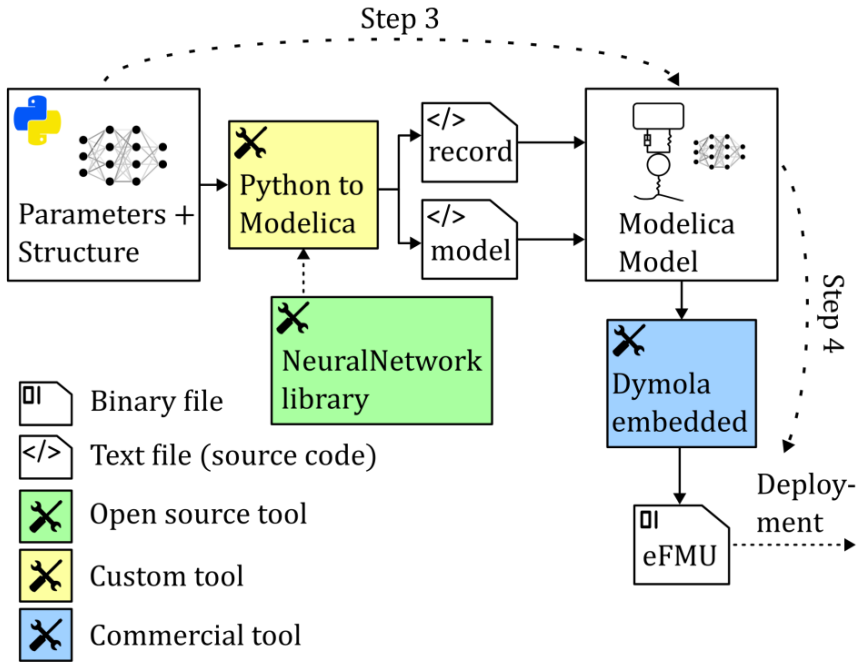
Shortcomings

- Detour via onnx2c
- Once compiled, the neural networks are static
- Deploying the *whole* Neural ODE
  is not straightforward
  - Just the NNs are available as embedded code;
    The physics-equations & integration with NNs is an open issue



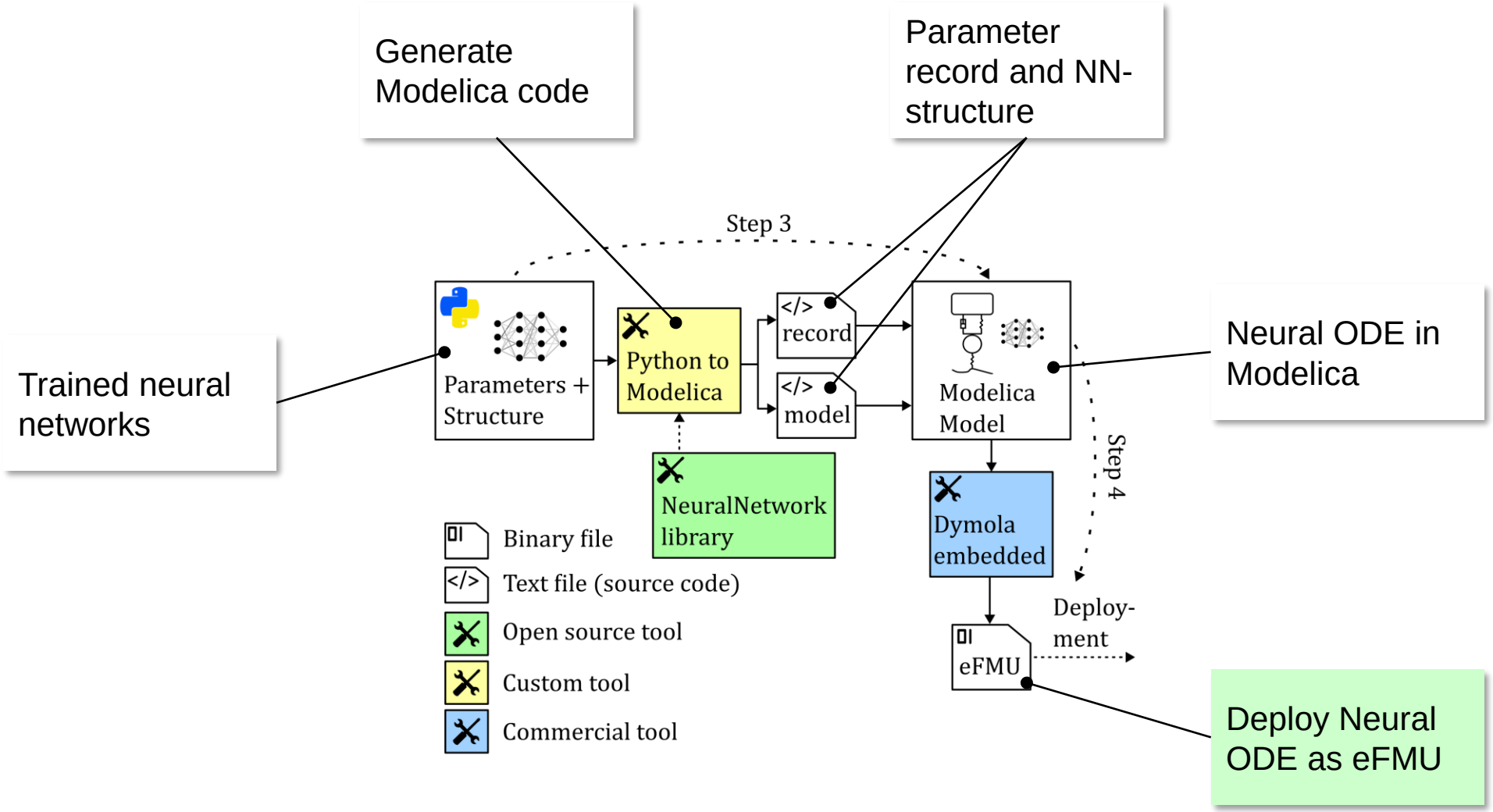
Native Modelica
Concept

- Generate Modelica code targeting the Modelica NeuralNetwork Library[1] from your training Framework
- Implement the Neural ODE in Modelica
- Export the Neural ODE as eFMU with Dymola embedded for deployment



[1] https://github.com/AMIT-HSBI/NeuralNetwork

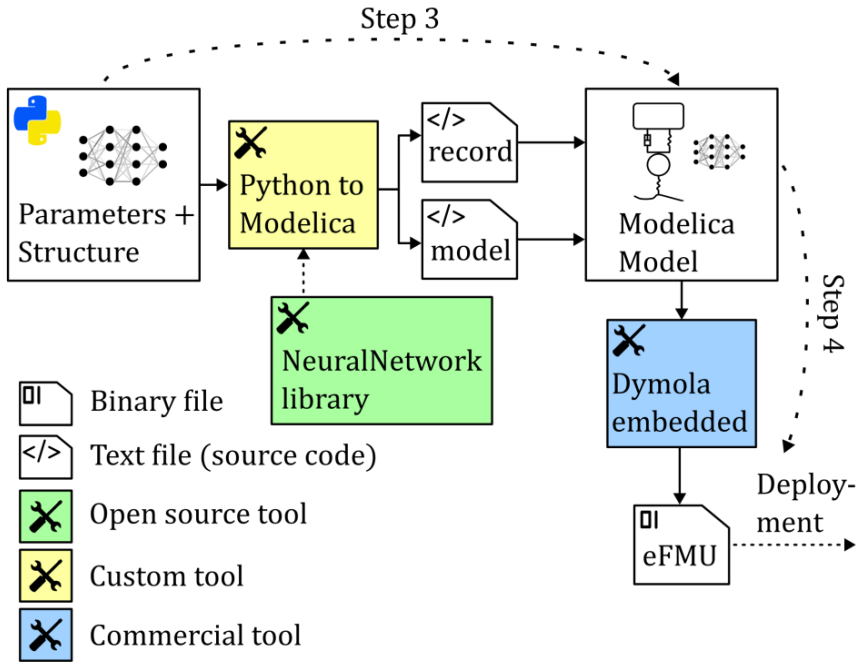
Native Modelica
Workflow



Native Modelica
Evaluation

Benefits

- You obtain native Modelica neural networks with separate parameter records
- Recalibration of neural networks during runtime
- Embedded deployment of the *whole* Neural ODE is straightforward, thanks to eFMI

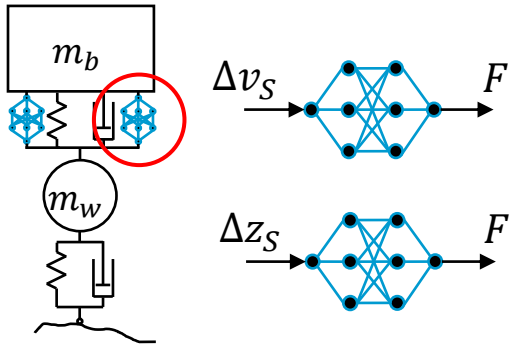


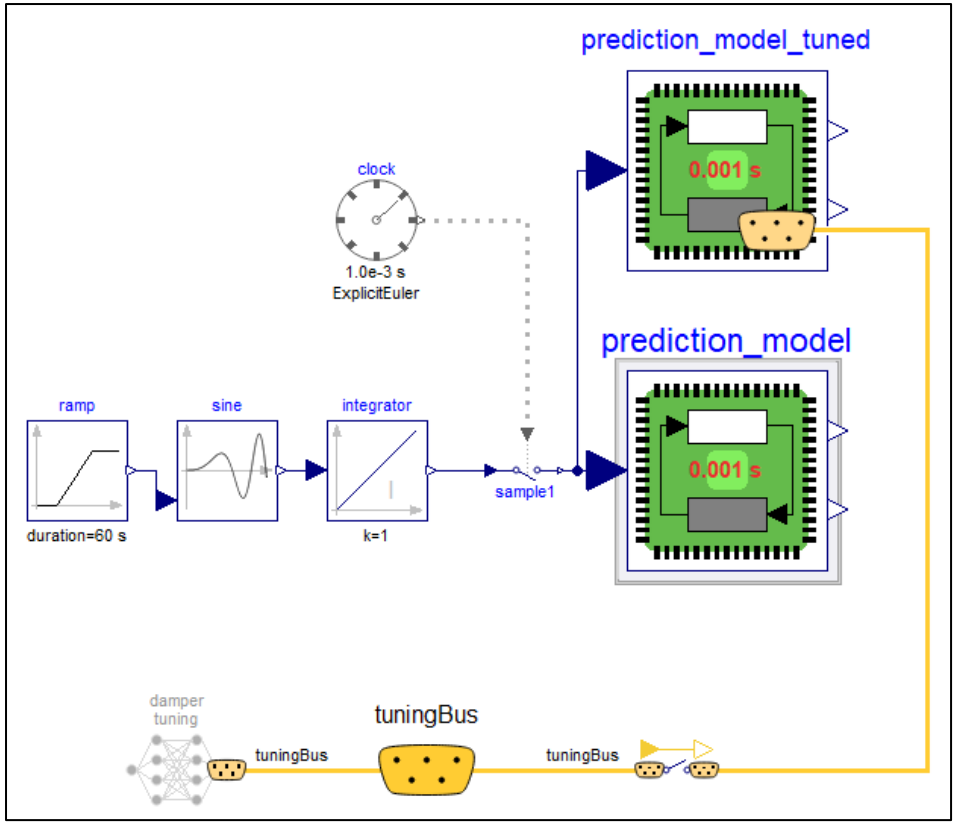
Shortcomings

- Limited functionalities of NeuralNetwork Library
- Python to Modelica compiler not (yet) available
- Dependence on Dymola embedded

Validation
Software in the Loop Setup

- Hybrid Quarter Vehicle Model with two neural networks (each with ≈ 600 parameters)
- Export to Modelica via custom Python to Modelica compiler
- Export Neural ODE as eFMU via Dymola embedded
- Recalibration of the neural damper in SiL-Experiment

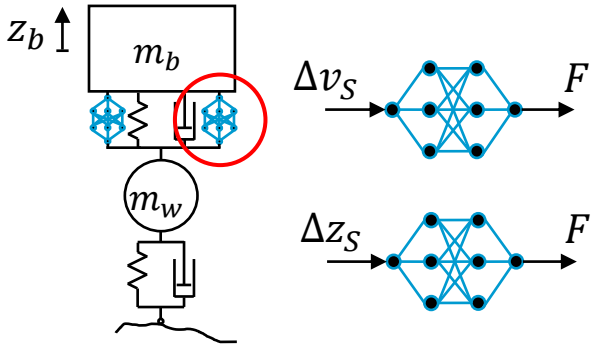


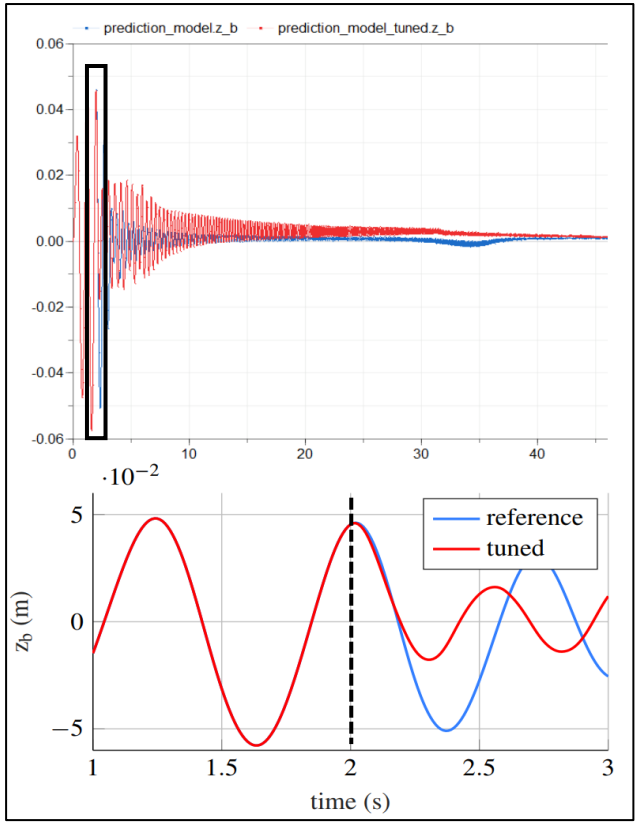


SiL setup of the recalibration Test.

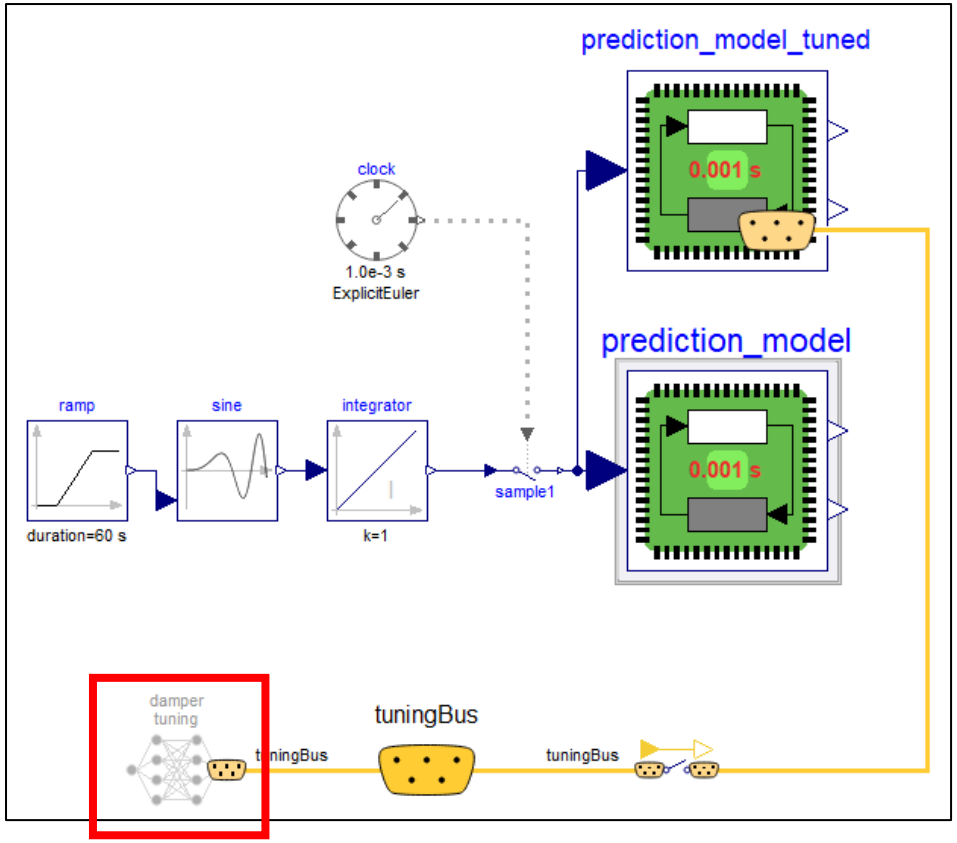
Validation
SiL-Experiment Results

- Damper-tuner provides parameters as time-table
- Recalibration of the neural damper after 2 seconds
- Altered dynamics
- No discontinuity in the states





Exemplary simulation results.



SiL setup of the recalibration Test.

Future Work

- This work is part of the ITEA 4 OpenSCALING[1] project
  - Efficient simulation of Large Scale Systems
  - Standard enhancements for hybrid modelling
  - Avoiding scalarization to decrease code size; This is crucial for embedded applications (already available in Dymola 2026x)

- Standardized online recalibration of neural networks
  - SSP[2] defines varying parameter sets, but it has no notion of timing

- Pending: Hardware in the Loop and real world tests

[1]https://itea4.org/project/openscaling.html

[2]https://ssp-standard.org/

Avoid scalarization of tensorflows

Multi-dimensional GALEC code generated by Dymola 2026x

- Dymola 2026x provides *prototype* facilities to preserve multi-dimensional equations and variables (avoid scalarization)

```
'damperNN.layer_2.y[1]' := max(0.0,
    ((((((((((((((((self.'damperNN.layer_2_weights[1,1]' * 'damperNN.layer_1.y[1]')
    + (self.'damperNN.layer_2_weights[1,2]' * 'damperNN.layer_1.y[2]'))
    + (self.'damperNN.layer_2_weights[1,3]' * 'damperNN.layer_1.y[3]'))
    + (self.'damperNN.layer_2_weights[1,4]' * 'damperNN.layer_1.y[4]'))
    ...
    + (self.'damperNN.layer_2_weights[1,16]' * 'damperNN.layer_1.y[16]'))
    + self.'damperNN.params.layer_2_bias[1]'));
'damperNN.layer_2.y[2]' := max(0.0,
    ((((((((((((((((self.'damperNN.layer_2_weights[2,1]' * 'damperNN.layer_1.y[1]')
    + (self.'damperNN.layer_2_weights[2,2]' * 'damperNN.layer_1.y[2]'))
    + (self.'damperNN.layer_2_weights[2,3]' * 'damperNN.layer_1.y[3]'))
    ...
    + self.'damperNN.layer_2_bias[2]'));
...
```

Scalarized GALEC code for QVM ≈117 KB

```
'damperNN.denseLayer2.y' :=
    ((self.'damperNN.layer_2_weights' * 'damperNN.denseLayer1.y')
    + self.'damperNN.layer_2_bias');
for i in 1 : 16 loop
    'damperNN.denseLayer2.y'[ i ] :=
        max(0.0, 'damperNN.denseLayer2.y'[ i ]);
end for;
```

Multi-dimensional GALEC code for QVM ≈2.77 KB

- Neural network parameters now comprise the bulk of GALEC code

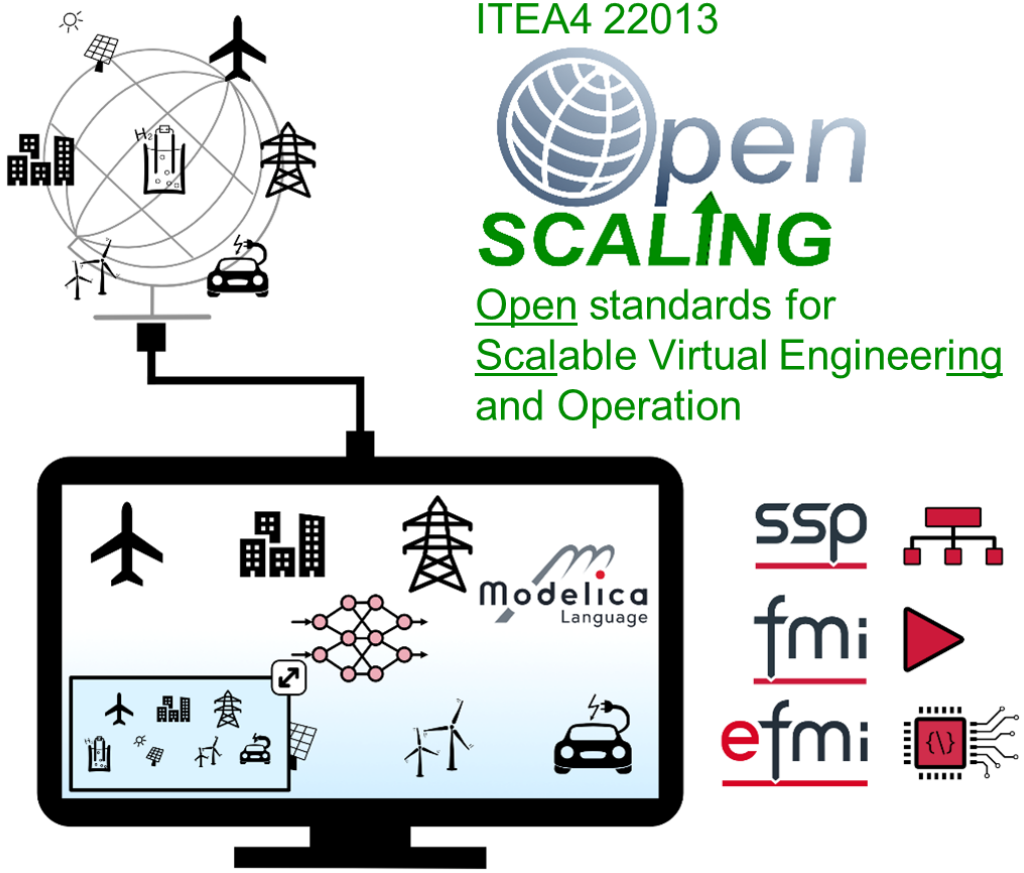
  Weights & biases for QVM ≈29.3 KB GALEC

  *≈4.8 KB data memory (2 * 600 parameters * 4 bytes for 32-Bit floating-point precision)*

Conclusion

- Two workflows to simulate and deploy trained **Neural ODEs**
- C code from **ONNX** via onnx2c: modeling & simulation in Modelica with minimal dependencies; but only NN-parts, not whole PeN-ODE
- Native Modelica generation enabling eFMI export of whole PeN-ODE
  - Meets requirements of **embedded applications**
  - Supports recalibration of neural ODE
- Neural QVM model open-source
  - `eFMI_TestCases` Modelica library, `M11_NeuralQVM` example
  - Since version 1.0.2, released September 4, 2025
- In OpenSCALING, we are advancing **hybrid modelling** capabilities of Modelica, FMI and eFMI

Acknowledgments



ITEA4 22013

Open SCALING

Open standards for Scalable Virtual Engineering and Operation

ssp

fmi

efmi

ITEA 4

DASSAULT SYSTEMES

SPONSORED BY THE

Federal Ministry of Education and Research

VINNOVA

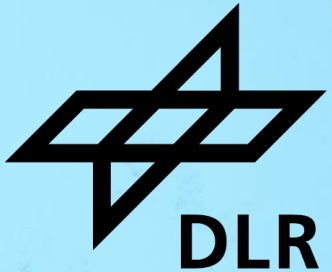
Sweden's Innovation Agency

This work has been supported by the Swedish Agency for Innovation Systems (Vinnova[1], grant number 2023- 00969) and the German Federal Ministry of Education and Research (BMBF[2], grant number FKZ 01IS23062A) within the ITEA 4 Project Open standards for SCALable virtual engineerING and operation (OpenSCALING[3], ITEA Project 22013).

[1]https://www.vinnova.se [2]https://www.bmbf.de [3]https://itea4.org/project/openscaling.html

THANK YOU!

Feedback & Questions!



Imprint

Topic	**Hybrid Simulation Models for Embedded Applications: A Modelica and eFMI approach**
	16th International Modelica & FMI Conference September 8 – 10, 2025, Lucerne, Switzerland Presentation of scientific paper (DOI: 10.3384/ecp218545)
Date	September 10, 2025
Authors	Tobias Kamp, Christoff Bürger, Johannes Rein, Jonathan Brembeck
Institute	Institute of Vehicle Concepts
Licence	All images “DLR (CC BY-NC-ND 3.0)” unless otherwise stated