Br(e)aking the Boundaries of Physical Simulation Models: Neural Functional Mock-up Units for Modeling the Automotive Braking System

Tobias Thummerer[1] Fabian Jarmolowitz[2] Daniel Sommer[2] Lars Mikelsons[1]

[1]Chair of Mechatronics, University of Augsburg, Germany, {tobias.thummerer,lars.mikelsons}@uni-a.de
[2]Robert Bosch GmbH, Germany, {fabian.jarmolowitz,daniel.sommer4}@de.bosch.com

Abstract

Testing real hardware and simulation models in combination in a software- or hardware-in-the-loop set-up is challenging. One of the key factors is the high demand for accuracy in the simulation model. If classical modeling based on physical principles is not sufficient to reach the desired level of accuracy, *hybrid modeling*, the combination of physical simulation models and machine learning can be applied. In this publication, we train a hybrid model for a controlled electric motor within the electro-hydraulic braking system of a car under the conditions and restrictions of a real engineering application in the field. We apply state-of-the-art modeling patterns for this, and further extend them with application specific methodological optimizations. Finally, we investigate and show the quantitative and qualitative advantages of the proposed approach for this specific application, resulting in a gain in accuracy by multiple factors.

Hybrid Modeling, Graybox Modeling, Scientific Machine Learning, Functional Mock-Up Unit, Functional Mock-up Interface, Braking System, Automotive

1 Introduction

Most simulation models in the automotive environment are built based on physical equations. This has the advantage that the model builder, as well as other engineers, have an intuitive understanding of the model and its parameters. As these physical equations also have very broad validity, most physical models can also generate correct results outside the required working range (extrapolation), making them robust in various application scenarios. When creating models, equations are chosen on the basis of assumptions, i.e. simplifying assumptions are made. This introduces a modeling error that leads to deviations between the simulation model and the real system. On the other hand, there are machine learning approaches that can theoretically generate arbitrarily accurate results depending on the selected model and model dimension if a sufficient amount of data is given. This flexibility comes at the cost of poor interpretability due to the black-box nature of most machine learning models like arifical neural networks (ANNs). The field of *hybrid modeling*, which is part of *scientific machine learning*, attempts to combine the advantages of both modeling and simulation approaches, e.g., to create more accurate, yet interpretable models.

1.1 Use Case: Bosch Integrated Power Brake

The Integrated Power Brake (IPB) is a vacuum-independent and electro-hydraulic brake control system that combines brake boosting and brake control functions in one piece of hardware. By applying the brake pedal, the control unit calculates the driver's brake request based on internal sensors. This brake request is implemented by activating a pressure build-up unit consisting of an electric motor, a gearbox, and a hydraulic piston. Finally, brake pressure is transferred to the wheel brakes via brake pipes to decelerate the vehicle. For more information on the system, see Robert Bosch GmbH (2025) and Robert Bosch GmbH (2024).

The developed control software is tested using software-in-the-loop simulation. This requires a model of the hardware that generates sufficient quality of results. A very detailed physical model only makes sense to a certain extent, as otherwise the required simulation performance cannot be achieved. Therefore, strict requirements are given for both simulation accuracy and performance. A hybrid modeling approach should be able to fulfill these requirements.

The model used for the use case contains the electric motor, including the mechanical parts of the pressure build-up unit. The physical model calculates the position, speed, torque, and current of the rotor. The motor is controlled by a motor controller that runs software that calculates control signals based on the position of the rotor. This motor controller is implemented in software for the use case considered.

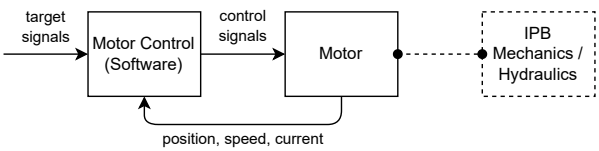

Figure 1. IPB motor control.

The model itself poses various challenges:

- The system is very sensitive with respect to the initial state of the rotor position, as the control signals are calculated based on this. Even small deviations make the motor run *out-of-sync*, as can be observed for the physical model during experiments (s. Sec. 4).
- The control signals are sampled with a frequency of $10^4\,Hz$, triggering time events at this frequency.
- The nonlinear physical model is marginally stable with real zero eigenvalues independent of the system state, so for the entire operation range of the motor.
- Special model conditions (such as reaching end-stop) cause additional state events.

The system consists of two mechanical and two electrical states. The mechanical states for the electric motor are the angular position and the angular speed of the motor rotor. In addition, the inputs to the electric motor are pulse-width modulation (PWM) signals for each motor phase, the motor supply voltage, and the time-varying mechanical load caused by the hydraulic subsystem.

The simulation model is given as functional mock-up unit (FMU), which is introduced in detail in the next section. The use of a FMU for the considered use case not only allows one to simulate the motor model but also to later extend it by a machine learning model without actually sharing the model equations and parameters. This is because the FMU contains only compiled machine code. In fact, the use case shown here was processed without knowing the physical model or its parameters. This demonstrates a significant advantage of the method, as models can be improved by hybrid modeling without disclosing the corresponding intellectual property.

1.2 Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) is an open, tool-independent standard (container format) for exchanging and co-simulating dynamic models, usually in the form of ordinary differential equations (ODEs), across different simulation systems and tools. Developed initially in 2008-2011, FMI has become the de facto industry standard for model exchange and co-simulation, supported by more than 230 different simulation tools (Modelica Association 2020; Bertsch 2022; Modelica Association 2023). FMI supports two main types of FMUs: Co-simulation (CS) and Model-exchange (ME). The ME-FMU offers an interface similar to the right-hand side of a system of ODEs, for example providing the state derivative for a given state. This allows for the complementation of the ODE and its underlying physics with data-driven components from machine learning to improve, for example, the accuracy (Thummerer, Mikelsons, and Kircher 2021).
FMI 3.0 has introduced several features that enables one to integrate AI models in simulation, e.g. by updating parameters much more efficiently than in previous versions. For example, with the added support of adjoint derivatives, FMUs can be integrated in automatic differentiation frameworks, as demonstrated in *FMISensitivity.jl* [1].

1.3 Neural ODEs

In machine learning, using an ANN to learn for the right-hand side of an ODE is referred to as *neural ODE* (R. T. Q. Chen et al. 2018), compare Fig. 2. The gradient required for training neural ODEs is typically determined by back-propagation through the numerical solver (reverse mode automatic differentiation) or by applying adjoint methods. This concept was later extended to support event-ODEs as well (R. T. Chen, Amos, and Nickel 2020). Numerous publications show that neural (ordinary) differential equations are an excellent tool to learn dynamical system behavior (Ramadhan et al. 2023; Tac, Sahli Costabal, and Tepole 2022; Xie, Parlikad, and Puri 2019).

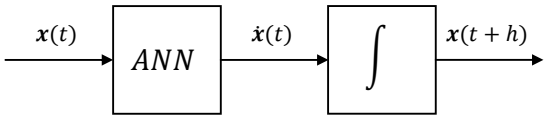

Figure 2. Within neural ODEs, the right-hand side of an ODE is expressed by an ANN and solved by a numerical integration algorithm (ODE solver). Based on a given state $\boldsymbol{x}(t)$, the ANN computes a state derivative $\dot{\boldsymbol{x}}(t)$, which is integrated to the next system state $\boldsymbol{x}(t+h)$ by performing a step of h. Figure adapted from (Thummerer, Kolesnikov, et al. 2023).

1.4 UODE / PeN-ODE

Compared to the concept of neural ODEs, one can conclude that, besides ANNs, other universal approximators are possible as well to approximate the right-hand side of an ODE. Further, the right-hand side can contain a priori knowledge in the form of equations. This concept was introduced as universal ordinary differential equation (UODE) (Rackauckas et al. 2021) together with a software framework. This framework serves as a software foundation for *FMIFlux.jl*[2], which is used within this application. Within the *OpenSCALING* research project (s. Sec. Funding), a special kind of UODE is introduced, combining explicitly ANNs and physical simulation models to be solved by an ODE solver, referred to as physics-enhanced neural ODE (PeN-ODE). Similar and related research was performed in Sorourifar et al. (2023) and Thummerer, Kolesnikov, et al. (2023), using the same terminology. A detailed discussion on PeN-ODEs can also be found in (Hofmann and Mikelsons 2025) and the model investigated in this paper can be interpreted as PeN-ODE.

[1] `https://github.com/ThummeTo/FMISensitivity.jl`
[2] `https://github.com/ThummeTo/FMIFlux.jl`

DOI 10.3384/ecp218575

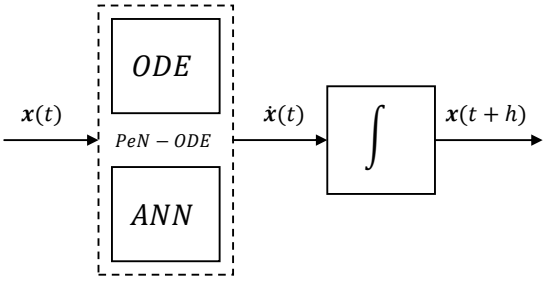

Figure 3. A physics-enhanced neural (PeN)-ODE, which consists of a ODE, an ANN and the ODE solver. Based on a given state $\boldsymbol{x}(t)$, the PeN-ODE computes a state derivative $\dot{\boldsymbol{x}}(t)$, which is integrated to the next state of the system $\boldsymbol{x}(t+h)$. There are no restrictions on how ODE is connected to the ANN.

1.5 Preliminary Work

ME-FMUs offer a similar interface compared to ODEs and can therefore be intuitively replaced with each other on a conceptual level. However, on the technical level, multiple adjustments are required. In Thummerer, Mikelsons, and Kircher (2021), *neural FMUs* were introduced as the combination of FMUs and ANNs. From the point of view of PeN-ODEs, a neural FMU can be constructed by replacing the ODE by a ME-FMU. In addition, neural FMUs can also be constructed based on CS-FMUs (Thummerer, Mikelsons, and Kircher 2021), but are not further investigated in this contribution.

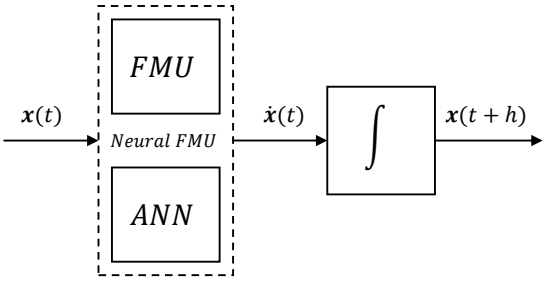

Figure 4. A neural FMU (ME), that consists of a ME-FMU, an ANN and the ODE solver. Based on a given state $\boldsymbol{x}(t)$, the neural FMU computes a state derivative $\dot{\boldsymbol{x}}(t)$, which is integrated into the next state of the system $\boldsymbol{x}(t+h)$. No assumptions on how the FMU is connected to the ANN are made. In addition, multiple FMUs and ANNs can be involved.

In Thummerer, Stoljar, and Mikelsons (2022), multiple training strategies for neural FMUs are investigated. In the following, we briefly explain the methodological foundation that we use as a starting point for the presented extensions. The following decisions are made:

Gates As a starting point for building a hybrid model architecture, the gates topology is used (Thummerer, Stoljar, and Mikelsons 2022), based on a parallel-serial layout (Thummerer and Mikelsons 2025). Here, two gates are applied that control how much of the (original) FMU motor dynamics and how much of the (newly) learned ANN motor dynamics contribute to the resulting hybrid model motor dynamics. The gates are parameterizable and are trained together with other optimization parameters, choosing proper initial parameters is part of hyperparameter optimization.

Pre- and Post-Processing To normalize the values between machine learning and the domain of the physical model, pre- and post-processing is applied online (Thummerer, Stoljar, and Mikelsons 2022). Here, pre- and post-processing corresponds to the introduction of two additional linear operations that shift and scale signals at the domain boundaries (between the physical and machine learning model and vice versa) in order to prevent suboptimal operation ranges for the nonlinearities inside the ANN. For example, if hyperbolic tangent activation is applied (as we used in the later experiment), small values close to zero lead to operation in (almost) linear ranges of this function and therefore neglection of the intended nonlinearity, whereas large values drastically saturate to 1 or -1, leading to a vanishing gradient. Both issues are major obstacles when training (nonlinear) hybrid models.

Interfaces In Thummerer and Mikelsons (2025), it is investigated how models can be combined, and learnable connections between physical and machine learning models are proposed. However, it is shown that this is not always efficient if only a limited amount of data is available. This motivates a clever selection of interface signals for the ANN. Because only little is known about the not modeled physical effect, it is decided to use the following signals as inputs for the ANN:

- The **system state** is used, to allow the ANN to learn its own representation of the right-hand side of the ODE. This corresponds to a parallel-like model architecture.
- The **system inputs**, to allow the learned right-hand side to further depend on the current input.
- The **state derivative** calculated by the FMU, which allows to *correct* the dynamical representation of the system calculated by the physical model. This corresponds to a serial-like model architecture.
- The time is explicitly not used as input to the ANN, because a time-invariant system is assumed, so without explicit time dependency.

The presented decisions result in the architecture visualized in Fig. 5, which is used as the foundation for the considered application.

2 Method

Although the architecture introduced in Fig. 5 was able to produce good results in a very similar form for different application scenarios from mechanical engineering (Thummerer, Stoljar, and Mikelsons 2022; Thummerer,

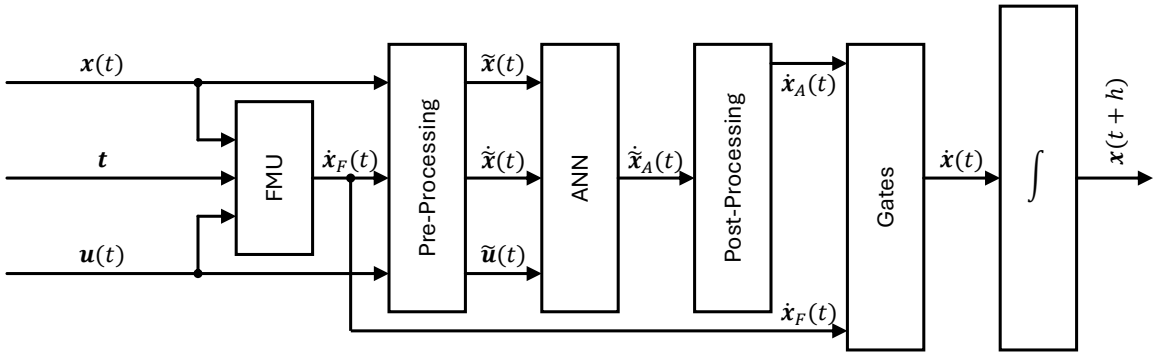

Figure 5. The starting point for derivation of the neural FMU (ME) used in this paper. It features pre- and post-processing layers as well as gates. First, the state $\boldsymbol{x}$, time t, and inputs $\boldsymbol{u}$ are fed into the FMU. The FMU computes a state derivative $\dot{\boldsymbol{x}}_F$, which is pre-processed together with the original state and input values. After pre-processing, the considered values are passed to an ANN, that computes its own state derivative vector $\tilde{\dot{\boldsymbol{x}}}_A$, which is further post-processed to $\dot{\boldsymbol{x}}_A$. The gates layer then controls how much of the ANN dynamics and the FMU dynamics contribute to the overall dynamics of the hybrid model $\dot{\boldsymbol{x}}$, which is finally integrated to the next system state $\boldsymbol{x}(t+h)$ by performing a step with size h in time.

Kolesnikov, et al. 2023) and medical technology (Thummerer, Tintenherr, and Mikelsons 2021), further methodological adjustments must be made for the particularly challenging application case at hand. These are introduced and discussed below.

2.1 Architecture: Learning Input Delay

The measurement data used is recorded at high frequency ($10^4\,Hz$). It is worth mentioning that this sampling frequency is not an excessive choice and is required to capture significant information, for example PWM signals to drive the electric motor. Due to the combination of several sensor signals in the data logger or errors in time synchronization between these sensors, time offsets may occur between the measurement signal of different sensors (see Fig. 6). In general, for small deviations, it is difficult to distinguish these temporal deviations from system dynamics. If the cause cannot be clearly identified (and explained), appropriate correction options can be provided in the architecture for both possible causes. This is achieved by integrating a block for introducing a constant-time offset for the input signal in addition to a neural network for correcting the system dynamics. Mathematically, this is defined simply as follows:

$$\boldsymbol{u}_{out}(\boldsymbol{u}(t),t) := \boldsymbol{u}(t+\Delta t), \tag{1}$$

where Δt is a trainable parameter and $\boldsymbol{u}_{out}$ is the output signal of the delay correction block. Since the time-offset block is parameterized and its parameter Δt can be differentiated, this parameter can be optimized together with the parameters of the neural network in a gradient-based training.

2.2 Architecture: Bypass Dynamics

For the presented application, we decide to only manipulate the *mechanical* part of the system state vector, so the mechanical subsystem, respectively. This is justified

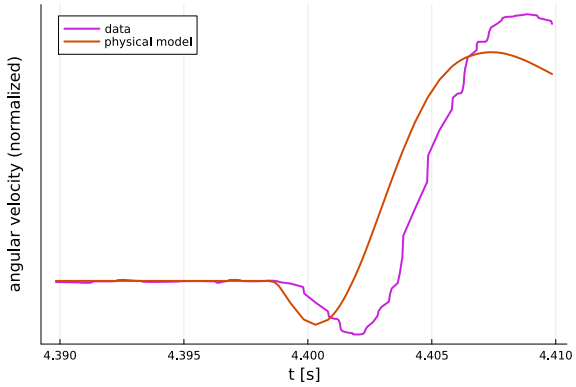

Figure 6. The plot shows the angular velocity data, as well as the output of the simulation model. There is a time shift in between that could be compensated by adding an offset in time to the input signals.

by the fact that the modeling and parameterization of the electrical subsystem are assumed to be accurate, while the mechanical subsystem has a higher level of uncertainty. As a consequence, the electrical part of the state derivative is forwarded to the ODE solver without any manipulation, which corresponds to leaving the right-hand side of the electrical subsystem untouched. To further preserve the second-order ODE structure for the mechanical subsystem, we only allow manipulation of the rotor acceleration (and not the rotor speed), which we achieve by forwarding the unmodified rotor speed to the ODE integrator as well.

Based on the preliminary work summarized in Fig. 5, the proposed input offset correction block and signal forwarding, a model architecture as in Fig. 7 can be deployed to address the presented use case.

2.3 Data Handling

As shown in Fig. 8, real measurement data is often very heterogeneous in terms of information content. For a

DOI
10.3384/ecp218575

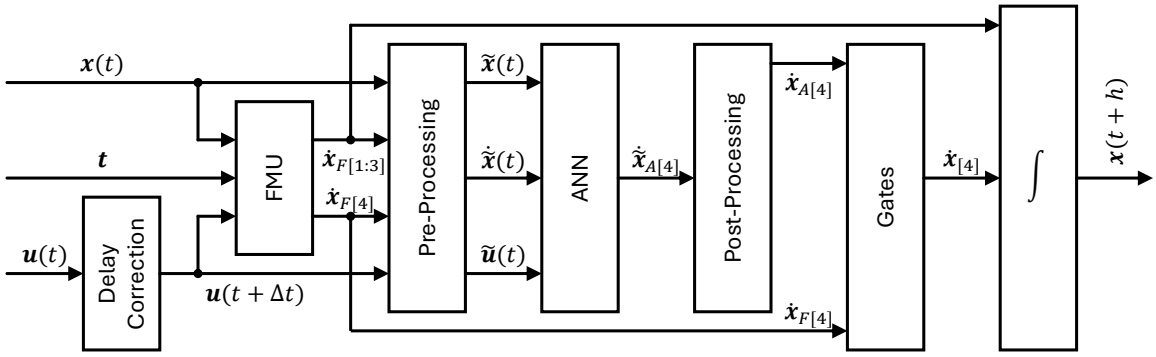

Figure 7. The architecture designed for the presented use case, based on the introduced architecture in Fig. 5 and extended by input delay correction. Not all state derivatives $\dot{x}$ are computed by the ANN and post-processing, only $\dot{x}_{A[4]}$ (the motor rotor acceleration). The others $\dot{x}_{F[1:3]}$ are cached and forwarded directly to the ODE solver.

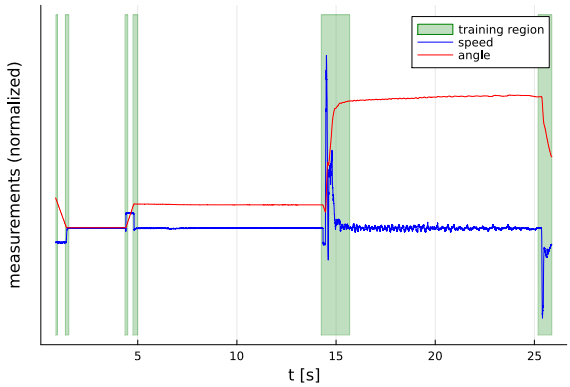

Figure 8. The training data. As data is *heterogneous* (varying information content), only the green regions (that offer information *variance*) are used for training.

large part of the time axis, the system is at rest (or almost at rest). This is due to a delay between the start and end of the recording and the execution of the actual test. In this application, we have decided to thin out the data manually, but we are working in parallel on robust approaches to automate this step. It is important to note that this step is more complex than, for example, filtering only for derivatives and has a significant influence on the quality of the results with such a small amount of training data.

3 Experiment

The presented method is validated in the challenging use case of the IPB introduced in Sec. 1.1. The experiment involves the training process, followed by the comparison of results against measurements of the real physical system (ground-truth data, baseline). To determine good hyperparameters, a hyperparameter optimization is performed.

3.1 Data

For intellectual property reasons, all data plots and error measures within this publication are normalized. The data used for training, validation, and testing was generated under typical development conditions:

- All measurements were performed on the real IPB hardware (prototype) and
- an ECU software measurement tool was used to measure the sensor and actor control quantities, so no additional sensor are required/used.

To show the ability of the presented method to provide good results on a very small data set under conditions of real development in the field, only three measurements with durations of $\approx 12-25\,s$ are used. One trajectory is used for training (gradient determination for optimization), one for validation (evaluating the success of hyperparameter optimization), and one for testing (completely unknown to the model and optimization processes).

3.2 Training

Because the experiment includes a real industrial simulation model that is part of active development, this model cannot be shared. However, the experiment setup is described in the following, to allow for reproduction within similar use cases:

Hyperparameter Optimization For hyperparameter optimization, the optimizer *Hyperband* (Li et al. 2018) is applied with parameters $R = 81$ and $\eta = 3$. The optimizer is paired with a simple random sampler as the inner sampling algorithm[3]. As resource for *Hyperband*, a time span is used for the processed training data, which is limited to a maximum of $100\,s$.

Optimizer For the actual training, the optimizer *Adam* (Kingma 2014) is applied, with step size $\eta \in [10^{-6}, 10^{-4}]$ (best: 3^{-6}), $\beta_1 \in \{0.99, 0.9\}$ (best: 0.99) and $\beta_2 \in \{0.9999, 0.999, 0.99\}$ (best: 0.9999). Stochastic mini-batching is applied as training strategy, with a batch element length of $[0.01\,s, 0.1\,s]$ (best: $0.02\,s$).

[3]Implementation available: `https://github.com/ThummeTo/DistributedHyperOpt.jl`.

Loss function The loss function consists of two separated errors, one for the motor angle and one for the speed. For error determination, mean squared error (MSE) as well as mean absolute error (MAE) (best) are used. To obtain a scalar loss, the loss terms are weighted by a percentage between 0 % and 100 % (best: 70 % speed, 30 % angle) and added together. This allows for loss functions that involve only the speed, only the angle, or a mixture of both.

ANN layout For the ANN, different widths $\{8, 16, 32\}$ (best: 8) and depths $\{2, 3, 4\}$ (best: 4) are tested. Only the *tanh* activation function is applied for all layers.

Gates The gates for the influence of the ANN are trained together with the ANN parameters, but feature different initial openings between $[0\,\%, 50\,\%]$ (best: 30 %).

4 Results & Discussion

After training, we find the following results that we want to discuss in detail in this section. It is important to note that training, validation, and testing are performed *open-loop*. This means that control signals are recorded from the closed-loop (controlled) system, but for training and validation, the motor is operated based on the pre-recorded control signals without involving the controller (remove feedback). This allows us to evaluate the actual accuracy of the hybrid model without controller influences positively distorting the results.

Training, validation, and test data vary w.r.t. different aspects, for example the maximum applied load or the maximum rotor velocity vary by at least 10 % between the individual datasets.

We deliberately decided against baselining against a pure ML model because under the given circumstances (very little data, very few training resources) no presentable ML model can be trained, even if more advanced approaches such as neural ODE are used for this purpose. Please note that training data only captures a part of the state space of the system, and therefore a pure ML approach is by design not able to recover the system behavior beyond the limited training data.

4.1 Validation

The best hyperparameter run values can be found in Sec. 3.2, leads to the following results for the motor angle (s. Fig. 9). The determined value for the time offset (input delay) is $\approx -1.995 \times 10^{-5} s$. Because the motor is very sensitive with respect to the inputs, this small offset has a significant positive influence on the simulation results.

Qualitatively, it can be stated that the neural FMU trajectory does not have any significant, obvious deviation compared to the data trajectory. This does not apply to the original FMU that has a large deviation, especially near the end of the trajectory. This can be seen in Fig. 10 starting at around $18.3\,s$, which shows the deviation in more detail.

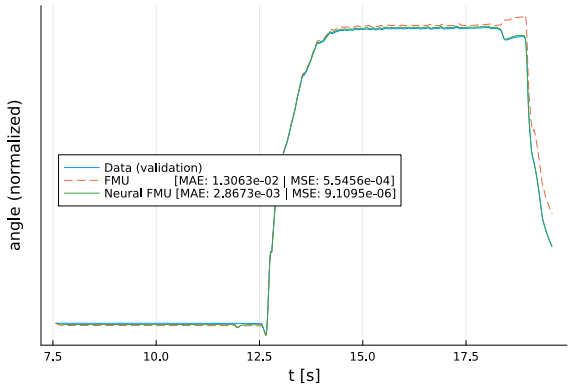

Figure 9. The motor angle on the validation data on the full trajectory. Loss values are given in square brackets.

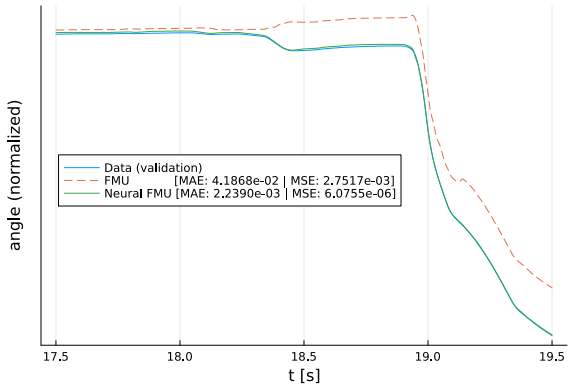

Figure 10. The motor angle on the validation data, zoomed in to the time interval from $17.5\,s$ to $19.5\,s$. The loss values for this interval are given in square brackets.

On a quantitative level, the deviation for the motor angle for the neural FMU is ≈ 4.6 times smaller compared to the FMU on MAE, and even around 61 times on MSE. At this point, we need to highlight that these factors are very impressive from an engineering point of view and deliver massive added value for the (hybrid) simulation model.

As expected from the good results for the motor angle, the neural FMU also provides a qualitatively good fit on the motor speed, see Fig. 11.

Again, we investigate a part of the speed trajectory in more detail. What is particularly striking are the deviating spikes of the FMU, which can be investigated in more detail in Fig. 12. The spikes result from the fact that the model is very sensitive with respect to the control signals and small deviations between the control signal and the position of the rotor lead to unstable behavior in the open-loop control system. Although the original simulation model (FMU) consistently shows these oscillations that even lead to overshoots in the event of rapid changes in speed (acceleration), the neural FMU is able to compensate for the unintended oscillations and provides a nice

DOI
10.3384/ecp218575

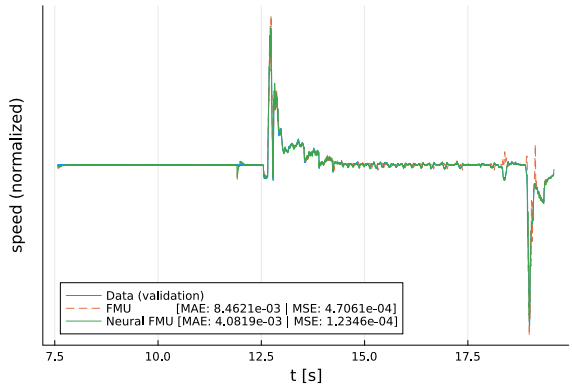

Figure 11. The motor speed on the validation data on the full trajectory. Loss values are given in square brackets.

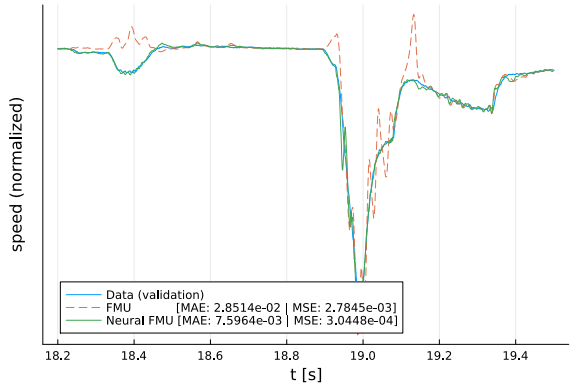

Figure 12. The motor speed on validation data, highlighting the interval between 18.2 s and 19.5 s. Loss values are given in square brackets.

and smooth fit without significant deviations.

4.2 Testing

For proper validation of the hybrid model, the neural FMU is further investigated under testing data, that is completely unknown from training as well as hyperparameter optimization. As for validation data, the neural FMU qualitatively performs very well on testing data (s. Fig. 13), and is even able to provide a little better increase in precision of the predicted rotor angle of factor ≈ 5.4 on MAE or factor ≈ 74 on MSE, respectively.

In contrast to the validation, we want to inspect a segment with raising edge for the motor angle in more detail, see Fig. 14. Within this segment, the original model (FMU) already performs very well, and we want to investigate if the neural FMU is able to retain the parts of the original model that already perform well. Even if the original model has high accuracy in the investigated time span, the neural FMU is able to outperform this performance by a factor of around 2.2 (MAE).

Finally, we also investigate the motor speed on testing data in the following. When investigating the rotor speed on testing data (s. 15), we find quantitative improvements similar to the validation scenario of factor > 2 (MAE).

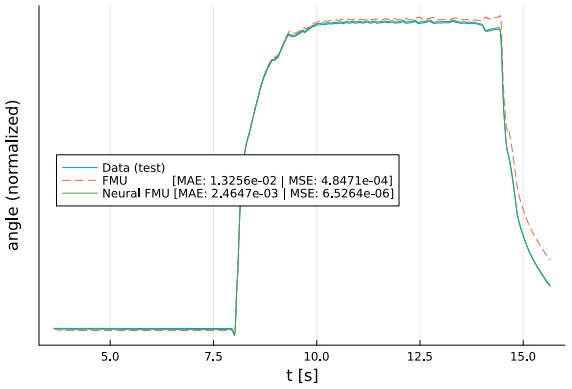

Figure 13. The motor angle on test data on the full trajectory.

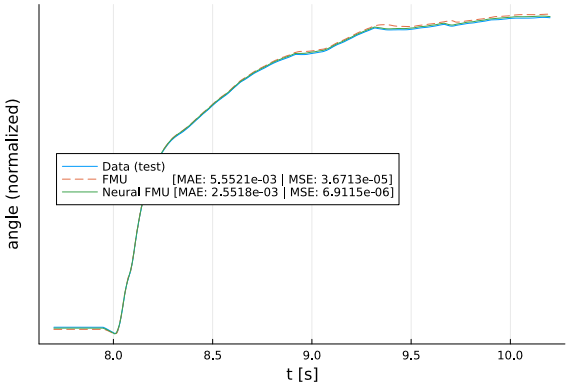

Figure 14. The motor angle on the test data, between 7.8 s and 10.2 s. Loss values are given in square brackets.

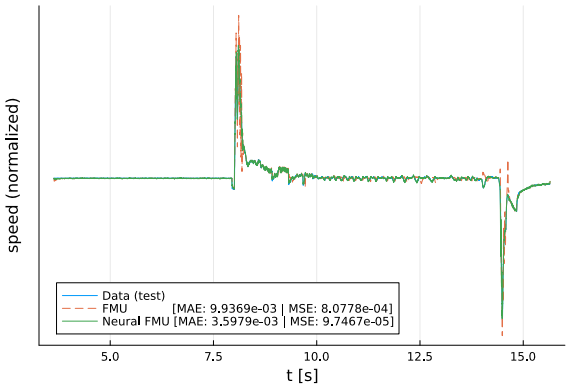

Figure 15. The motor speed on the test data on the full trajectory. Loss values are given in square brackets.

Again, more significantly, the qualitative fit has drastically improved in terms of reduced (wrong) oscillations in the open-loop simulated motor, see Fig. 16.

5 Conclusion

We can summarize that hybrid modeling has the potential to be used in challenging real industrial applications. In this particular use case of the automotive braking system, we were able to increase the accuracy by multiple factors, even though the physical model was based on a

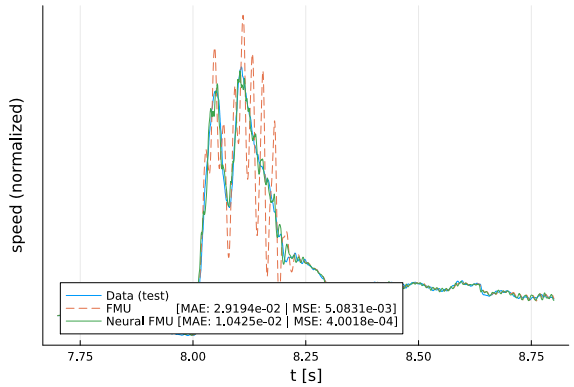

Figure 16. The motor speed on test data, highlighting the interval between 7.75 *s* and 8.75 *s*. Loss values are given in square brackets.

simulation model from real state-of-the-art engineering, built and parameterized by experts. Nevertheless, the successful results presented are still based on well-founded decisions by the development engineer, e.g. the choice of a model architecture that meets the requirements of the use case. We also see this as the task of the modern simulation engineer in the future. Although such decisions can in principle also be optimized by ML approaches, the approach presented here impresses with its very low computing power requirements compared to large-scale optimizations. In this use case, even the in general expensive hyperparameter optimization was executed on a standard desktop CPU with a runtime of only around 48 hours.

For the considered use case, we used data from the (partly) available real prototype to improve an existing simulation model. This approach is of course not applicable, e.g. during early phases of the development, when a real demonstrator is not available. However, the proposed method can also be applied based on synthetic data from another simulation model, e.g. to create a hybrid surrogate model with improved properties (e.g. computational performance or memory requirements) compared to the original model.

Future work will focus, for example, on training and testing based on a larger data set. In this application, in particular, a relatively small data set (one maneuver each for training, testing, and validation) was used to demonstrate the applicability of the method even with only small amounts of data, which is common for many industrial application fields — especially during protoyping. Further, we look at ways to further automate parts of development (e.g. training and batching strategies) to further reduce the development time of such hybrid models.

Abbreviations

ANN arifical neural network

CS co-simulation

FMI Functional Mock-up Interface

FMU functional mock-up unit

IPB Integrated Power Brake

MAE mean absolute error

ME model-exchange

MSE mean squared error

ODE ordinary differential equation

PeN physics-enhanced neural

PWM pulse-width modulation

UODE universal ordinary differential equation

Funding

This research was partially funded by the following sources:

- ITEA3-Project UPSIM (Unleash Potentials in Simulation) No. 19006. For more information, see: `https://www.upsim-project.eu/` (accessed on 12.12.2024).
- ITEA4-Project OpenSCALING (Open standards for SCALable virtual engineerING and operation) No. 22013. For more information, see: `https://openscaling.org/` (accessed on 12.12.2024).

The authors thank all those involved for their support in making our research possible.

Availability of Software, Data and Model

The software used *FMIFlux.jl*[4] is open source on GitHub. However, the simulation model and the data presented are not available to the public due to intellectual property. All data shown was normalized. However, *FMIFlux.jl* offers demo applications that are similar to the workflow presented, and application-specific features, such as learning of time shifts, are available as part of the open source repository.

Author Contributions

Conceptualization, T.T., J.F., S.D., M.L.; methodology, T.T., M.L.; software, T.T.; validation, T.T., S.D.; first-principle model: S.D.; writing, T.T., S.D., J.F., M.L.; visualization, T.T., S.D.; All authors have read and agreed to the published version of the manuscript.

[4] `https://github.com/ThummeTo/FMIFlux.jl`

References

Bertsch, Christian (2022). *FMI 3.0 – The next generation exchange format for system simulation beyond tool borders*. Tech. rep. Robert Bosch GmbH. URL: https://www.bosch.com/stories/fmi-3-0-the-next-generation-exchange-format-for-system-simulation-beyond-tool-borders/.

Chen, Ricky T. Q. et al. (2018). "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Chen, Ricky TQ, Brandon Amos, and Maximilian Nickel (2020). "Learning neural event functions for ordinary differential equations". In: *arXiv preprint arXiv:2011.03902*.

Hofmann, Andreas and Lars Mikelsons (2025). "Towards Integration of PeN-ODEs in a Modelica-based workflow". In: *16th International Modelica & FMI Conference, Lucerne, Switzerland, Sep 8-10, 2025*. (submitted).

Kingma, Diederik P (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Li, Lisha et al. (2018). "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *Journal of Machine Learning Research* 18.185, pp. 1–52.

Modelica Association (2020-12). *Functional Mock-up Interface for Model Exchange and Co-Simulation. Document version: 2.0.2*. Tech. rep. Linköping: Modelica Association. URL: https://github.com/modelica/fmi-standard/releases/download/v2.0.2/FMI-Specification-2.0.2.pdf.

Modelica Association (2023-07). *Functional Mock-up Interface Specification. Version 3.0.1*. Tech. rep. Modelica Association. URL: https://fmi-standard.org/docs/3.0.1/.

Rackauckas, Christopher et al. (2021). *Universal Differential Equations for Scientific Machine Learning*. arXiv: 2001.04385 [cs.LG].

Ramadhan, Ali et al. (2023). *Capturing missing physics in climate model parameterizations using neural differential equations*. arXiv: 2010.12559 [physics.ao-ph]. URL: https://arxiv.org/abs/2010.12559.

Robert Bosch GmbH, ed. (2024). *Kraftfahrtechnisches Taschenbuch*. Wiesbaden: Springer Vieweg. ISBN: 978-3-658-44233-0. DOI: 10.1007/978-3-658-44233-0. URL: https://link.springer.com/book/10.1007/978-3-658-44233-0.

Robert Bosch GmbH (2025). *Bosch Integrated Power Brake*. https://www.bosch-mobility.com/en/solutions/driving-safety/integrated-power-brake/ [Accessed: 04-03-2025].

Sorourifar, Farshud et al. (2023). "Physics-enhanced neural ordinary differential equations: application to industrial chemical reaction systems". In: *Industrial & Engineering Chemistry Research* 62.38, pp. 15563–15577.

Tac, Vahidullah, Francisco Sahli Costabal, and Adrian B. Tepole (2022). "Data-driven tissue mechanics with polyconvex neural ordinary differential equations". In: *Computer Methods in Applied Mechanics and Engineering* 398, p. 115248. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2022.115248. URL: https://www.sciencedirect.com/science/article/pii/S0045782522003838.

Thummerer, Tobias, Artem Kolesnikov, et al. (2023). "Paving the way for hybrid twins using neural functional mock-up units". In: *Modelica Conferences*, pp. 141–150.

Thummerer, Tobias and Lars Mikelsons (2025). *Learnable & Interpretable Model Combination in Dynamical Systems Modeling*. arXiv: 2406.08093 [cs.LG]. URL: https://arxiv.org/abs/2406.08093.

Thummerer, Tobias, Lars Mikelsons, and Josef Kircher (2021). "NeuralFMU: towards structural integration of FMUs into neural networks". In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Ed. by Martin Sjölund et al. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181297.

Thummerer, Tobias, Johannes Stoljar, and Lars Mikelsons (2022). "NeuralFMU: Presenting a Workflow for Integrating Hybrid NeuralODEs into Real-World Applications". In: *Electronics* 11.19. ISSN: 2079-9292. DOI: 10.3390/electronics11193202. URL: https://www.mdpi.com/2079-9292/11/19/3202.

Thummerer, Tobias, Johannes Tintenherr, and Lars Mikelsons (2021-11). "Hybrid modeling of the human cardiovascular system using NeuralFMUs". In: *Journal of Physics: Conference Series* 2090.1, p. 012155. DOI: 10.1088/1742-6596/2090/1/012155. URL: https://dx.doi.org/10.1088/1742-6596/2090/1/012155.

Xie, Xiang, Ajith Kumar Parlikad, and Ramprakash Srinivasan Puri (2019). "A Neural Ordinary Differential Equations Based Approach for Demand Forecasting within Power Grid Digital Twins". In: *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6. DOI: 10.1109/SmartGridComm.2019.8909789.