

Status of the SMArtInt Library: Simple Modelica Artificial Intelligence Interface

T. Hanke¹ J. Brunnemann¹ R. Flesch¹ J. Eiden¹

¹XRG Simulation GmbH, Hamburg, Germany, {hanke,brunnemann,flesch,eiden}@xrg-simulation.de

Abstract

This paper presents the current status of the open-source Modelica library SMArtInt (Simple Modelica Artificial Intelligence Interface) (SMArtInt 2025), which offers users a straightforward and efficient approach of integrating artificial intelligence via neural networks directly into Modelica models. We provide a detailed overview of the library's features, area of application, and development process. The primary focus is on the diverse use cases of SMArtInt. The new version 0.5.1 brings an exciting feature with the support of the ONNX (ONNX 2025) format. ONNX (Open Neural Network Exchange) is an open, cross-platform format for the representation and exchange of deep learning models. This means that in addition to TensorFlow (TensorFlow 2025) models (TFLite), many other models, such as PyTorch (PyTorch 2025) can now also run in SMArtInt via the ONNX format.

Keywords: AI, SMArtInt library, Artificial Intelligence, Neural Network, TensorFlowLite, ONNX, BNODE

1 Introduction

1.1 Context of Paper

In recent years, machine learning techniques have successfully established themselves in various areas of science and technology, often summarized under the term “artificial intelligence” (AI). These methods have proven to be particularly effective in processing large amounts of data and enable an automatic recognition of correlations, patterns and classifications within the data (Lam et al. 2023; Kong et al. 2025).

Time series models, in which AI techniques such as neural networks can be used to predict and analyze temporal data, are more and more integrated into physical simulations. Such hybrid systems (see figure 1) combine data-driven insights with traditional physical models (Matei 2018; Kurz et al. 2022; Rudolph, Kurz, and Rakitsch 2024; Quarteroni, Gervasio, and Regazzoni 2025). This integration enables more accurate and flexible system modeling, especially for complex, non-linear or unknown processes, see for instance (Von Krannichfeldt, Orehounig, and Fink 2025), (Gijon et al. 2025) or (Markovic et al. 2018) as example with Modelica context. It also improves the ability to simulate and optimize dynamic systems, see e.g. (Sin-

graval, Suykens, and Geyer 2018), (Aka, Brunnemann, Freund, et al. 2023).

In the Modelica context of state space models, described by a system of ordinary differential equations (ODE), the hybrid model architecture (figure 1) leads to the concept of a Neural ODE (Chen et al. 2018): time derivatives of the states are computed from a combination of equations and neural networks.

There are different possibilities for integrating neural networks into physical system simulations. On the one hand, software manufacturers may develop tool-specific AI interfaces for integrating neural networks into their Modelica simulation environments. On the other hand, there are already different tool-independent, open-source approaches available in the Modelica community, such as the NeuralNetwork (NeuralNetwork 2025) library, which maps neural layers using Modelica blocks, whereby the associated weights and biases are obtained from a read-in matrix, or the SMArtInt library, which implements neural networks directly via external C-functions.

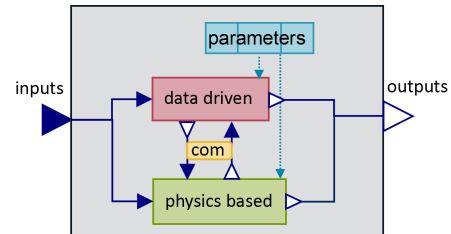


Figure 1. Concept of hybrid modeling: data driven and physics based model parts integrated into a single system model. Internal data exchange via interface variables (com).

The aim of SMArtInt is to create a library that allows integration of trained neural networks directly into Modelica very much in the fashion of input-output blocks. The library is explicitly intended to enable Modelica users to create hybrid models in a simple way while staying inside their preferred Modelica development tool, hence strengthening Modelica's position as multi physics engineering modeling language. Additionally, usage of external C-functions makes the Modelica equation system independent of the size of inferred neural networks, which may in practice contain several thousand parameters. Moreover this approach allows for exchanging the inferred neural network without re-translating the Modelica model, as

long as the input-output structure stays the same.

SMARTInt was developed within a German research collaboration (DIZPROVI 2021-2024)¹ (Brunnemann, Flesch, and Hanke 2024) under the name *SMARTInt*. The first official release of version 0.1.0 dates back to March 2023.

1.2 Outline of Paper

This paper is structured as follows: Section 2 provides the scope and structure of the library as well as an insight into the functionality of SMARTInt. Section 3 discusses the innovations introduced in version 0.5.1, with a particular focus on the support of the ONNX format. Section 4 presents three different application cases for the library: the integration of a simple feed-forward neural network within a discretized pipe, replacing a heat transfer correlation; a hybrid superheater model trained on measurement data from a coal-fired power plant, developed as part of the DIZPROVI project; and a Balanced Neural ODE used to create a surrogate model of an entire steam cycle. Section 5 presents the SMARTInt funding concept and outlines the additional content of the SMARTInt+ commercial version. Finally, a summary and outlook is given in section 6.

2 Overview of the Library SMARTInt

2.1 Scope of Library

SMARTInt enables the integration of trained artificial intelligence (AI) models into Modelica tools. SMARTInt currently supports TensorFlow models (exported in TFLite format), as well as ONNX models. LiteRT (formerly TensorFlow Lite) (Shuangfeng 2020) was initially selected as the core inference runtime due to its optimizations for embedded systems and edge computing environments, offering a small binary size, high execution efficiency, and support for low-latency inference, while ONNX support was added in the latest version (see section 3) to improve SMARTInt's framework compatibility. For integration, users need to specify the path to the AI model, as well as the input and output dimensions of the neural network within the interface block. The network structure, including layers and neurons, does not need to be defined, allowing neural networks to be easily exchanged as long as the input/output structure remains unchanged. This flexibility enables repeated training with different datasets, hyperparameters, or internal architectures (e.g., layer configurations) without requiring modifications to the Modelica model in which the neural network is used. Inputs and outputs are accessible in Modelica as arrays via the interface blocks, ensuring a simple and flexible integration process.

In hybrid modeling, different levels of hybridization can be distinguished (see figure 2 for illustration). On the top level, entire models can be replaced by data driven surrogates. On the system level, individual submodels and replaceable models can be substituted to implement

physical correlations directly from data. Additionally AI models can be used to model detailed physical effects that are usually not accessible at the level of system simulation: thanks to acausality of Modelica a simplified physics based system model may be inverted in order to 'subtract' its result from measurement data. For example the effect of heat exchanger fouling, represented in a simple manner by a model parameter in the original model, may be extracted in more detail as a time series by imposing the measured heat flow and extracting the effective time depending fouling as output of the inverted model. The generated time series may then be used in order to train a detailed data driven fouling model.

The approaches described above were investigated and implemented as part of the DIZPROVI research project.

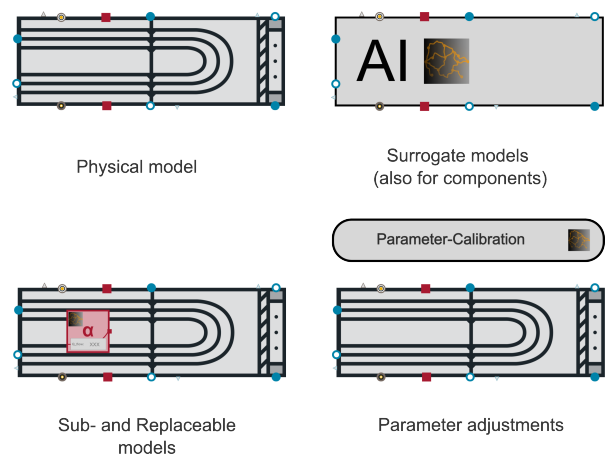


Figure 2. Different levels of hybridization

In the construction of hybrid models care has to be taken, when combining AI models based on measurements of the real system with an abstract physics based system model, subject to several simplifying model assumptions. In practice a thorough investigation is recommended in order to determine the 'best' level of hybridization as well as interface variables between physics and data based model parts (figure 1). For example it may happen that certain spatially resolved sensor positions are not available in the 1D system model or vice versa certain system variables are not accessible for measurement. Moreover validity of data based model parts is limited to the input range of measurement data. It is also important to realize that AI models usually expect a causal input-output structure, while physics based Modelica models are equation based and acausal. A more detailed discussion of these issues can be found in the DIZPROVI project report (Brunnemann, Flesch, and Hanke 2024).

2.2 Structure of Library

The SMARTInt library is designed to be as simple and user-friendly as possible. Table 1 provides an overview of its top-level structure. The library consists of four packages and a *UsersGuide*, which offers essential information for getting started and effectively using the

¹ funded by the German Federal Ministry of Education and Research under reference number FKZ 03WIR0105E

library.

The *Blocks* package contains interface blocks (as shown in Figure 3) necessary for integrating neural networks into Modelica models. The *Tester* package includes various examples demonstrating the correct usage of these interfaces and their integration into different models.

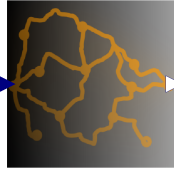








Figure 3. SMARTInt's interface block for neural networks

For advanced users, the *BaseClasses* package provides direct access to the core functionalities of the library. Finally, the *Internal* package contains the underlying external C functions that build the base of SMARTInt.

Table 1. SMARTInt library structure

	SMARTInt	
	UsersGuide	Basic information on usage and licenses
	Tester	Examples to introduce new users to the library
	BaseClasses	Basic models/templates for integrating different types of neural networks in Modelica
	Blocks	Blocks to integrate different types of neural networks into Modelica models (based on the <i>BaseClasses</i>)
	Internal	Internal functions for integrating/interacting with the neural networks

2.2.1 Technical realization

A large part of the functionality of SMARTInt is located outside of Modelica in external C functions, which are integrated into SMARTInt as dynamic libraries (.dll and .so). SMARTInt supports three different neural network types, each of which has its own interface block:

1. Feed-forward neural networks (FFNNs): Output depends only on the current input.
2. Recurrent neural networks (RNNs): Use previous inputs (time series) to compute the current output.
3. Stateful recurrent neural networks (sRNNs): Preserve their hidden state across inference steps to capture long-term dependencies.

SMARTInt operates as a real-time application, processing only the input variables provided by the solver at the current time step. This creates particular challenges regarding the input management of RNNs. This is addressed by the input management system, which supports RNNs as well as stateful RNNs in combination with solvers that utilize variable time step sizes, ensuring seamless integration within Modelica simulations.

Since RNNs must operate at the same temporal resolution as their training data, SMARTInt uses an external C-based container to store input values. If the time step of the Modelica solver exceeds the trained step size of the RNN, the required values are interpolated to maintain consistency. In addition, this input storage mechanism ensures robustness by allowing retrieval of the last accepted values in cases where solver steps are rejected.

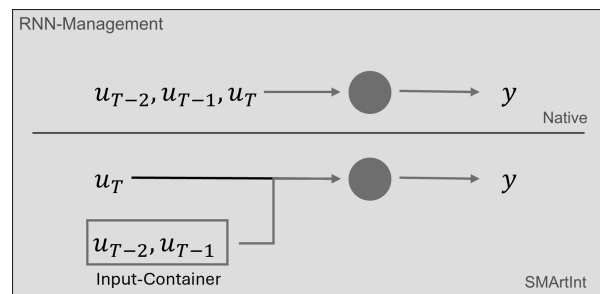


Figure 4. RNN input management

Stateful RNNs are not natively supported in TFLite and have limited functionality in ONNX (ONNX 2025), where network states can only be stored internally for batched inputs. Stateful RNNs must be explicitly restructured by representing layer states as outputs and defining corresponding state inputs. These states are managed externally by the SMARTInt input management system in a special state container, which enables seamless integration while maintaining time dependencies.

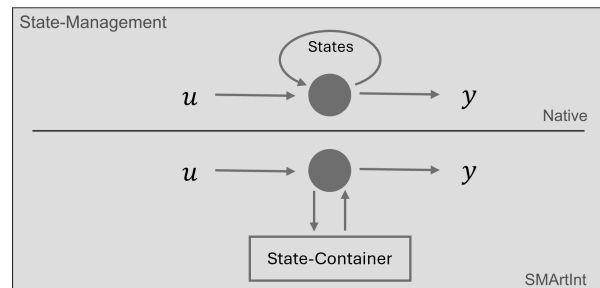


Figure 5. State input management

3 Latest Features of SMARTInt

Version 0.5.1 introduces support for the ONNX format. The Open Neural Network Exchange (ONNX 2025) is an open-source, cross-platform format designed for the exchange of deep learning models. It serves as an interface between various machine learning frameworks, en-

abling model conversion across different formats. The integration of ONNX extends SMartInt's compatibility beyond TensorFlow (TensorFlow 2025) to include frameworks such as PyTorch (PyTorch 2025), NeoML (NeoML 2025), and many others. A comprehensive list of supported frameworks is available on the ONNX website². SMartInt automatically detects whether the specified model path corresponds to a TFLite or ONNX model and processes it accordingly. This ensures that Modelica users can work with a unified interface without requiring any additional configuration, simplifying the integration of neural networks into Modelica simulations.

4 Example use cases

In this chapter, three different examples are presented that illustrate different use cases of SMartInt. Both feed-forward neural networks and recurrent neural networks (RNNs) are integrated into Modelica models. Finally, a complex architecture of neural networks (BNODE) (Aka, Brunnemann, Eiden, et al. 2024) is presented, which can also be implemented with SMartInt in Modelica.

4.1 FFNN: Nusselt-Correlation

The first example model is a simple, trained feed-forward neural network integrated into Modelica as a TFLite model. It replaces the physical calculation of the Nusselt number with a neural network. The underlying correlation is given by the following relation:

$$Nu = f\left(Re, Pr, \frac{L}{d}\right) \quad (1)$$

Figure 6 illustrates the model and the exchangeable heat transfer model used in the pipe. The pipe is discretized into ten segments, resulting in inference being performed ten times per time step on the same instance of the neural network.

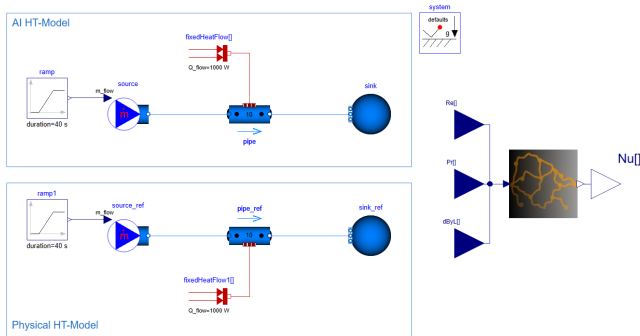


Figure 6. Left: Pipe model with AI Nusselt correlation and physical reference model. Right: Diagram of the build in data based heat transfer correlation.

The model is primarily based on the SMartInt example *Tester.PipeHeatTransferExample.TFLite.TestPipe_tflite*

²ONNX, supported frameworks and converters: <https://onnx.ai/supported-tools.html#buildModel>

and has been extended with a reference pipe and a dynamic mass flow input. During the simulation, a constant heat flow is applied to the pipe, while the mass flow is defined by a ramp function. Figure 7 shows the variation of

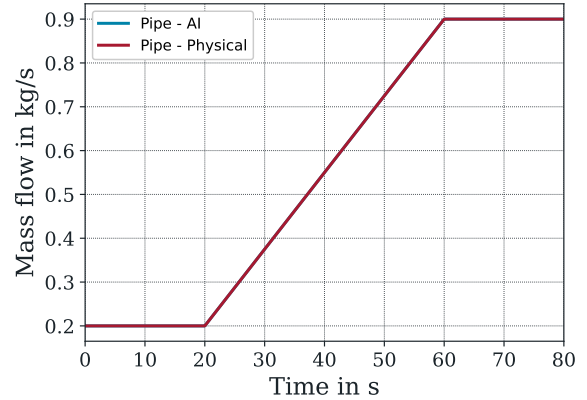


Figure 7. Mass flow in the pipe and the reference pipe

mass flow rates through the pipe. Initially, the mass flow is constant at 0.2 kg/s, and after 20 seconds, it is linearly increased to 0.9 kg/s over a period of 40 seconds. This dynamic change also impacts the heat transfer, as a higher mass flow with a constant heat flow results in a decrease in temperature within the pipe (as shown in Figure 8). When comparing the resulting temperatures of the pipe segments in Figure 8, no significant difference is observed between the physical and data-driven correlations.

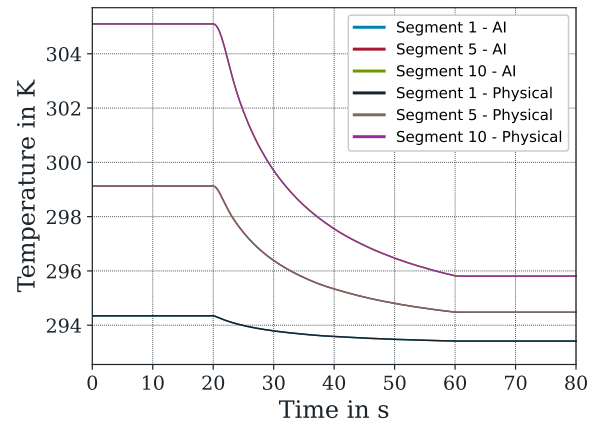


Figure 8. Temperature in the different pipe segments

4.2 sRNN: Superheater heat transfer model

The second example is based on a model (Brunnemann, Koltermann, et al. 2022) that comes from the DIZPROVI (Brunnemann, Flesch, and Hanke 2024) research project, in which SMartInt also was developed. The project's primary focus was the replacement of physically modeled components with data-driven neural networks as part of a digital power plant twin (illustrated in Figure 9). In this example, the physics-based heat transfer model for

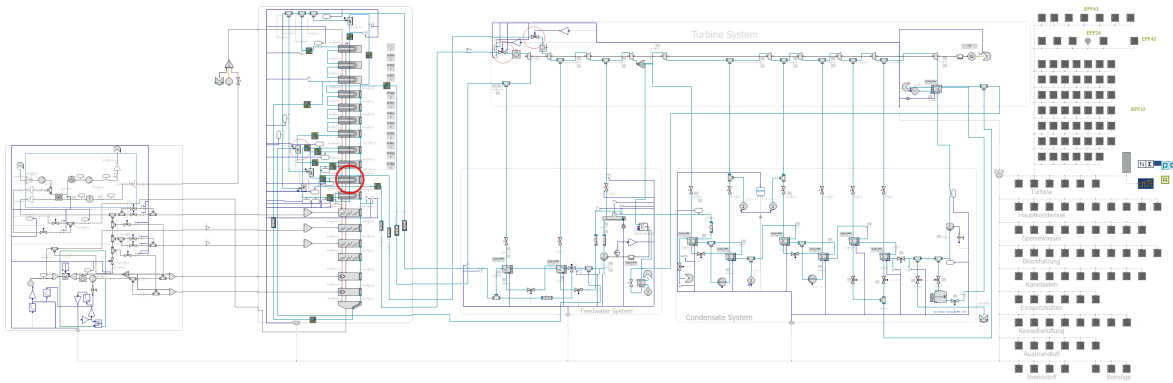


Figure 9. Digital twin of the investigated coal fire power plant, taken from (Brunnemann, Koltermann, et al. 2022)

one of the superheaters (SH4), which relies on energy balance, a Nusselt number correlation, and thermal radiation exchange, was replaced by a recurrent neural network (RNN). Using the inputs of flue gas and steam mass flow, as well as steam enthalpy and flue gas temperature at the superheater inlet, the neural network predicts the transferred heat flow as output. Figure 10 illustrates the predicted heat flow for both the physical and AI-based models within a validation scenario. The detailed control of the model presents a particular challenge for AI-based modeling, as the neural network must produce stable and reliable predictions even in regions outside the training distribution. Due to the mutual dependency between prediction and control, initial inference errors can cause the control logic to drive the system further into areas of low model confidence. This, in turn, degrades prediction quality and may ultimately lead to instability or simulation failure. This leads to two different integration approaches for the AI model:

1. Closed-loop integration, where the neural network is directly embedded into the superheater model, allowing interaction with the power plant's control system.
2. Open-loop integration, where the neural network operates independently of the overall model, serving primarily as a performance test for the trained neural network.

The AI-based heat transfer model was trained using data sets derived from measurement data. During data preparation, the spatial distribution of the sensors as well as the four-pass configuration of the evaporator area in the actual system were taken into account. In addition, the measurement data contains information on the degree of fouling and other relevant factors that influence the heat transfer surfaces. Therefore, in contrast to the physically based model, no further model calibration is required. Figure 10 shows that both the open-loop and closed-loop models align well with the calibrated physical model in the steady-state full-load range. In the partial-load range, however, the heat flow predicted by both AI models exceeds the physically computed heat flow. This discrepancy

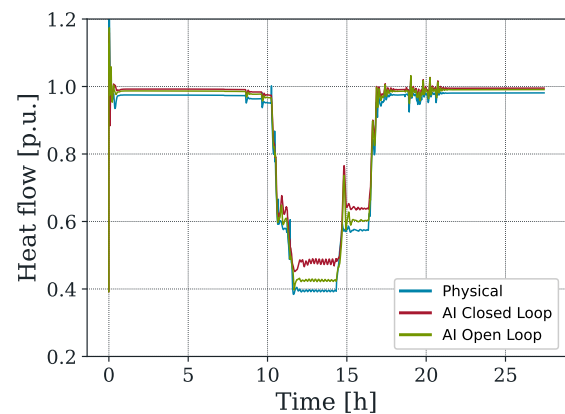


Figure 10. Heat flow SH4 (replaced heat transfer model)

does not necessarily indicate poor prediction accuracy of the AI model but may instead result from the calibration process of the physical model. For the physical model, the heat flow was adjusted via the fouling factor (CF) of the heat transfer surfaces under steady-state full-load conditions. Consequently, the actual heat flow in the partial-load range of SH4 may deviate from the physically computed and calibrated heat flow. Additionally, certain assumptions made during data processing may have led to lower-quality training data in the partial-load range, further affecting prediction accuracy. More details can be found in the DIZPROVI final report. The comparison of open-loop and closed-loop integration highlights the impact of control. The cooling regulation (CoolerSH3) stabilizes the steam outlet temperature of SH4, reducing thermal stress on the turbines. If heat transfer in SH4 increases, more cooling water is injected, lowering the outlet temperature and improving overall heat transfer. This effect is particularly evident in partial load operation. More detailed insight into the example and the research project can be found in the final report (Brunnemann, Flesch, and Hanke 2024).

4.3 BNODE: Steam-Cycle

The third example builds on the concept of a Balanced Neural ODE (BNODE), introduced by (Aka, Brunnemann, Eiden, et al. 2024). This approach employs a complex architecture composed of multiple interacting neural networks and can be used as a surrogate model for simulating the behavior of complex Modelica systems.

Balanced Neural ODEs combine Neural Ordinary Differential Equations (NODEs) (Chen et al. 2018) with Variational Autoencoders (VAEs) (Kingma and Welling 2019). Instead of approximating the original terms of the ODE directly by a neural network, the state space is mapped into a latent space and the neural network describes an ODE in the latent space. The latent states are then decoded back into the physical space to generate predictions that can be compared to the actual solutions of the ODE. By combining variational autoencoders with state space models, the generalization capability induced by the information bottleneck of VAEs can be used effectively. The information bottleneck created by the numerically defined latent space enforces a compressed, abstract representation of the data and promotes both data locality and the orthogonality of latent dimensions.

The term Balanced Neural ODE (BNODE) refers to the trade-off between state space compression and reconstruction quality, which is established during training and can be controlled via the β parameter. A detailed description of the BNODE concept is provided in the work of (Aka, Brunnemann, Eiden, et al. 2024).

Figure 11 illustrates the concept of the BNODE. The encoders (for state and control) map the input signals and initial system states into the latent space. The state encoder is only required for initialization, as the subsequent latent states are computed by the Neural ODE (NODE). The NODE is a neural network that estimates the state derivatives, from which the next latent states are determined using an ODE solver. These latent states can then be transformed back into the state space using the decoder.

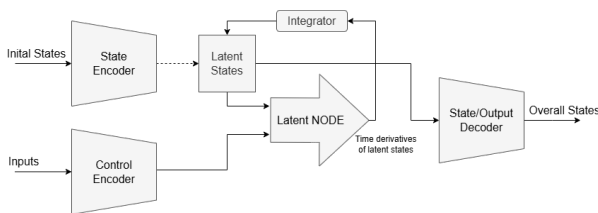


Figure 11. BNODE Concept

For this example, a BNODE was developed for a steam cycle of a coal-fired power plant, based on the concept proposed by (Aka, Brunnemann, Eiden, et al. 2024) and using a training process similar to that described in (Aka 2025). The model replaced by the BNODE originates from the Modelica library ClaRa (ClaRa 2025) (*Examples.SteamCycle_01*) and describes a steam cycle of a coal power plant, considering both mass and energy balances. The original physical model, consisting of 134 states,

29,775 variables, and 8,522 equations, is shown in Figure 12.

Out of the 134 states, the surrogate model approximates 102 states along with 7 additional outputs. Only states related to control elements (e.g., PI controllers), which do not carry essential information about the systems physical state, are omitted. The key parameters of the BNODE surrogate model are listed in the tables 2 and 3.

Table 2. Basic parameters of the BNODE model

Parameter	Value
States	102
Outputs	7
Control inputs	1
Latent states	128
β	0.1

Table 3. Hyperparameter of Encoders, Decoder and Latent ODE

Hyperparameter (for all NNs)	Value
Layers	4
Hidden dimension	64
Activation	ELU

After completion of the training process, the individual neural networks that constitute the BNODE architecture can be exported from the PyTorch format to the ONNX format, which can in turn be integrated into Modelica environments using SMARTInt. The Neural ODE solver employed during training is replaced by the native solver of the Modelica tool, by defining the output of the NODE network as the time derivative of the latent states (see code listing 1).

```
for i in 1:n_lat_states loop
    der(lat_states[i]) =
        LNODE_func.y[i];
end for;
```

Listing 1. Modelica loop of BNODE state derivatives

In addition to the trained networks, the initial states must be provided for model initialization and mapped into the latent space using the state encoder. It is essential that, along with the initial states, the corresponding control input is also specified. After initialization, the control input can be varied within the bounds defined during the training data generation. Figure 13 shows the BNODE brought back to Modelica. The initial states are stored in the *states_tab* table, while the control input is provided in the *P_Target* table. The neural networks that form the BNODE architecture are implemented as individual SMARTInt blocks within the model, and their inputs and outputs are processed at the equation level.

If the same power target (*P_Target*) time series is applied to both the physical model and the surrogate BNODE model, their simulation results can be directly compared. In this example, the control input is initially set to

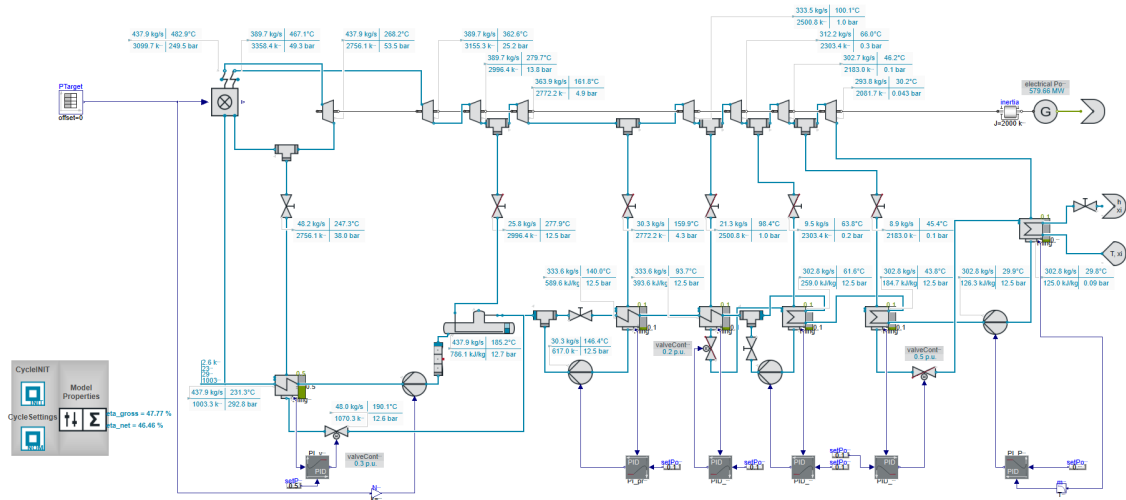


Figure 12. ClaRa: Steam cycle example

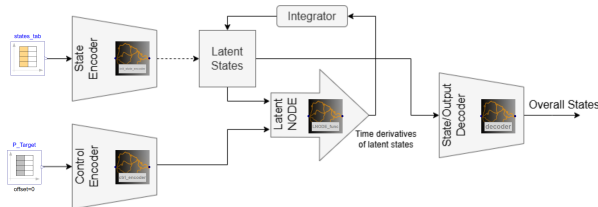


Figure 13. BNODE brought back to Modelica

the BNODE's starting value. After 1000 seconds of simulation time, it is linearly ramped up to 100% (full load) over the course of 100 seconds. Following an additional 900 seconds at full load, the power target of the steam generator is decreased to 70% (partial load), before being increased back to full load at 3000 seconds.

Therefore, the steam cycle displayed by both the BNODE and the physical model experiences a load change. Figure 14 shows the comparison results, including the power target and the seven additional outputs for both the physical model (in red) and the BNODE model (in blue).

The BNODE approximates the results of the physical model quite well, while performing state reduction at the same time. Only 116 of the 128 possible latent state dimensions are utilized. By adjusting the β value, the number of utilized latent state dimensions could be further reduced, although this might result in a decrease in the prediction accuracy of the BNODE. Figure 15 additionally presents a comparison of the computation times of both models. It becomes evident that, especially during dynamic operation, the BNODE requires significantly less computation time compared to the physical model. In this example scenario, the simulation computation time can be reduced by almost a factor of 10 using the BNODE concept. The speed-up factor can increase further in more complex scenarios involving additional load changes, as these have a significant impact on the computation time of the physical model, while leaving the BNODE largely unaffected.

5 The SMARTInt Funding Scheme: Sustaining Open-Source Development

SMARTInt is and will remain an open-source project, continuously evolving with regular updates and releases. To ensure the long-term financial sustainability of its development, it is complemented by a commercial companion version: SMARTInt+ (SMARTIntPlus 2025).

SMARTInt+ extends the functionality of the open-source SMARTInt library by further simplifying its use while also enhancing the quality of the results. The key features of the premium version are listed below:

- Extrapolation Warning
- Modelica Model Generator
- Network Generator (in development)
- Advanced Interfaces
- Tensorflow-Converter (Python)
- Tensorflow-Dedimensionalization-Layer (Python)

As previously mentioned, the individual features of SMARTInt+ are designed to address different aspects of model improvement. One such feature is the **Extrapolation Warning**, which helps ensure the reliability of neural network predictions. Neural networks are highly effective at interpolating within the range of their training data. However, their behavior can get unpredictable when making inferences on unseen data outside that range. In order to identify an extrapolation, SMARTInt+ includes an *Extrapolation Warning* based on three alternative methods: convex hull, kernel density estimation, and a simplified density estimation. These techniques monitor inputs in real time and detect whether the neural network is operating outside its training data.

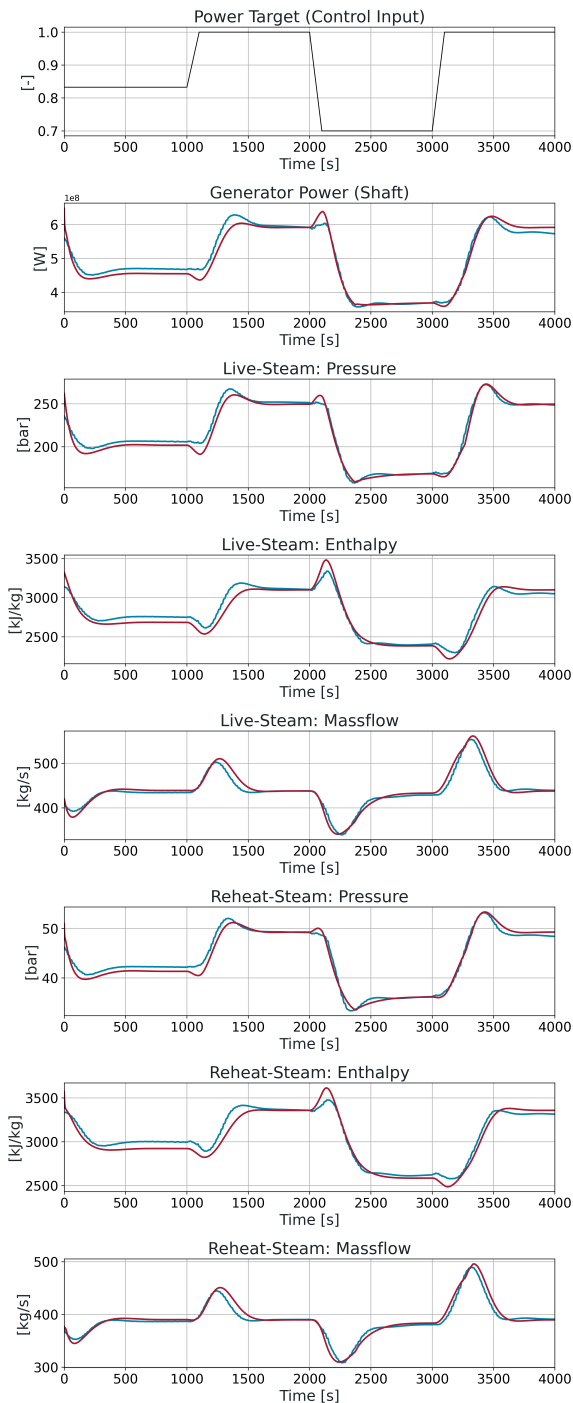


Figure 14. Comparison of different states between physical model (red) and the BNODE surrogate model (blue)

Figure 16 illustrates the extrapolation warning icon triggered during an extrapolation event. While Figure 17 presents a plot showing its status throughout the entire simulation.

The Modelica **Model Generator** automates the integration of pre-trained neural networks (in TFLite and ONNX format), enabling users to incorporate their models into Modelica with minimal effort. Especially when used in combination with the Network Generator (currently under

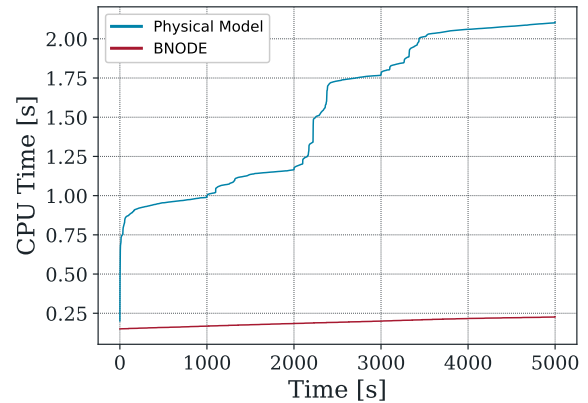


Figure 15. CPU times of the physical model (Dassl-Solver) and the BNODE surrogate model (Euler-Solver)

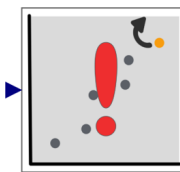


Figure 16. Icon of the ExtrapolationWarning Block

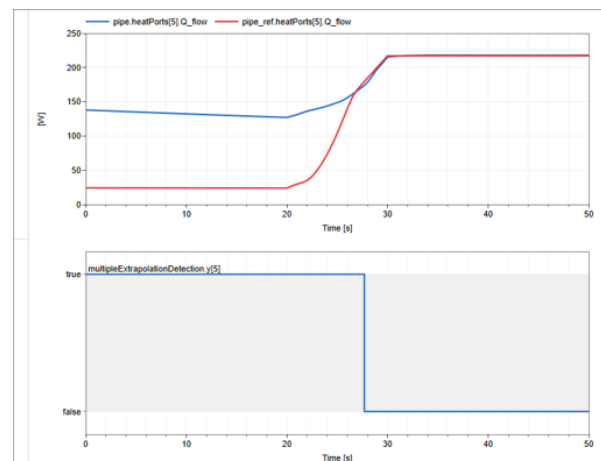


Figure 17. Extrapolation example of a neural network (network prediction and extrapolation curve)

development) which will allow users to define and train sequential feed-forward and recurrent neural networks with TensorFlow directly in Modelica, the Model Generator becomes a useful tool. SMARtInt+ also features enhanced **interface blocks** equipped with input/output smoothing, integrated scaling, and extrapolation monitoring.

The Modelica library also comes with a **Python-based converter** specifically designed for TensorFlow models with internal states (such as sRNNs), enabling their conversion to TFLite format for seamless integration into SMARtInt (see Section 2.2.1). In addition to the converter, SMARtInt+ includes **dedimensionalization layers (PI-Layer)** (Bakarji et al. 2022), which are custom TensorFlow layers that allow dimensionless quantities to be

used as inputs to neural networks, either automatically or manually. This approach reduces the dimensionality of the underlying problem, enabling the use of smaller neural network architectures with fewer layers or units for the same application. Additionally, dedimensionalization improves the generalization capability of AI models. Figure 18 shows the structure of a neural network wrapped with a PI-layer.

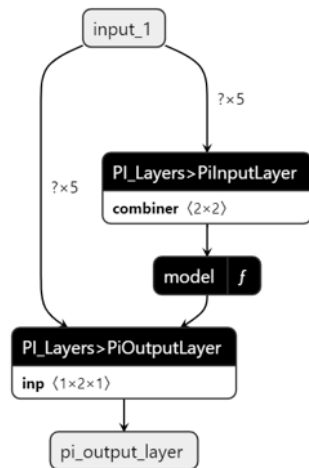


Figure 18. Structure of a neural network wrapped in a dedimensionalization layer

Further information on SMArtInt+ can be found in (SMArtIntPlus 2025).

6 Summary & Outlook

In this paper we have introduced the status of the open-source Modelica library SMArtInt, which enables Modelica users to combine physics based Modelica code with data driven neural networks in a Modelica-based workflow.

With its latest release, users of SMArtInt can now integrate ONNX models into the field of physical Modelica simulations. This enables users to utilize machine learning, deep learning, and other AI techniques from various ONNX-compatible frameworks, thus enhancing the modeling of complex physical systems. The capabilities of SMArtInt are illustrated by examples given in section 4.

SMArtInt is open source and intended to become a Modelica community project and we warmly invite the Modelica community to participate in the libraries improvement by providing user feedback, feature requests and discussing / coding new ideas in order to further enhance Modelica's hybrid modeling capabilities.

The SMArtInt development team will continue development in ongoing and planned research projects.

Acknowledgements

This work has been partially supported by the German Federal Ministry of Education and Research within the DIZPROVI research project under grant number 03WIR0105E.

The authors thank the anonymous referees for valuable comments and suggestions.



References

- Aka, Julius (2025). *Balanced Neural ODEs - Code*. fetched March, 4th 2025. URL: <https://github.com/juliusaka/balanced-neural-odes>.
- Aka, Julius, Johannes Brunnemann, Jörg Eiden, et al. (2024). "Balanced Neural ODEs: nonlinear model order reduction and Koopman operator approximations". In: DOI: <https://doi.org/10.48550/arXiv.2410.10174>.
- Aka, Julius, Johannes Brunnemann, Svenne Freund, et al. (2023-10). "Efficient Global Multi Parameter Calibration for Complex System Models Using Machine-Learning Surrogates". In: *Proceedings of the 15th International Modelica Conference*. Aachen, pp. 107–120. DOI: 10.3384/ecp204107.
- Bakarji, Joseph et al. (2022). "Dimensionally Consistent Learning with Buckingham Pi". In: DOI: <https://doi.org/10.48550/arXiv.2202.04643>.
- Brunnemann, Johannes, Robert Flesch, and Tim Hanke (2024). *DIZPROVI - Digitale Zwillinge für Prozessoptimierung und vorausschauende Instandhaltung : Schlussbericht im WIR! - KOI - Verbundprojekt*. Hamburg. DOI: 10.2314/KXP:1919534997.
- Brunnemann, Johannes, Jan Koltermann, et al. (2022). "Maschinelles Lernen zur vereinfachten Erstellung und Wartung Hybrider Digitaler Zwillinge (Machine Learning for Simplification of Creation and Maintenance of Hybrid Digital Twin Models)". In: *Proceedings of the 24th Colloquium on Power Plant Technology, Dresden, Germany*. in German.
- Chen, Ricky T. Q. et al. (2018). *Neural Ordinary Differential Equations*. DOI: 10.48550/ARXIV.1806.07366.
- ClaRa (2025). Ed. by ClaRa Development Team. version 1.8.2, fetched March, 4th 2025. URL: <https://github.com/xrg-simulation/ClaRa-official>.
- Gijon, Alfonso et al. (2025). *Integrating Physics and Data-Driven Approaches: An Explainable and Uncertainty-Aware Hybrid Model for Wind Turbine Power Prediction*. arXiv: 2502.07344 [cs.LG]. URL: <https://arxiv.org/abs/2502.07344>.
- Kingma, Diederik P. and Max Welling (2019). "An Introduction to Variational Autoencoders". In: *Foundations and Trends in Machine Learning* 12.4, pp. 307–392. ISSN: 1935-8237. DOI: 10.1561/22000000056. URL: <http://dx.doi.org/10.1561/22000000056>.
- Kong, Xiangjie et al. (2025-02). "Deep learning for time series forecasting: a survey". In: *International Journal of Machine Learning and Cybernetics*. ISSN: 1868-808X. DOI: 10.1007/s13042-025-02560-w. URL: <http://dx.doi.org/10.1007/s13042-025-02560-w>.
- Kurz, Stefan et al. (2022). "Hybrid modeling: towards the next level of scientific computing in engineering". In: *Journal of Mathematics in Industry* 12.1, p. 8. DOI: 10.1186/

- s13362-022-00123-0. URL: <https://mathematicsinindustry.springeropen.com/articles/10.1186/s13362-022-00123-0>.
- Lam, Remi et al. (2023). *GraphCast: Learning skillful medium-range global weather forecasting*. arXiv: 2212.12794 [cs.LG]. URL: <https://arxiv.org/abs/2212.12794>.
- Markovic, Romana et al. (2018). “Window Opening Model using Deep Learning Methods”. In: <https://doi.org/10.48550/arXiv.1807.03610>. DOI: <https://doi.org/10.48550/arXiv.1807.03610>.
- Matei, I. (2018). “Learning algorithms for physical systems: challenges and solutions”. In: *Palo Alto Research Center*. <https://www.naefrontiers.org/190462/Matei-presentation>.
- NeoML (2025). Ed. by ABBYY. fetched March, 4th 2025. URL: <https://www.abbyy.com/neoml/>.
- NeuralNetwork (2025). Ed. by Fabio Codeci, Politecnico Di Milano. fetched March, 4th 2025. URL: <https://github.com/modelica-3rdparty/NeuralNetwork>.
- ONNX (2025). Ed. by onnx. fetched March, 4th 2025. URL: <https://onnx.ai/>.
- PyTorch (2025). Ed. by pytorch. fetched March, 4th 2025. URL: <https://pytorch.org/>.
- Quarteroni, Alfio, Paola Gervasio, and Francesco Regazzoni (2025). “Combining physics-based and data-driven models: advancing the frontiers of research with scientific machine learning”. In: *Mathematical Models and Methods in Applied Sciences* 35.04. DOI: 10.1142/s0218202525500125.
- Rudolph, Maja, Stefan Kurz, and Barbara Rakitsch (2024). “Hybrid modeling design patterns.” In: *J.Math.Industry* 14. DOI: <https://doi.org/10.1186/s13362-024-00141-0>.
- Shuangfeng, Li (2020). “TensorFlow Lite: On-Device Machine Learning Framework”. In: *Journal of Computer Research and Development* 57.9, pp. 1839–1853. ISSN: 1000-1239. DOI: 10.7544/issn1000-1239.2020.20200291. URL: <https://crad.ict.ac.cn/en/article/doi/10.7544/issn1000-1239.2020.20200291>.
- Singraval, Sundaravelpandian, Johan Suykens, and Philipp Geyer (2018-10). “Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction”. In: *Advanced Engineering Informatics* 38, pp. 81–90. ISSN: 1474-0346. DOI: <https://doi.org/10.1016/j.aei.2018.06.004>.
- SMarInt (2025). Ed. by XRG Simulation GmbH. version 0.5.1, fetched March, 4th 2025. URL: <https://github.com/xrg-simulation/SMarInt>.
- SMarIntPlus (2025). Ed. by XRG Simulation GmbH. version 0.1.0. URL: <https://xrg-simulation.de/en/seiten/smartint>.
- TensorFlow (2025). Ed. by tensorflow. fetched March, 4th 2025. URL: <https://github.com/tensorflow>.
- Von Krannichfeldt, Leandro, Kristina Orehounig, and Olga Fink (2025-08). “Combining physics-based and data-driven modeling for building energy systems”. In: *Applied Energy* 391, p. 125853. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2025.125853. URL: <http://dx.doi.org/10.1016/j.apenergy.2025.125853>.