A Dymola-Python framework for data-driven model creation and co-simulation

Sandra Wilfling[1] Basak Falay[2] Qamar Alfalouji[1] Gerald Schweiger[1]

[1]Institute of Software Technology, Graz University of Technology, Austria, sandra.wilfling@tugraz.at
[2]AEE INTEC, Austria, b.falay@aee.at

Abstract

The introduction of cyber-physical systems has been a recent development in energy systems. Cyber-physical systems contain digital components for applications such as monitoring or control. In many cases, modeling multiple aspects of such cyber-physical systems poses a challenge to conventional simulation tools. In addition, recent modeling approaches, such as data-driven modeling, are being applied. The combination of such data-driven models, which may consist of a different architecture than traditional models, with traditional models can be implemented through co-simulation methods. In co-simulation, components created from different simulation tools can be combined and coupled through standardized interfaces. This work presents a framework for data-driven model generation and co-simulation. The framework is implemented in Python and Dymola and is based on the Functional Mock-up Interface (FMI) standard. The framework implements the creation of data-driven models in Python, the generation of Functional Mock-up Units (FMUs) through the frameworks *uniFMU* and *pythonFMU*, as well the creation of a testbench model in Dymola and the co-simulation of this model. The framework is demonstrated on the application of a solar collector from a single family house heating system.

Keywords: Energy Systems, Modeling and Simulation, Data-driven Modeling, Co-Simulation

1 Introduction

The area of energy systems covers a wide range of applications, such as heating, cooling or electrical power systems. All these systems have in common that their demand for energy must be met by the energy providers while their energy demands are constantly growing. To respond to the increasing demand, energy providers have recently been focusing on embedding cyber-technologies into their systems in order to monitor and optimize system operation. This means that state-of-the-art energy systems are being extended into complex cyber-physical systems (Lund et al., 2017). The analysis of such cyber-physical energy systems poses new challenges in the area of simulation and modeling due to these systems' complexity (Palensky, 2014). Cyber-physical systems combine computational systems with other physical systems, meaning that their analysis requires combined modeling techniques for different system types. While the modeling of certain components can be implemented in specialized simulation tools, the full modeling of a combined system is a more difficult task. To model cyber-physical energy systems, different approaches exist, which can be classified into three groups: white-box, gray-box and black-box modeling (Arendt et al., 2018). White-box methods include traditional physical modeling methods based on system dynamics. Gray-box models may also be based on system dynamics, but may contain assumptions or approximations. Black-box models may consist of a completely different architecture than the underlying system. Traditionally, energy systems are modeled in simulation tools based on the physical relations of their components. Physical models are created by analysing the physical properties of the system, and these models are implemented mostly as white-box or gray-box models and based on the knowledge of the system dynamics and parameters. In order to model and simulate these systems, often numerical solvers are used to solve the underlying differential equations, as described by (Gomes et al., 2018). The numerical simulation methods are then implemented by simulation tools such as, for instance, Dassault Systemes Dymola®, MathWorks® Matlab/Simulink or EnergyPlus™. In contrast to traditional modeling, the data-driven modeling approach has recently been gaining popularity. Data-driven models are mainly based on modeling the underlying system as a black box. This means that the architecture and the parameters of the system are arbitrary, any structure can be used as a model. Data-driven models are mainly implemented by machine learning (ML) methods, such as linear regression models, decision-tree based models or neural networks. In the data-driven approach, the models are trained on existing measurement data by using optimization methods. This approach was applied for instance in (Ghofrani et al., 2020) and (Xu et al., 2019). The advantage of the data-driven modeling approach is that the ML models are trained based on measurement data and do not require exact system knowledge and parameters. While domain knowledge is helpful in creating the models, it is not necessary to know all features of the underlying system beforehand. A recent approach in cyber-physical systems modeling is the combination of physical and data-driven models in a co-simulation (CS) environ-

ment. The term co-simulation describes the combination of different simulation tools or environments. This may include a combination of continuous-time and discrete-time models, as well as simulation tools like Dymola or Matlab/Simulink. In co-simulation, different systems are integrated into a global environment. The co-simulation approach is used in applications such as building control systems, especially in model-predictive control (Wang et al., 2019). Applications in energy systems modeling or control often contain feedback loops containing components implemented in different simulation tools. These components must be coupled with each other through a defined interface. For this purpose, organizations such as the Modelica Association or the Institute of Electrical and Electronics Engineers (IEEE) have developed standards for co-simulation interfaces, such as the High-Level Architecture (HLA) (IEEE, 2010) or FMI (Modelica Association, 2020) standard. These standardized interfaces can be implemented by various tools without having to adapt the models for each simulation environment and are supported by different simulation tools. Additionally, these interfaces can be implemented by data-driven models, which may be created in programming languages such as Python. For our work, the FMI standard was selected. The FMI standard is developed by the Modelica Association, with current version FMI 2.0 (Modelica Association, 2020). The standard defines an interface for coupling models of different types and architectures. The FMI standard defines the format of models that are compatible to the standard as FMU. Simulation tools such as Dassault Systemes Dymola® (Dymola) or Simulink offer the option to generate FMUs from an existing model. For data-driven models, there are open-source tools available to export these models into the FMU format, such as the *pythonFMU* framework (Hatledal et al., 2020) and the *uniFMU* framework (Legaard et al., 2021).

1.1 Related Work

In energy systems modeling, different co-simulation frameworks have been created for the purpose of combining models created different simulation tools. For instance, several frameworks based on the FMI standard have been developed.The *Maestro* framework (Thule et al., 2019) implements a co-simulation orchestration engine for discrete-time and continuous-time co-simulation. The framework is implemented in Java, Scala and C and is based on the FMI standard. This framework supports Hardware-in-Loop (HiL) co-simulation. The *CyDER* (Nouidui et al., 2019) framework focuses on simulation for smart power grids. The framework is implemented in Python and suports HiL simulation. The *CyDER* framework offers the tool *Simulator2FMU*, which makes the interfaces of different power grid simulators compatible to the FMI standard. The main simulation is executed through the Python library *PyFMI* . In addition, smaller frameworks that focus on certain simulation tools have been developed. For the communication between Python and Dymola, several Python libraries have been developed. The Python library *buildingspy* (Wetter and USDOE, 2019) supports communication from Python to Dymola as well as to the Modelon Inc. *OPTIMICA* Compiler Toolkit. The Python package *dymat* (Rädler, 2013) supports reading and writing of Dymola output files. Based on existing Python libraries, different co-simulation frameworks have been developed. A Python-Modelica framework specialized for wind turbines called *MoWIT* was created in (Leimeister, 2019). This framework is based on the *buildingspy* library. Another framework called *PyMo* was created by (Febres et al., 2014).

1.2 Main Contribution

This work presents a workflow called *HybridCosim* that combines the creation of data-driven models with co-simulation. In this workflow, data-driven models are automatically created and then combined with physical models inside a co-simulation environment. The workflow is based on the FMI standard 2.0. The data-driven models are created in Python, converted into FMUs, and then simulated in Dymola as a part of an automatically generated testbench. The framework supports the creation of ML models of different architectures, as well as simulation in Dymola. The framework is demonstrated on a case study of a solar collector.

2 Methodology

The presented framework consists of four steps. Firstly, a data-driven model of an existing system is trained in Python. This model is then converted into an FMU. For the FMU, a Modelica testbench model is generated. Finally, the testbench is simulated in Dymola. An overview of the created workflow is given in Figure 2.

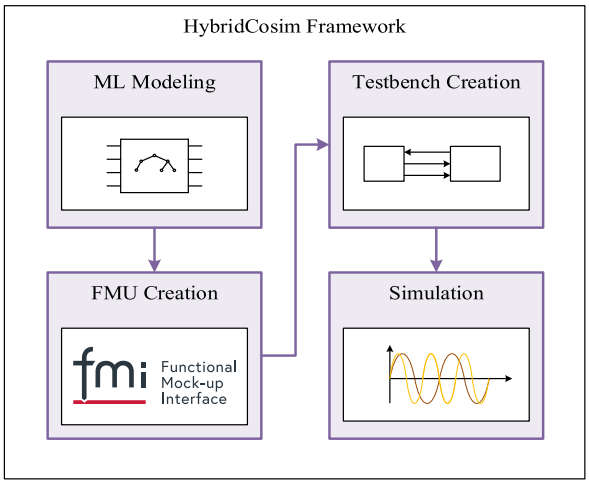

Figure 1. Co-Simulation Workflow

The first three steps of the workflow are executed purely in Python, the last step is executed through Python and Dymola. While the simulation itself is executed in Dymola, the orchestration and the result post-processing are done in Python. This framework is based on the research in (Falay et al., 2021) and (Wilfling et al.).

DOI
10.3384/ecp193165

2.1 Model Training

To create data-driven models, we implemented a basic framework in Python to train models of different architectures, such as linear regression models, decision tree-based models or Support Vector Machine (SVM) models. These models could be created based on different datasets and feature configurations. The models are based on the research in (Schranz et al.) and the Python packages *scikit-learn* (Pedregosa et al., 2011) and *statsmodels* (Seabold and Perktold, 2010).

2.2 Interfacing - FMI

In our work, the FMI standard was used as an interface between models of different types. Therefore, the models had to be converted into the FMU format, for which the *uniFMU* framework (Legaard et al., 2021) and the *pythonFMU* framework (Hatledal et al., 2020) were evaluated. The *uniFMU* framework allows to export models from different programming languages such as Python, C#, Matlab or Java into an FMU. *uniFMU* supports the FMI standard 2.0 and contains a graphical user interface to generate and validate FMUs. The *pythonFMU* framework supports FMU generation from Python files.

FMU Creation

In our work, machine learning models implemented in Python can be translated into FMU format through the *pythonFMU* or *uniFMU* framework. While the *pythonFMU* framework supports the generation of a full FMU from a Python model, the *uniFMU* framework requires additional steps for creating the FMU. The FMU format contains a model description in Extensible Markup Language (XML), in which the model interface, consisting of the model inputs, outputs and parameters, and the basic model structure, which may include dependencies, is defined. To create an FMU through *uniFMU*, the model description must be adapted to the interface of the model.

For the FMU creation through *uniFMU*, a method to adapt the FMU model description automatically depending on the required inputs and outputs for the model was created. When using the framework, either of the two frameworks can be selected.

2.3 Automatic Testbench Creation

In our framework, Dymola was selected as the main simulation master, therefore our top-level model had to be implemented in Dymola. To automatically create a simple testbench for the FMU, a Python module was created. This module could generate a Modelica model based on input data, a specification of input and output features, and the FMU file. In addition, components created in Modelica could be imported and added to the model. The data-driven model was imported into Dymola and connected to Dymola-native modules or other FMUs. With this structure, it was possible to create fully-coupled systems, such as feedback control loops, or simpler systems with fewer components.

2.4 Simulation

For the generated top-level model including the FMU, a co-simulation was executed in the Dymola environment. This simulation was implmented using parts of the process created in (Wilfling et al.). In our implementation, the main control for the simulation is implemented in Python. The Python controller then sends commands to Dymola, which executes the simulation. The simulation commands are based on Modelica .mos scripts, which are automatically generated in Python. Figure 2 gives an overview of the implemented simulation method.

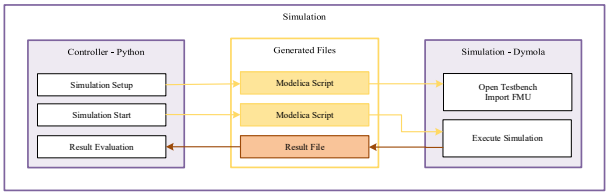

Figure 2. Python-Dymola Communication, c.f. (Wilfling et al.)

Alternatively, the testbench could be simulated directly through Dymola.

2.5 Framework Implementation

The framework was implemented mainly in Python. The framework is structured into four Python packages, each of which contains a step of the workflow. For each package, an example testscript is available to execute the operations of the step. In addition, all steps can be executed in combination as a full workflow run. In this case, the four steps are executed sequentially. During the execution of each workflow step, different files are created, which are then used by the next steps. The combined workflow requires two components as inputs: a dataset, and a configuration file containing definitions of the model inputs and outputs. Figure 3 depicts the full workflow structure with input and output files.

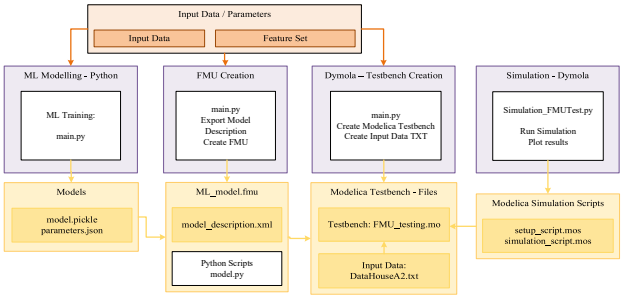

Figure 3. Workflow structure. Input files to the workflow are marked in red, automatically created output files are marked in yellow.

File Structure

The results of an experiment using the combined framework are stored inside a directory structure containing all

automatically generated files including the models and the simulation results. An overview of this structure is given in Figure 4.

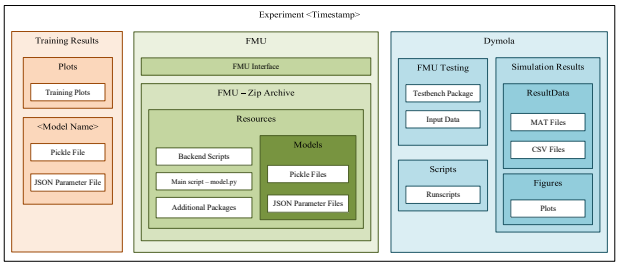

Figure 4. Directory Structure.

The structure is separated into three directories: one for the model training results, one for the FMU files, and one for the Dymola testbench and simulation results.

3 Case Study

To demonstrate the proposed framework, a case study on a use case from the energy domain was performed. For this purpose, a solar collector from a single-family house heating system was selected (Wilfling et al.). In a single-family house, the main heating demand is generated from the central heating for the rooms and the warm water consumption. In order to give options to optimize the heating energy consumption of such a house, the heating system should be modeled as accurately as possible. For this purpose, two different architectures for the data-driven model were evaluated.

3.1 Application - Solar Collector

The application of the case study was the supply temperature prediction for a flat-plate solar collector. This collector, which was already available as a physical model (Falay et al., 2021), should be modeled through a data-driven model. For the collector, the supply temperature T_S should be predicted based on the return temperature T_R, the mass flow through the collector V_d, the ambient temperature T_A and the solar radiation S_{Global}.

Underlying System

According to (Mahanta, 2020), the behavior of a flat-plate solar collector can be modeled through linear relations. The main factors affecting the solar collector supply temperature are the heat gain through the solar radiation and the heat loss to the ambient. While in the active state of the collector, the heat gain is affected by the mass flow through the collector. A simplified version of these relations can define the active behavior of the collector through Equation 1:

$$T_S = T_R + \frac{C_1 S_{Global}}{V_d} + \frac{C_2 (T_S - T_A)}{V_d} \quad (1)$$

3.2 Data-driven Model

For the solar collector, a data-driven model was created through the model training part of the framework. To compare different model architectures, two models were created, one consisting of a linear regression model and one using Random Forest (RF) regression. The models were trained based on measurement data in a duration from 02/2019 to 10/2019, which was sampled with a timestep of 15 min. For the training, a train-test split of 0.8 was selected. The trained models were stored in the Pickle format.

3.3 FMU Creation and Testbench Generation

From the trained models, an FMU was created. Afterwards, a Dymola model to test the FMU was generated. This Dymola model was generated using the input measurement data and the description of the FMU inputs and outputs. Figure 5 depicts the generated Dymola model.

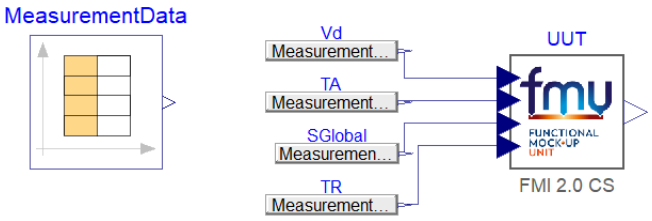

Figure 5. Graphical depiction of the generated Dymola model. The component placement was adapted manually for visualization.

The Dymola model contains the FMU and a Modelica *CombiTimeTable* containing the measurement data. For the *CombiTimeTable*, a text file was automatically generated from the input data to act as datasource.

3.4 Experimental Results

Finally, a simulation was executed for the generated Dymola model. The simulation duration was set to a time window of 30 days, with a timestep of 15 min. The results were post-processed in Python.

Performance Metrics

To evaluate the performance of the model, the metrics Coefficient of Determination (R^2), Coefficient of Variation of the Root Mean Square Error (CV-RMSE) and Mean Absolute Percentage Error (MAPE)(Falay et al., 2021) were selected. The performance metrics for the model are described in Table 1.

Table 1. Performance Metrics

Model	R^2	CV-RMSE	MAPE
Linear Regression	0.94	0.06	4.58%
Random Forest	0.98	0.04	2.21%

DOI
10.3384/ecp193165

The performance metrics show the higher accuracy of the RF model. However, the linear regression model performs only slightly worse than the RF model despite its simple structure.

Timeseries Analysis

Figure 6 shows the timeseries analysis for the solar collector for a selected period of five days from the simulation duration. The timeseries analysis shows more accurate predictions of the data-driven model during daytime than during nighttime. This behavior was accredited to the characteristics of the solar collector, which is inactive during nighttime.

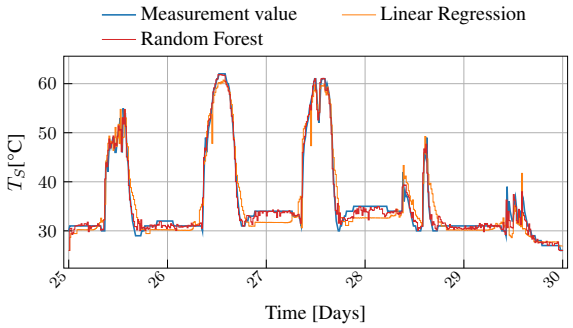

Figure 6. Timeseries Analysis for selected period from simulation.

The prediction error plots for the solar collector case study during the full simulation duration are depicted in Figure 7.

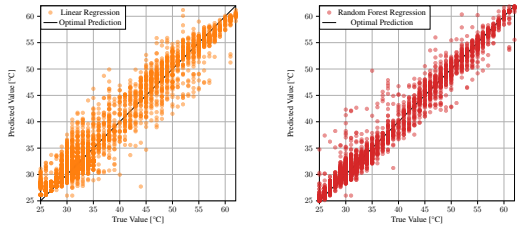

Figure 7. Predicted Values for T_S

From the prediction error plots, the higher accuracy of the RF regression model can be observed. The distribution of the residual error does not show significant anomalies.

4 Conclusion

We present a framework for data-driven model creation and co-simulation that allows the combination of different models. The framework is implemented in Python and Dymola and is based on the FMI standard. This framework allows automatic creation of data-driven models, translation into the FMU format, creation of a Dymola testbench model and simulation in Dymola. A case study performed on an application from the energy domain showed the performance of the created data-driven models.

4.1 Future Work

The current version of the framework gives many options for extensions. For instance, it is possible to extend the model training part of the framework to support additonal model types. The FMU creation part of the framework could be extended to support FMI 3.0, as well as include further extensions from FMI 2.0. Finally, the Dymola simulation part could be extended to support different simulation masters such as OpenModelica.

Acknowledgements

This work was funded through the project *"Next Generation Energy Services for Intelligent Buildings and Energy Systems“ (NextGes)* as part of the Styrian *Zukunftsfonds Steiermark - "NEXT GREEN TECH"*.

References

K Arendt, M Jradi, H R Shaker, and C T Veje. Comparative Analysis of white-, gray- and black-box models for thermal simulation of indoor environment: Teaching Building Case Study. In *2018 Building Performance Modeling Conference and SimBuild Co-Organized by ASHRAE and IBPSA-USA Chicago*, page 8, 2018.

Basak Falay, Sandra Wilfling, Qamar Alfalouji, Johannes Exenberger, Thomas Schranz, Christian Møldrup Legaard, Ingo Leusbrock, and Gerald Schweiger. Coupling physical and machine learning models: Case study of a single-family house. *Modelica Conferences*, pages 335–341, September 2021. ISSN 1650-3740. doi:10.3384/ecp21181335.

Jesús Febres, Raymond Sterling, and Marcus Keane. A Python-Modelica Interface for Co-Simulation. *International Conference on Sustainability in Energy and Buildings*, page 11, 2014.

Ali Ghofrani, Seyyed Danial Nazemi, and Mohsen A. Jafari. Prediction of building indoor temperature response in variable air volume systems. *Journal of Building Performance Simulation*, 13(1):34–47, January 2020. ISSN 1940-1493, 1940-1507. doi:10.1080/19401493.2019.1688393.

Claudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a survey. *ACM Computing Surveys (CSUR)*, 51(3):1–33, 2018.

Lars Ivar Hatledal, Houxiang Zhang, and Frederic Collonval. Enabling Python Driven Co-Simulation Models With PythonFMU. In *ECMS 2020 Proceedings Edited by Mike Steglich, Christian Mueller, Gaby Neumann, Mathias Walther*, pages 235–239. ECMS, June 2020. ISBN 978-3-937436-68-5. doi:10.7148/2020-0235.

IEEE. IEEE Std 1516’-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA),Framework and Rules. page 41, 2010.

Christian Møldrup Legaard, Daniella Tola, Thomas Schranz, Hugo Daniel Macedo, and Peter Gorm Larsen. A universal mechanism for implementing functional mock-up units.

In *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH 2021, page to appear, Virtual Event, 2021.

Mareike Leimeister. Python-Modelica Framework for Automated Simulation and Optimization. In *The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, pages 51–58, February 2019. doi:10.3384/ecp1915751.

Henrik Lund, Poul Alberg Østergaard, David Connolly, and Brian Vad Mathiesen. Smart energy and smart energy systems. *Energy - The International Journal*, 137:556–565, October 2017. ISSN 03605442. doi:10.1016/j.energy.2017.05.123.

Deba Kumar Mahanta. Mathematical Modeling of Flat Plate Solar Collector. In *2020 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, pages 1–5, December 2020. doi:10.1109/PEDES49360.2020.9379669.

Modelica Association. Functional Mock-up Interface 2.0.2. 2020.

Thierry S. Nouidui, Jonathan Coignard, Christoph Gehbauer, Michael Wetter, Jhi-Young Joo, and Evangelos Vrettos. CyDER – an FMI-based co-simulation platform for distributed energy resources. *Journal of Building Performance Simulation*, 12(5):566–579, September 2019. ISSN 1940-1493. doi:10.1080/19401493.2018.1535623.

Palensky. Simulating Cyber-Physical Energy Systems: Challenges, Tools and Methods. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(3): 318–326, March 2014. ISSN 2168-2216, 2168-2232. doi:10.1109/TSMCC.2013.2265739.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jörg Rädler. Dymat-hdf5 export and other modelica/python projects, 2013.

Thomas Schranz, Johannes Exenberger, Christian Møldrup Legaard, Jan Drgona, and Gerald Schweiger. Energy prediction under changed demand conditions:robust machine learning models and input feature combinations. In *Building Simulation 2021. International Building Performance Simulation Association*.

Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

Casper Thule, Kenneth Lausdahl, Cláudio Gomes, Gerd Meisl, and Peter Gorm Larsen. Maestro: The INTO-CPS cosimulation framework. *Simulation Modelling Practice and Theory*, 92:45–61, April 2019. ISSN 1569-190X. doi:10.1016/j.simpat.2018.12.005.

Jiangyu Wang, Shuai Li, Huanxin Chen, Yue Yuan, and Yao Huang. Data-driven model predictive control for building climate control: Three case studies on different buildings. *Building and Environment*, 160:106204, August 2019. ISSN 03601323. doi:10.1016/j.buildenv.2019.106204.

Michael Wetter and USDOE. Buildingspy, 4 2019. URL `https://www.osti.gov//servlets/purl/1569219`.

Sandra Wilfling, Basak Falay, Qamar Alfalouji, Johannes Exenberger, Thomas Schranz, Mina Basirat, and Gerald Schweiger. Smart Energy Systems Modeling - Component Exchange during Simulation. page 10.

Chengliang Xu, Huanxin Chen, Jiangyu Wang, Yabin Guo, and Yue Yuan. Improving prediction performance for indoor temperature in public buildings based on a novel deep learning method. *Building and Environment*, 148:128–135, January 2019. ISSN 03601323. doi:10.1016/j.buildenv.2018.10.062.