



Dipl.-Ing. Thomas Schranz, BSc

Data-Driven Modeling of Cyber-Physical Systems: Machine Learning for Building Simulation

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Franz Wotawa, Univ.-Prof. Dipl.-Ing. Dr.techn.
Institute of Software Technology

Graz, May 2023

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date, Signature

Abstract

Resource-intensive industrial and agricultural processes, transportation, and energy-intensive indoor climate control require vast amounts of energy. Ensuring energy security through fossil sources and the ensuing emission of greenhouse gases profoundly affect the environment. Policies to cut energy-related greenhouse gas emissions aim to reduce overall energy demand and increase the share of sustainable energy sources. By integrating sensors and actuators into energy-consuming processes, we can deploy automated operation strategies that increase energy efficiency and synchronize availability with demand. We typically develop such strategies based on dynamical models. Machine learning (ML) methods are well suited for learning the behavior of non-deterministic processes or complex physical phenomena from observations. However, there are issues impeding the use of ML-based models in areas with significant energy-saving potentials, such as building simulation.

This thesis describes the application of ML methods for modeling dynamical processes in cyber-physical systems such as *smart* buildings. In the first part, we formalize the task of time-series analysis as a regression problem and discuss the intricacies of prediction and forecasting. We provide practitioners with guidelines and a sandbox framework for testing and benchmarking ML architectures for time-series analysis. We describe two case studies concerned with the water supply and the energy demand in a university building. To address the inherent shortcomings of traditional ML methods, we investigate the paradigm of *theory-informed* ML. We provide the reader with a comprehensive taxonomic overview of state-of-the-art, theory-driven, neural-network-based simulation methods. We identify common mechanisms and showcase their application in a case study example. In the second part, we investigate how to use ML techniques in conjunction with other system simulation methods. We discuss the paradigm of co-simulation and the problems that ensue from using scripting languages with co-simulation standards. We present an open-source framework for creating simulation units from any programming language and showcase its application in a case study example. Finally, we explore the end-to-end integration of simulation models into cyber-physical systems. We present results from an expert survey and a literature review on middleware infrastructure. We discuss requirements and concepts for software systems that collect data and deliver results from ML models to a physical system and its operators.

Kurzfassung

Ressourcenintensive Industrie- und Agrarprozesse, der Transport und die energieintensive Klimatisierung von Gebäuden erfordern große Mengen an Energie. Die Versorgung durch fossile Energieträger und die daraus resultierende Treibhausgasemissionen haben tiefgreifende Auswirkungen auf die Umwelt. Politische Maßnahmen zur Verringerung der energiebedingten Treibhausgasemissionen zielen darauf ab, den Gesamtenergiebedarf zu senken und den Anteil nachhaltiger Energiequellen zu erhöhen. Durch die Integration von Sensoren und Aktoren in energieverbrauchende Prozesse können wir automatisierte Betriebsstrategien einsetzen um die Energieeffizienz zu erhöhen und die Verfügbarkeit mit der Nachfrage zu synchronisieren. In der Regel verwenden wir zur Entwicklung solcher Strategien dynamische Modelle. Methoden des maschinellen Lernens (ML) sind gut geeignet, um das Verhalten nicht-deterministischer, dynamischer Prozesse oder komplexer physikalischer Phänomene aus Beobachtungen zu lernen. Es gibt jedoch Probleme, die den Einsatz von auf ML basierenden Modellen in Bereichen mit erheblichem Energiesparpotenzial, wie z. B. der Gebäudesimulation, behindern. Diese Arbeit beschreibt die Anwendung von Methoden des ML zur Modellierung dynamischer Prozesse in cyber-physikalischen Systemen wie *intelligenten* Gebäuden. Im ersten Teil formalisieren wir die Aufgabe der Zeitreihenanalyse als Regressionsproblem und diskutieren die Details von Vorhersagen und Prognosen. Wir geben Anwender:innen Leitlinien und eine Entwicklungsumgebung zum Testen und Vergleichen von Architekturen für die Zeitreihenanalyse. Wir beschreiben zwei Fallstudien, die sich mit der Wasserversorgung und dem Energiebedarf in einem Universitätsgebäude befassen. Um die Unzulänglichkeiten traditioneller Methoden des ML zu beheben, untersuchen wir Methoden des *theoriegeleiteten* ML. Wir geben Leser:innen einen umfassenden taxonomischen Überblick über theoriegeleitete, auf neuronalen Netzen basierende Simulationsmethoden. Wir identifizieren gemeinsame Mechanismen und zeigen ihre Anwendung in einem Fallstudienbeispiel auf. Im zweiten Teil untersuchen wir, wie Techniken des ML in Verbindung mit anderen Simulationsmethoden eingesetzt werden können. Wir diskutieren das Paradigma der Co-Simulation und die Probleme, die sich aus der Verwendung von Skriptsprachen mit Co-Simulationsstandards ergeben. Wir stellen eine Entwicklungsumgebung für die Erstellung von Simulationseinheiten aus beliebigen Programmiersprachen vor und zeigen seine Anwendung in einem Fallstudienbeispiel. Schließlich untersuchen wir die Integration von Simulationsmodellen in cyber-physische Systeme. Wir präsentieren die Ergebnisse einer Expertenbefragung und einer Literaturübersicht über Middleware-Infrastrukturen. Wir erörtern Anforderungen und Konzepte für Softwaresysteme, die Daten sammeln und Berechnungen von Modellen an physische Systeme und seine Betreiber:innen liefern.

To the wonderful people
without whom my delusions of grandeur could not prevail
to Susanne, Erika, Stefan,
to my friends and colleagues,
to Hirschi, Kathi, and Christian,
thank you.

Contents

1	Introduction	1
1.1	Dynamical Systems	2
1.2	Machine Learning for Simulation	3
1.2.1	Limitations	3
1.2.2	Domain Knowledge	4
1.3	Co-Simulation	4
2	Objectives, Contributions and Structure	5
2.1	Research Objectives	5
2.2	Contributions	7
2.2.1	Publications and Collaborations	8
2.3	Structuring Content, Related Work and Methodology	9
2.3.1	Related Work and Methodology	10
3	Traditional ML Methods for Dynamical System Simulation	11
3.1	Prediction and Forecasting	11
3.2	Algorithms	13
3.3	Performance Metrics	13
3.4	Case Studies at the Graz University of Technology	15
3.4.1	Background	16
3.4.2	Implementation	16
3.4.3	Water Supply Supervision	16
3.4.4	Energy Demand Estimation	25
4	Theory-Informed Deep Learning for Dynamical System Simulation	35
4.1	Related Work	35
4.2	The Initial Value Problem	36
4.3	Dependence on Time	37
4.4	Case Study	37
4.5	Neural Networks	39
4.6	Model Taxonomy	39
4.7	Direct-Solution Models	40
4.7.1	Network Architecture, Training, Evaluation	41
4.7.2	Vanilla Direct-Solution	41
4.7.3	Automatic Differentiation in Direct-Solution models	43
4.7.4	Physics-Informed Neural Networks	43

4.7.5	Hidden-Physics Networks	46
4.8	Time-Stepper Models	46
4.8.1	Network Architecture, Training, Evaluation	48
4.8.2	Integration Schemes	49
4.8.3	External Input	53
4.8.4	Network Architectures	55
4.8.5	Graph Time-Steppers	57
4.8.6	Uncertainty	58
4.9	Discussion	63
4.9.1	Open Questions	64
5	Co-Simulation	65
5.1	Coupling Techniques	65
5.2	The Functional Mockup Interface	66
5.3	Practical Example	67
5.3.1	Robotic Arm	68
5.3.2	Controller	68
5.3.3	Robotic Arm FMU	69
5.3.4	Controller FMU	69
5.3.5	Creating an FMU in UniFMU	71
5.3.6	Docker Support	72
5.3.7	Simulation	73
5.3.8	Discussion	75
6	End-to-End Integration	77
6.1	Background	77
6.2	Technologies and Related Work	78
6.2.1	IoT Middleware Surveys	78
6.2.2	Open Standards	78
6.2.3	Graz University of Technology	79
6.3	Implementation	80
6.3.1	Sensors	80
6.3.2	Data Collection	80
6.3.3	Data Persistence	81
6.3.4	Database Updates	83
6.3.5	Data Management and Data Access	83
6.3.6	Data Analysis	84
6.3.7	Data Visualization	86
6.4	Discussion	87
7	Conclusion and Future Extensions	89
7.1	Open Questions	92
	Bibliography	95

List of Figures

3.1	Regular water consumption on workdays and holidays	19
3.2	Water consumption during/outside the semester	20
3.3	Feature engineering pipeline	21
3.4	Predicted/actual water consumption normal/faulty behavior	24
3.5	Lookback and lookahead	26
3.6	Comparison of the CV-RMSE for all model and feature combinations. . .	32
4.1	An ideal pendulum	37
4.2	Solutions to the ideal pendulum IVP	38
4.3	Network-based Simulation Models	40
4.4	Vanilla direct-solution model	42
4.5	Automatic differentiation in a direct-solution model	44
4.6	Physics-informed direct-solution model	45
4.7	Hidden physics network	47
4.8	Initial steps for time-stepper models	48
4.9	Direct time-stepper model	49
4.10	Residual time-stepper model	50
4.11	Euler time-stepper model	52
4.12	Neural ordinary differential equations	53
4.13	External inputs in a time-stepper model	54
4.14	Lagrangian time-stepper	56
4.15	Deep Markov model	59
4.16	Latent neural ordinary equations	60
4.17	Bayesian NODEs	61
4.18	Stochastic NODEs	62
5.1	Robot-controller connection	67
5.2	Standalone test of robotic arm	70
5.3	Standalone controller test	71
5.4	UniFMU Python FMU directory structure	72
5.5	Deployment of a Dockerized FMU	74
5.6	Co-simulation of the controller and robot	74
6.1	The MQTTAgent handling sensor updates.	81
6.2	The database wrappers.	82
6.3	The MQTT data collection pipeline	84
6.4	The prediction service	85
6.5	Showcase Grafana dashboard	87

List of Tables

3.1	Candidate values for the hyperparameter search	22
3.2	Best results from the hyperparameter search	23
3.3	Results for random forest and neural network	23
3.4	Energy demand forecast, results	31
4.1	Comparison of direct-solution and time-stepper models	63
5.1	FMU-enabled libraries/tools	67

1 Introduction

Socioeconomic changes, such as the advent of resource-intensive industrial and agricultural processes, the integration of electronic devices into every aspect of day-to-day life, and the use of energy-intensive heating, ventilation, air conditioning and cooling (HVAC) require unprecedented amounts of energy. Since 1950, global primary energy consumption has increased by over 500% from 28,516 terawatt-hours (TWh) to 173,340 TWh per year [HR20]. As of 2019, fossil energy sources, such as oil, coal, and gas, provided 84.3% of global primary energy and 63.3% of global electricity [HR20].

Production, distribution, and utilization of fossil fuels are resource intensive and produce vast quantities of greenhouse gases. Resource depletion and the emission of greenhouse gases profoundly impact the environment. The increasingly frequent occurrence of extreme weather events [Sto16] and the loss of habitable land, farmable soil, and unpolluted water to climate change [SSB+19] constitute one of the gravest challenges of our time [AEK+15]. As a reaction, governments and organizations worldwide have committed to reducing human impact on the environment.

The European Union (EU) composed a comprehensive list of policies to cut greenhouse gas emissions and become the first climate-neutral continent by 2050 [EC19]. With a share of 78%, the energy sector is the single largest contributor of carbon emissions [Eur20] in the EU. There are two principal approaches to reducing energy-related greenhouse gas emissions: (i) reducing overall energy demand and (ii) generating a larger share of energy from sustainable, climate-neutral sources.

The transition from fossil fuels to sustainable energy sources constitutes a profound paradigm shift. The availability of renewable energy from sources such as solar photovoltaic (PV) systems or wind turbines depends on factors beyond human control, such as the day and night cycle, the intensity of solar radiation, cloudiness, and wind. Consequently, power station operators can no longer govern energy production based on demand. Instead, energy consumers, including the industry, must adapt and schedule energy-consuming processes around the availability. Strategies to match demand and production require the means to estimate both measurements accurately.

In recent years, the buildings sector has become the single largest contributor to humanity's energy demand, accounting for over 40% of primary energy consumption in the United States and the EU [CDL16]. In 2019, CO₂ emissions from building construction and operation had increased to 38% of global energy-related CO₂ emissions, and the electric energy demand in buildings accounted for close to 55% of global electricity consumption[Uni20]. It is apparent that building designers and operators play a vital role

1 Introduction

in reducing building energy demand through efficient design, construction, and operation [CDL16]. Chwieduk [Chw03] highlighted the importance of environmentally-friendly energy technologies in the residential building stock. With the integration of microcontrollers, buildings are transitioning into cyber-physical systems, where embedded digital components monitor and control physical processes [LS17]. This development provides operators with new methods to automate control, predictive maintenance, diagnosis, and fault correction. However, the transition introduces new challenges as well. The rapid development of new devices, standards, protocols, and data formats results in interoperability issues. Solving these interoperability problems has become a key research interest [BK20; RMP+16; NGM+16].

We can develop strategies to synchronize energy demand with production and services that improve energy efficiency based on simulations of the systems that consume and produce energy. In the next part of this opening chapter, we discuss system simulation and how recent advances in technology and methodology have changed dynamical system models.

1.1 Dynamical Systems

The study of dynamical systems has a variety of applications throughout all scientific fields. In the context of dynamical system theory, we generally view a system as a group of cohesive, interrelated, interacting components and describe its state through a set of observations [Bac00]. To characterize the behavior of a system, we describe its state changes over time [BS02]. A formal description of the system's behavior constitutes a conceptual model, and a sequence of observations from a conceptual model is a simulation. Dynamical system theory uses functions to describe how the states of a system evolve. Typically, these functions are implicit relations of the current system state to one future state for a given time interval in the form of differential equations.

$$\dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t)) \quad (1.1)$$

Hence, to obtain a conceptual model, it is necessary to solve the *initial value problem* (IVP) constituted by the equations and the system's initial state. The solution to the IVP is another function that determines the system state for any point in time.

$$\mathbf{y}(t) = g(t) \quad (1.2)$$

In many instances, it is impossible to derive an analytical solution to the IVP, and even if a closed-form solution exists, it can be computationally expensive to calculate. Instead, it is common to approximate the solution numerically. For a comprehensive introduction to dynamical systems and numerical approximation, we refer the reader to the seminal work of Birkhoff [Bir27], or more recent works by Arrowsmith et al. [APP+90], or Brin and Stuck [BS02].

Advances in methodology and the increasing availability of computational power allow practitioners to approximate dynamical systems of increasing complexity. There are two concepts in particular that have changed dynamical system simulation in recent years: *machine learning* and *co-simulation*.

1.2 Machine Learning for Simulation

Driven by the increase in computational power, the wide availability of training data, and the development of open-source frameworks, machine learning (ML) has seen significant advances in recent years. ML has a wide range of applications in many domains [SS18]. Large companies such as Google, Facebook, or Amazon [Mar19] continue to expand their use of ML methods, accelerating the development. Besides its apparent appeal for commercial applications, ML has also received interest from physical sciences [CCC+19], engineering mathematics, and mathematical physics [BK22]. Theory-agnostic ML methods have become a viable alternative to traditional analytical or numerical approaches for complex, physical, and mathematical problems [CCC+19; Sch21; KAF+17].

In the field of building simulation, we can use ML methods to solve a variety of tasks. Consider a model of the HVAC system: a numerical model requires detailed information about the characteristics and layout of the building, the size, and characteristics of windows and doors, the placement of vents, ventilators, and radiators, as well as their characteristics and states. Even if we have access to all the necessary information, it can be challenging and time-consuming to derive an accurate, numerical model of the system [ZM12]. With ML, we can learn how some quantity of the system changes from observing it. We can also use ML models to simulate non-deterministic influences, which is particularly useful in residential buildings or offices where occupant behavior governs energy and water consumption. We refer the reader to [AS22] for an overview of ML applications for *smart* buildings, [CE19] for an analysis of ML-based building performance simulation, and [ZM12; MNG+14; RDH+17; AE18; SRG+18; WZS+18; FSF+20] or [SHF20b] for reviews on ML methods for building energy prediction.

1.2.1 Limitations

The success of ML models depends primarily on data quality and quantity. While many commercial applications, such as natural language processing, can draw from billions of labeled data samples, scientific applications repeatedly suffer from a lack of labeled data points [KAF+17], often forcing models to find non-linear mappings between high-dimensional input/output pairs from partial data [RPK19]. As a result, the models may correctly replicate relationships in training and test data but generalize poorly, even if we apply cross-validation.

The second concern with traditional ML methods is that they are black boxes, hiding any causal relationships between their input and output [Rud19]. This lack of transparency

1 Introduction

prevents practitioners from understanding how a model takes a decision, leaving them with uninterpretable models [RBD+20]. Many experts agree that we cannot use uninterpretable models in high-risk applications [KAF+17; vMB+21; RBD+20; Rud19; ACB21] or subsequent scientific development.

1.2.2 Domain Knowledge

An interpretable model is constrained such that it obeys the structural or physical constraints of the domain [Rud19]. The emerging paradigms of *informed ML* [vMB+21], *theory-guided data science* [KAF+17] or *physics-informed neural networks* [RPK19], aim to address the inherent shortcomings of traditional black-box models and produce *trustworthy artificial intelligence*.

While incorporating domain knowledge into ML is not an entirely new concept, there is a rising interest in more advanced methods to integrate formal knowledge into ML pipelines. As a result, there is a significant number of publications introducing novel methods for domain-knowledge integration with inconsistent terminology [vMB+21].

1.3 Co-Simulation

Cyber-physical systems with increasing numbers of heterogeneous, interacting entities introduce new challenges for simulation [Lee08]. The introduction of networkable micro-controllers into physical processes has propelled the complexity of distributed, integrated systems. The result is a growing number of *systems of cyber-physical systems*, where embedded systems are connected using network technology [FPL14]. The inherent heterogeneity of physical processes, software components, and control logic poses challenges to the development of unified, monolithic simulation models.

The paradigm of co-simulation addresses the issue of system-of-systems simulation in a divide-and-conquer approach and combines partial models in a single simulation. The key challenge with co-simulation, however, is the integration of partial black-box solutions, their notations, and toolchains [GTB+19]. A popular solution to enable communication between partial simulators is to have them implement a co-simulation standard, such as the *functional mockup interface* (FMI) [BOA+11]. To create simulators implementing the FMI, so-called *functional mockup units* (FMUs), many practitioners use FMI-enabled modeling tools. However, to implement specialized FMUs, or FMUs based on models written in interpreted languages such as Python, it is necessary to build them manually. Hence, the many pitfalls of this process and the crucial, detailed understanding of the FMI standard prevent practitioners from fully embracing the co-simulation paradigm in many applications.

2 Objectives, Contributions and Structure

In this chapter, we introduce the reader to the research objectives of this thesis, highlight the resulting contributions, and outline its structure. Section 2.1 establishes the overarching objective for this thesis and introduces six concrete research objectives. Section 2.2 highlights the main contributions of the thesis and presents an overview of the papers it is based on. In Section 2.3, we explain how this thesis is structured and where to find the related work and methodology for the content.

2.1 Research Objectives

In this thesis, we describe the application of ML methods for modeling dynamical processes in cyber-physical systems and address the challenges outlined in the previous sections. While we can apply many concepts described in this thesis to other cyber-physical systems in science and industry, the main focus is the application of ML-based simulation models in building simulation. Thus, we formalize the overall research objective as follows:

RO₀ *Apply state-of-the-art machine learning methods to the simulation of dynamical systems in buildings.*

We separate this multifaceted objective into four thematic chapters: (i) Traditional ML Methods for Dynamical System Simulation, (ii) Theory-Informed Deep Learning for Dynamical System Simulation, (iii) Co-Simulation, and (iv) End-to-End Integration. In the remainder of this section, we present the research objectives for each chapter.

Traditional ML Methods for Dynamical System Simulation

The first objective is to apply established ML methods to simulate dynamical processes in buildings. As a starting point, we formalize the dynamical processes as regression problems and establish a distinction between prediction-based approaches and forecasts. From this formalization, we apply prediction and forecasting methods to the problem and investigate and compare their performance. We formulate the corresponding research objectives as follows:

RO₁ *Apply machine learning methods to predict the behavior of dynamical systems in buildings.*

RO₂ *Compare prediction and forecasting approaches.*

Theory-Informed Deep Learning for Dynamical System Simulation

The non-linear behavior of many building subsystems, data scarcity, and the inherent lack of transparency in the decision process of black-box ML methods motivates the second thematic chapter, where we investigate novel theory-informed neural network architectures. In the literature, we find a plethora of approaches that emerged from the intersection between deep learning, numerical simulation, and physics, many of which are very difficult to digest. To alleviate the confusion and to find common mechanisms, we survey the literature and apply the methods to a case study example. From the results, we categorize the approaches in a high-level taxonomy. The corresponding research objectives are:

- RO₃** *Survey theory-informed neural-network architectures for the simulation of dynamical systems and apply them to a case study example.*
- RO₄** *Identify common mechanisms in theory-informed neural-network architectures and derive a taxonomy.*

Co-Simulation

Co-simulation is an essential instrument for the simulation of complex cyber-physical systems. Co-simulation standards are a popular method to enable communication between partial simulators. However, many co-simulation standards and tools require tight coupling between the implementation and the interface. While many numerical and physical simulation frameworks provide out-of-the-box support for popular co-simulation standards, this is not true for ML frameworks. Hence, to generate standard-compliant simulators from ML models, practitioners have to handle the intricacies of packaging and cross-platform compilation. The required detailed understanding of co-simulation standards and the need to pre-compile code impede the introduction of ML into the co-simulation ecosystem. To alleviate this issue and to provide practitioners with a means to integrate ML models into co-simulations, we formulate the following objective for the third thematic chapter:

- RO₅** *Develop a framework that generates co-simulation-ready software components from arbitrary models.*

End-to-End Integration

Given the methods to develop ML-based simulation models and the framework to integrate them into larger, more complex co-simulations, we arrive at the final challenge: end-to-end integration of ML-based simulation methods into cyber-physical systems. To fully leverage the potential of ML in cyber-physical systems, we need to be able to embed the

models within data collection and preprocessing, training and validation, and deployment. We hence formulate the following research objective:

RO₆ *Develop a scaleable, modular software platform that facilitates data collection and simulator integration.*

2.2 Contributions

The main contribution of this thesis is to take a holistic view of the application of ML for the simulation of dynamical processes in cyber-physical systems. The subsequent listing provides the reader with an overview of the main contributions to the field of ML-based dynamical system simulation.

- An open-source ML framework for dynamical system simulation. It is available at: <https://github.com/tug-cps/datamodels>. The framework allows practitioners to implement prediction and forecasting approaches for dynamical processes.
- A comprehensive analysis of neural network architectures for dynamical system simulation. We introduce a high-level model taxonomy and provide a structured analysis of how to incorporate theoretical knowledge, external input, and uncertainty.
- A sandbox providing a practical description of how to implement theory-informed neural network dynamical system simulation. It is available at: https://github.com/cleggaard/deep_learning_for_dynamical_systems.
- A software stack that generates portable, cross-platform simulators from arbitrary simulation models. The tool cli-tool UniFMU is available at: <https://github.com/INTO-CPS-Association/unifmu>. A case study example on how to package runtime dependencies is available at: https://github.com/Daniella1/robot_unifmu. Additional educational examples are available at: https://github.com/INTO-CPS-Association/unifmu_examples.
- A set of modular components for scaleable IoT platforms. We designed the components based on an extensive literature research and an expert survey. Communication between components, data structures, and knowledge representation conform to the standards introduced and advocated by the European Commission. The components and their source code are available at: <https://github.com/tug-cps/inframonitor-public>.

2.2.1 Publications and Collaborations

This thesis is based on the contributions to peer-reviewed journals and conferences in the subsequent listing. I organized the papers according to the research objectives formulated in the previous section. The listing provides a high-level description of the papers and details my contributions.

Traditional ML Methods for Dynamical System Simulation

1. **[SSP+20]**: A conference paper describing the application of ML-based prediction models for water supply supervision in buildings. I developed the conceptual idea, researched related work, and implemented the Python framework and the models. I contributed most of the writing and handled final editing and proofreading. This paper achieves **RO₁**.
2. **[SEL+21]**: A conference paper describing and benchmarking different ML methods for energy demand estimation. The paper compares prediction and forecasting methods using different sets of features. I co-developed the conceptual idea, researched related work, and coordinated and contributed to the software development tasks to extend the framework from [SSP+20]. I contributed most of the writing and handled final editing and proofreading. This paper achieves **RO₂**.

Theory-Informed Deep Learning for Dynamical System Simulation

3. **[LSS+22]**: A high-impact journal paper providing a comprehensive overview of theory-informed neural network architectures for dynamical system simulation. For this paper, I co-developed the conceptual idea and contributed to the literature research and the implementation. I contributed several paragraphs to all chapters of the paper and worked on final editing and proofreading. This paper achieves **RO₃** and **RO₄**.

Co-Simulation

4. **[LTS+21a]**: A conference paper introducing the co-simulation tool *UniFMU*. I contributed to the implementation of UniFMU's automated build process, cross-platform invocation, its command line interface (CLI), and the FMI method wrappers. For the paper, I provided content on the virtualization extension and other technical details on the implementation. This paper provides the foundation to achieve **RO₅**.

5. [SMT+21]: A conference paper that introduces virtualization to UniFMU. I developed the idea of using the *Docker* virtualization engine to package the model's runtime requirements and co-implemented the extension. I researched related work, contributed most of the writing, and worked on final editing and proofreading. This paper refines the software to achieve **RO₅**.
6. [ASF+23]: A journal paper presenting a taxonomic review on co-simulation for buildings and smart systems. I contributed to the literature review, helped with analyzing the results, contributed content to the introduction, and worked on proofreading.
7. As a side contribution, I provided software development support for a case study on coupling physical models with ML models in a co-simulation [BSQ+21] and a case study on component exchange during co-simulation [WFA+22].

End-to-End Integration

8. [ASK+22]: A journal paper presenting the results from an empirical expert survey on the state of the art in IoT middleware platforms for smart energy systems. I worked on the design of the expert survey, researched IoT technologies and middleware solutions in the literature, and analyzed the survey results. I contributed several paragraphs to all chapters and worked on final editing and proofreading.
9. [SAH+22]: A conference paper describing the development of a modular, scalable IoT middleware platform. I designed the platform based on a literature review and the results from [ASK+22]. I coordinated and contributed to the software development process. I contributed most of the writing and handled final editing and proofreading. This paper achieves **RO₆**.
10. Side contributions to this topic include (i) a supporting role analyzing the results from an empirical survey on data availability and data enrichment in the urban energy simulations [SEM+21], and (ii) the development of a preliminary position paper on the potential of active user participation for energy demand forecasting [SCS+20].

2.3 Structuring Content, Related Work and Methodology

This thesis is organized according to the four thematic chapters introduced in Section 2.1. In Chapter 3, we discuss ML-based simulation of dynamical systems in buildings in two case studies. We establish a distinction between prediction and forecasting and discuss the results. In Chapter 4, we present a comprehensive analysis of theory-informed neural network architectures for dynamical system simulation. We showcase the application of these architectures in a case study example and present a high-level taxonomy. In

2 Objectives, Contributions and Structure

Chapter 5, we describe the integration of partial ML-based simulation models into the co-simulation ecosystem. We introduce an open-source software stack that packages any simulation model such that it complies with the *functional mockup interface* co-simulation standard. In Chapter 6, we address the issue of data collection and the integration of simulation models in real-world cyber-physical systems. We discuss the challenges of real-time communication between networkable devices in cyber-physical systems and present a showcase example of a modular, scaleable IoT platform.

2.3.1 Related Work and Methodology

We derive the four thematic aspects of the overall research objective from the developments in the literature, problems encountered in the research projects we work on, and the collaboration with experts in the field. Searching the literature for related keywords, such as *machine learning for building simulation* or *machine learning for dynamical systems*, yields a plethora of papers and books. Most relevant works address a subset of the topics entailed by the four thematic aspects we derived in Section 2.1; [RDK+19] presents a high-level outline of how to use ML to tackle climate change. [AS22] provides an overview of ML applications for smart buildings, [MNG+14; RDH+17; SRG+18] describe estimating energy consumption in buildings, and [WE19; KP20; RTB12] discuss ML-based surrogate modeling. [RGJ+22] introduces the *JuliaSim* environment, a framework of acausal, algebraic models and ML surrogates. [GTB+19] provides a comprehensive introduction to co-simulation and its challenges, [CLB+19; BLT+12] and [FGL+15] discuss hybrid, co-simulation methods for cyber-physical systems engineering. [RMP+16; NGM+16] and [dRA+18] discuss the requirements and challenges for IoT infrastructure. Because of the specific requirements for each thematic aspect, we describe the detailed methodology and provide more references to related work within the respective chapters.

3 Traditional ML Methods for Dynamical System Simulation

With the increasing availability of data and the computational power to process it, we can use ML models in a growing number of contexts. Because of the theory-agnostic, black-box nature of ML methods, users can easily infer relationships between data points without prior knowledge. In building simulation, for instance, ML models can learn the behavior of a building subsystem, such as the energy or water supply, from operational sensor data. We can use behavioral black box models to automate building operation, increase energy efficiency and flexibility, improve comfort, and simplify maintenance.

While there are many applications for both classification and regression in the building context, the primary focus of this chapter is to solve regression problems constituted by dynamical processes in buildings. This chapter is based on the work in [SSP+20] and [SEL+21] and describes two case study examples where we use ML to model the energy and water supply of a building.

3.1 Prediction and Forecasting

There are two principal strategies for formalizing the regression problem constituted by a dynamical system. While there is no consistent terminology in the literature, there is a tendency to dub them *prediction* and *forecasting*. Even if it is usually possible to use the same regression algorithms for both approaches, they are fundamentally different problems from a conceptual perspective. In the scope of dynamical system analysis *prediction* refers to estimating one state variable based on the values of the other state variables at one point in time. Consider predicting the expected energy demand of a building from the time of day, the weekday and the number of occupants in the building, for example. *Forecasting* refers to estimating the future value of a state variable based on how it has changed in the past. Consider forecasting the energy demand of a building for the next hour based on the energy demand measurements from the past six hours.

Of course, it is possible to combine the two approaches, for instance, by forecasting the behavior of one state variable based on the past behavior of the variable and the past behavior of other state variables. An example would be to estimate the building energy demand for the next hour from how the energy consumption and the number of occupants have changed in the past six hours. Both modeling approaches can estimate a single quantity or multiple quantities in a single inference step. Consequently, a model

3 Traditional ML Methods for Dynamical System Simulation

can calculate the value of one state variable for one time instance, the values of multiple state variables for one time instance, the values of multiple state variables for multiple time instances in one step. While it is not always possible to refer to the concepts and approaches described in other works using this terminology, this thesis will try to use the terms *prediction* and *forecasting* as consistently as possible.

Mathematically, we can describe forecasting and prediction models by their input and output vectors. Equation 3.1 describes the input and output to a *prediction* model. The input vector \mathbf{x} contains a set of m state variables, recorded at time t , and the corresponding output is the value of a different variable x_{m+1} at the same point in time.

$$\mathbf{x} = [x_0^t \quad \dots \quad x_m^t], \quad y = x_{m+1}^t \quad (3.1)$$

Equation 3.2 formalizes the input and output for a simple *forecasting* model. The input vector \mathbf{x} contains a series of values for the state variable x captured at multiple, discrete points in time. The parameter l can be regarded as the lookback, specifying the number of past values used in the input. The forecasting model outputs the estimated value of the state variable x at a point in the future. The time index of the output value is denoted by $t + p$, where t is the current point in time coinciding with the most recent value in the input vectors, and p is some offset into the future.

$$\mathbf{x} = \begin{bmatrix} x^{t-l} \\ \vdots \\ x^t \end{bmatrix}, \quad y = x^{t+p} \quad (3.2)$$

Equation 3.3 shows a way to combine *forecasting* and *prediction*. The input \mathbf{X} is extended into a matrix that holds a set of state variables (along its columns), sampled at multiple points in time (represented along the rows). The output is a single state variable offset into the future by p .

$$\mathbf{X} = \begin{bmatrix} x_0^{t-l} & \dots & x_m^{t-l} \\ \vdots & \ddots & \vdots \\ x_0^t & \dots & x_m^t \end{bmatrix}, \quad y = x_k^{t+p} \quad (3.3)$$

Be aware that the output observation x_k may or may not be part of the input matrix \mathbf{X} . Correspondingly, this affects whether the value 0 is a meaningful choice for p . If x_k is part of the input, its value for time step t must be known, hence predicting it for $p = 1$, is unnecessary. Conversely, if x_k is not part of the input, setting $p = 0$ allows us to predict some current observation x_k^t from the pattern in which other observations have changed up to the current point in time.

3.2 Algorithms

Practitioners can choose from a plethora of models when they want to solve a regression problem, ranging from simple linear predictors through support vector machines and decision tree algorithms to arbitrarily complex, non-linear neural network architectures.

Linear regression models can be used to solve a wide variety of linear and non-linear problems [Bon17]. Linear regression is a surprisingly capable tool that is understandable, requires little parameter tweaking, and does not overfit easily [Ray19]. However, in many real-world applications, especially when the data is highly non-linear, linear regression oversimplifies the problem.

Decision tree induction is commonly used in data mining to obtain a predictive model from observations. While intuitively suited to classification problems, decision trees can solve regression problems. Decision trees are (i) computationally inexpensive because they automatically select features that convey the most information, (ii) easily interpretable at each decision level, and (iii) capable of handling non-linear relationships in the data [XWV+05]. However, decision trees are prone to overfitting on the training data, a lack of generalization capability [Ho95; XWV+05], and sampling errors that lead to locally optimal solutions [XWV+05]. Constructing multiple trees in independent, random subspaces of the feature space expands the ensemble's capacity and addresses the generalization issues [LWZ12; Ho95].

Neural networks are used to solve regression problems throughout a variety of fields. Neural networks are known to provide excellent generalization performance and high processing speed [Bis94]. However, performance is sensitive to the availability and fidelity of data, the choice of architecture, and hyperparameters. Selecting a suitable architecture, preprocessing, training, and validation can be time-consuming. Deep learning is an extensive research field that spans a variety of domains. Hence, for a broad introduction, we refer the reader to [Bis94] or [GBC+16].

Deciding on a suitable approach or a particular application depends on a variety of factors, such as data availability, availability of computation power, requirements on accuracy and/or generalization performance, robustness against outliers, etc. In this chapter, we investigate the application of popular regression algorithms to the simulation of dynamical systems. However, we first need to discuss methods to assess the performance of regression models.

3.3 Performance Metrics

Performance evaluation in ML is an extensive research topic in its own right. It is common to assess the performance of regression models by comparing the model output \hat{y} to some target value y . This approach presumes the existence of some distance metric.

3 Traditional ML Methods for Dynamical System Simulation

Two characteristic distance functions are the linear L_1 and quadratic L_2 norm used in the mean average (MAE) and the mean squared error (MSE), respectively.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i| \quad (3.4)$$

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (3.5)$$

The latter has been adapted into the root mean squared error (rMSE), transforming the metric's value range back into the dimension of the target values [CWJ21].

$$\text{rMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2} \quad (3.6)$$

The rMSE and the MAE are both popular in a variety of fields [Bot19; CD14]. [WM05] and [CD14] have discussed the suitability and reliability of either metric. As a rule of thumb, the MSE and its derivative, the rMSE, are more sensitive to outliers than the MAE [CWJ21]. Note that scape-dependent [HK06] metrics, like the MAE and the rMSE, express the error in the same physical dimensionality as the quantity of interest and cannot be used to compare datasets with different scales. Even though the MSE is in the squared domain of the quantity of interest, it is typically considered a scale-dependent metric, and the same considerations apply. The CV-rMSE is a scale-independent version derived from the rMSE. It expresses the error as a ratio between the rMSE and the mean of the true distribution.

$$\text{CV-rMSE} = \frac{\sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2}}{\bar{y}}, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (3.7)$$

The mean absolute percentage error (MAPE) is an extension to the MAE and reports the relative variation of the error, i.e., the ratio between the error and the actual value of the quantity of interest, rather than the absolute error value [dGL+16]. In contrast to the MAE and rMSE, the MAPE is scale-independent and can be used to compare the performance of models predicting values of different magnitudes.

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{\hat{y}_i - y_i}{y_i} \right| \quad (3.8)$$

However, the MAPE has an inherent bias toward emphasizing positive errors [SBS+13]. The MAPE is undefined for target values around 0 and excessively large for target values close to 0, so it is not suitable for datasets where 0 is a meaningful value, e.g., predicting the temperature in $^{\circ}\text{C}$ [HK06]. Despite these issues, the MAPE is a very popular metric [Bot19], often used because it provides an intuitive understanding of the relative error [dGL+16].

Note that the MAE, the MSE and even the scale-independent MAPE have no upper limit, ranging from 0, implying a perfect fit, to infinity, with their actual values depending heavily on the domain and the distribution of the quantity of interest [CWJ21]. Over the years, there has been an effort to mitigate some of the problems associated with the MAPE, including its asymmetry and its unboundedness [Mea86; Mak93; MH00; HK06; CTG17]. However, there is still some disagreement on the optimal mathematical formulation of this symmetric mean absolute percentage error (SMAPE) [CWJ21]. The original definition [Mea86] bounds the error between 0, the perfect fit, and 2, where all model outputs and target values are of opposite sign.

$$\text{SMAPE} = \frac{1}{n} \sum_{i=1}^n \frac{|\hat{y}_i - y_i|}{(|\hat{y}_i| + |y_i|)/2} \quad (3.9)$$

While addressing some of the issues with the MAPE, the SMAPE similarly disregards the distribution of the target values, causing reliability problems, especially with identifying regression models that perform poorly [CWJ21].

The coefficient of determination (R^2), formulated in 1921 [Wri21] quantifies the proportion of the variation in a dependent variable that is predictable from a (set of) independent variable(s). It can be expressed through the ratio between the sum of square residuals and the total sum of squares of the target dataset. This is equivalent to the ratio between the MSE and the variance of the target values.

$$R^2 = 1 - \frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{\sum_{i=1}^n (\bar{y} - y_i)^2}, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (3.10)$$

The R^2 has an upper bound of 1, where 1 indicates a perfect fit. Theoretically, R^2 has no lower bound. However, a value of 0 corresponds to the fit of a horizontal line at the mean of the target value distribution, so disregarding negative values does not diminish the informativeness of the R^2 [CWJ21].

The MAE, the MAPE, the MSE, the rMSE, the CV-rMSE and the R^2 are commonly used to assess model performance in data-driven building energy prediction [SHF20b; MNG+14; SRG+18].

3.4 Case Studies at the Graz University of Technology

In the subsequent sections, we describe two case studies that apply prediction and forecasting approaches in a real-world application. The first application is a water supply supervision pipeline developed for the Graz University of Technology (TUG). The objective here was to implement an automated, data-driven supervision system based on historical water consumption data and investigate the capabilities and limitations of prediction methods.

The second application is to forecast the electrical energy demand on a building level. The main objective of this application was to assess the performance of four ML algorithms (linear regression, random forest, fully-connected neural network, and recurrent neural network) commonly used for forecasting with various sets of input features. Additionally, we investigated how well these models adjust to changing demand patterns by assessing their performance on a dataset obtained during the initial wave of the Covid-19 pandemic.

3.4.1 Background

The buildings and technical support department (BATS) at the Graz University of Technology monitors and maintains infrastructure across three campuses with a cumulative floor space of 240,000 m² [Tec21]. BATS operators manage electrical energy supply, HVAC, water supply and drainage, as well as cleaning, maintenance, modernization and renovation of buildings and facilities housing offices, lecture halls, laboratories, social spaces and canteens.

Most supply systems are connected to digital metering devices, producing large amounts of operational data, including water consumption and energy demand on a building level, captured at 15-min and 1-hour intervals, respectively. Subsequently, we discuss the application of prediction and forecasting methods to the issue of water supply supervision and energy demand estimation.

3.4.2 Implementation

We implemented all models and the data-handling framework in *Python 3*, employing the *TensorFlow* [MAP+15] implementation of the *Keras* [Cho+15] API for the neural networks and the *scikit-learn* [PVG+11] implementation of the statistical regression models. For data handling and transformation we used the data analytics and manipulation library *pandas* [tea20] and the array programming library *numpy* [HMW+20]. The framework we developed is available on GitHub¹, and includes functionalities for or data preparation, preprocessing, training, benchmarking and plotting.

3.4.3 Water Supply Supervision

The World Economic Forum lists water crises among the top global risks in terms of the potential impact on society [Wor19]. A study considering inter-annual variations in water consumption and availability finds that more than four billion people experience severe water shortages for at least one month per year [MH16]. A shortage of unpolluted, drinkable water threatens biodiversity and the livelihood of entire communities; with the rising water demand likely exceeding the impact of greenhouse warming on the health of

¹<https://github.com/tug-cps/datamodels>

water systems [VGS+00]. Water consumption habits play a vital role in managing water scarcity. One step can be an efficient means to detect and mitigate leakages.

Building complexes are streaked with water pipes and sewers. In structures with little to no occupancy, such as offices, lecture halls or laboratories during weekends and holidays, leakages from broken pipes may go undetected for days. This entails two problems: (i) water is wasted and (ii) the spillage can damage expensive equipment. Because of the complexity of the water supply system in intertwined urban areas and campuses, building operators, such as the BATS department, cannot monitor the entire system manually, let alone ensure 24/7 coverage.

Retrofitting an existing supply system with sensors to measure pressures and throughput is costly. Developing physical models to compare expected and actual observations of the system state requires detailed information on the system’s components and interconnections, which is hardly ever available. However, water leakages cause irregularities in the consumption patterns that can be detected using ML approaches [KPL+18; JBS+19].

3.4.3.1 Data

The water consumption data provided by the BATS operators is sampled at 15-minute intervals and marked with a date and a timestamp. The training data does not provide information on whether an entry was captured during nominal or faulty system behavior. Consequently, tagging the values would require human input. However, talking to the BATS operators revealed that for the training data we were provided with, there was only one instance of a known leakage between three years of nominal behavior. Considering this lack of data representing faulty behavior, we decided to develop a regression model predicting the expected nominal consumption and subsequently compare the output of the regression model against the actual consumption value to detect irregular behavior.

3.4.3.2 Preprocessing

We downsampled the dataset from a 15-minute frequency to hourly intervals to reduce the influence of short-term peaks on the overall consumption pattern. The choice of one-hour bins was governed mainly by visual inspection of the variances in the dataset and the objective to find a balance between fast, real-time feedback in the supervision pipeline and sufficient prediction accuracy. As the objective was to train the models to predict the nominal behavior of the water supply system, all irregular data had to be removed. There are many approaches to detecting outliers, such as numeric wrapper methods or Voronoi algorithms; however, with access to expert input, we decided to have the experts manually identify the readings captured during a leakage or special public events. We removed the entries from the training and test data and subsequently used the readings from the leakage to verify that the models predict nominal values that are sufficiently different from those captured during a leakage.

3.4.3.3 Feature Construction and Selection

In a setting such as the one described here, where we are provided with time series data and no additional features, it is typically convenient to formalize the regression problem using a forecasting approach. However, we decided to use a prediction approach for two reasons: (i) since the objective was to predict an expected nominal baseline consumption rather than the actual consumption, we did not want the model to follow the consumption trajectory if it diverges from its regular pattern (e.g. during a leakage), and (ii) we wanted to examine the capabilities of prediction approaches for time series data, and in particular, the use of static input features that can be calculated for arbitrary points in the future, such as the time of day.

The water consumption data supplied by the BATS operators provides two features, the time of the day and the date. Preliminary experiments showed that using both the date and the time of the entries caused the models to overfit on the date and that using the time alone did not provide the model with sufficient information to produce satisfactory results. It became apparent that it was necessary to construct additional features from the data.

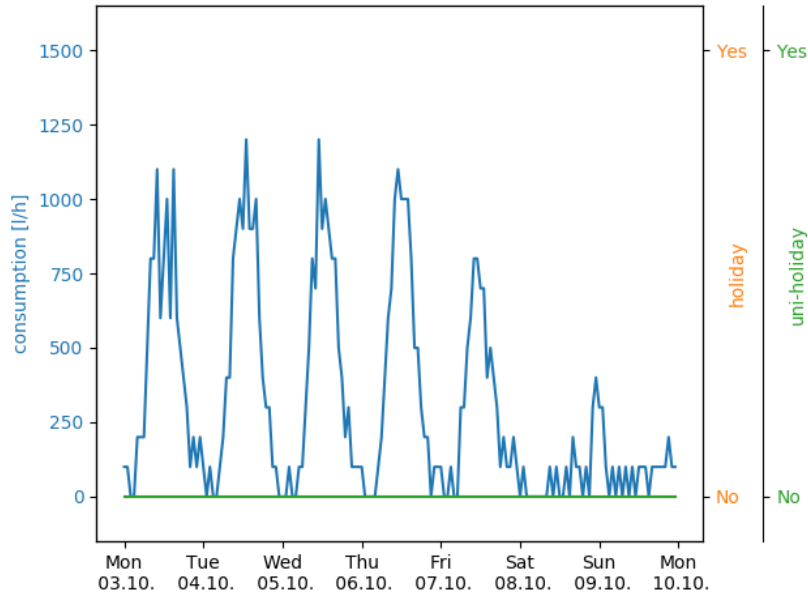
There are various approaches to feature engineering, including feature synthesis [KV15] and feature generation [GK96] algorithms, as well as feature generation methods using ML [KSS16a]. However, while often applied successfully, the selection process and the reasoning applied in such sophisticated methods can be difficult to comprehend and interpret for practitioners. So, to obtain a transparent feature set, we decided to apply human intuition instead.

Figure 3.1a shows the water consumption in liters per hour for a week in October 2016. To the human observer, it is apparent that there is a recurrent daily pattern whose amplitude varies depending on the day of the week. Accordingly, in addition to the *timeframe*, encoding the time of the day, we chose a numeric representation of the *weekday*, running from 0 to 6, as an input feature. Figure 3.1b shows the influence of a public holiday on water consumption. Considering how significantly lower the consumption is on days coinciding with public holidays, it stood to reason to add a binary feature (*holiday*) to encode the occurrence of public holidays. The weekly patterns are mostly consistent throughout the year. However, as shown in Figure 3.2, the amplitude is consistently higher by approximately 200 to 400 liters per hour around peak times when classes are held. We use *uni_holiday* as a binary feature to encode if a measurement was recorded during or outside the semester.

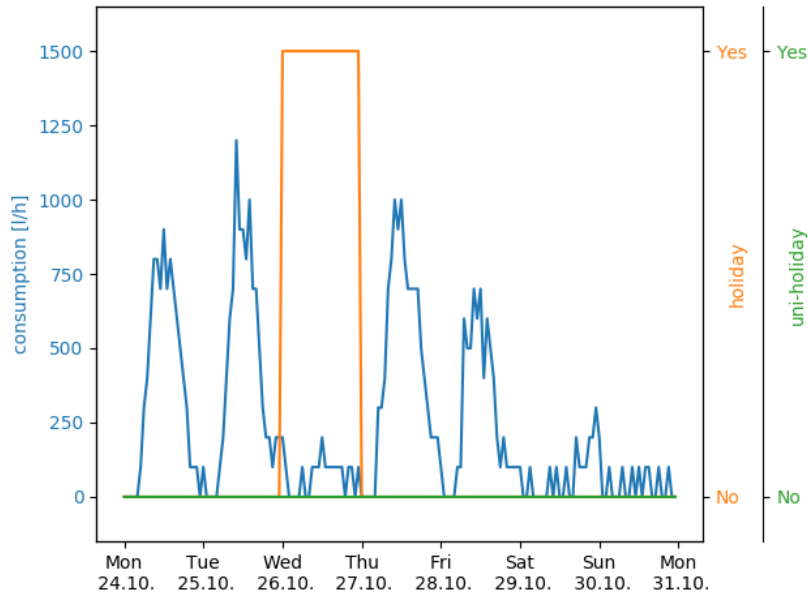
3.4.3.4 Feature Encoding

The features *timeframe* and *weekday* assume recurrent values; 0 to 23 (because consumption is sampled hourly) and 0 to 6, respectively. While it is obvious to a human that the time difference between weekday 6 (Sunday) and weekday 0 (Monday) is the same

3.4 Case Studies at the Graz University of Technology



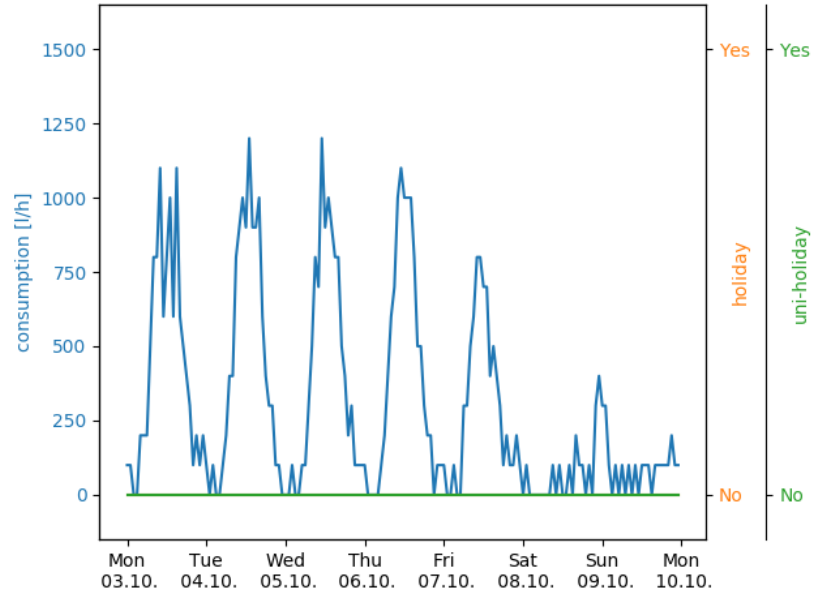
(a) Water consumption trajectory during the semester with no public holidays.



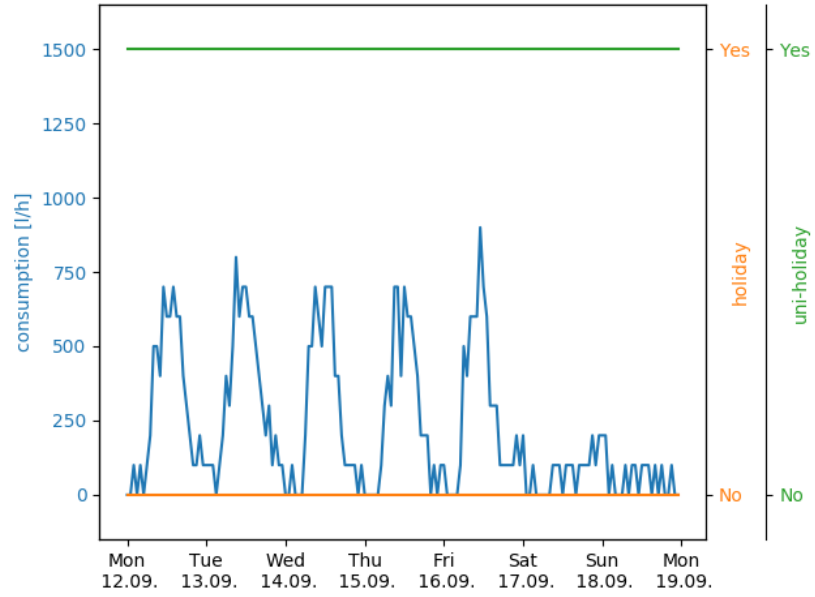
(b) Water consumption is significantly lower on the Austrian National Day.

Figure 3.1: The plot in (a) shows regular water consumption over a week in in liters per hour during a week in October 2016. (b) shows a different week in October 2016 that coincides with the Austrian National Day (October 26th) and exemplifies how public holidays affect the amplitude of the trajectory.

3 Traditional ML Methods for Dynamical System Simulation



(a) Water consumption trajectory during the semester.



(b) Water consumption during holidays.

Figure 3.2: The plots highlight the difference in amplitude depending on whether classes are held at university. Note that the consumption peaks are 200 to 400 liters per hour higher during the semester.

as the difference between weekday 2 (Wednesday) and 3 (Thursday), numerically, these differences are not the same. In order to capture the cyclic nature of the features, they were encoded with sine/cosine pairs [DR98], as seen in Equation 3.11.

$$v_{\sin}(v) = \sin\left(\frac{2\pi v}{T}\right), \quad v_{\cos}(v) = \cos\left(\frac{2\pi v}{T}\right) \quad (3.11)$$

where v is the variable value, and T is the maximum value of the variable.

The flow diagram in Figure 3.3 visualizes the feature selection, feature construction and feature encoding process that transforms a data entry into a feature vector. Note that the entities derived from the date and the time of the data entry constitute the input features \mathbf{x} and the consumption is the target value y .

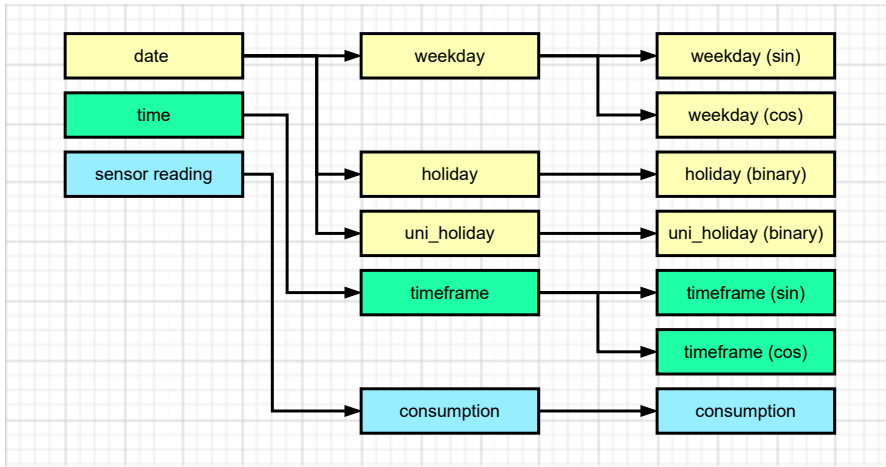


Figure 3.3: The feature selection and engineering pipeline. The diagram shows the transformation of a data entry into a feature vector. Features derived from the date and time constitute the input \mathbf{x} ; the consumption is the target y .

3.4.3.5 Algorithms

Based on our estimation of the complexity of the mapping between the input and the output of the models, we decided to test two non-linear regression algorithms: a random forest and multiple versions of fully-connected feedforward neural networks.

The **random forest ensemble** consists of ten estimators, with the mean squared error (MSE) as the splitting criterion and no limits on the maximum tree depth or the maximum number of leaf nodes. Splitting an internal node requires a minimum of two samples. Each leaf must contain at least one sample. The estimators use all input features when searching for the optimal split. The samples from the input set are bootstrapped randomly to build the estimators.

The **feedforward neural networks** consist of fully-connected intermediate layers of neurons activated with rectifier functions and a single, linear output layer. The weight kernels are initialized based on the input shape to each layer [GB10] and biases as zeros. There is no regularization of the weights, biases or outputs of the layers and no constraint functions on the weights and biases. The neural network models were trained on mini-batches using an *RMSprop* optimizer with a learning rate of 0.001, ρ and ϵ parameters of 0.9 and $1e^{-7}$ respectively. To find the most suitable architecture and training parameters we employed a hyperparameter search with two-fold cross validation using the candidate values in Table 3.1.

Table 3.1: Values used in hyperparameter search.

hyperparameter	values
intermediate layers	1, 2, 3
neurons	24, 48, 96, 128, 192, 384
epochs	24, 48, 96, 192, 384, 512
batch size	32, 64, 128, 512

3.4.3.6 Metrics

To assess and compare the performance of the models, we used the rMSE and the R^2 . Both metrics are frequently used to evaluate models solving regression problems in the building domain [RJB+19; SHF20a]. We compared predictions to actual readings for nominal and faulty system behavior, defining high accuracy on regular data and low accuracy on irregular data as the target.

3.4.3.7 Training, Test and Validation Data

The regression model in the supervision pipeline is trained on historical data. Subsequently, the pipeline monitors the supply system in real time. To emulate these boundary conditions, we split the data for training and validation along its time axis. The data for training and testing was recorded between August 1, 2016 and December 31, 2017. We resampled this dataset randomly and used 80% (9,697 samples) for training and 20% (2,242 samples) for testing. Note that the resulting training/test split is not chronological but random. For the hyperparameter search, we used a part of the training set as a separate validation set. To validate generalization performance, we used a dataset with consumption values recorded between January 1, 2018 and June 30, 2019 and a dataset with values captured during a water leakage between January 10, 2018 and March 7, 2018 (1,354 samples).

Table 3.2: Top-three results from hyperparameter search. R^2 was calculated on a independent validation set, generated from the training data.

layers	neurons	epochs	batch size	R^2 (split)
2	128	240	384	0.85003
2	384	80	192	0.84971
2	128	480	384	0.84965

Table 3.3: Root mean squared error (rMSE) and the R^2 performance metrics, calculated on training data, randomly resampled test data, separate validation data and data recorded during a leakage. Last row shows average training time in milliseconds.

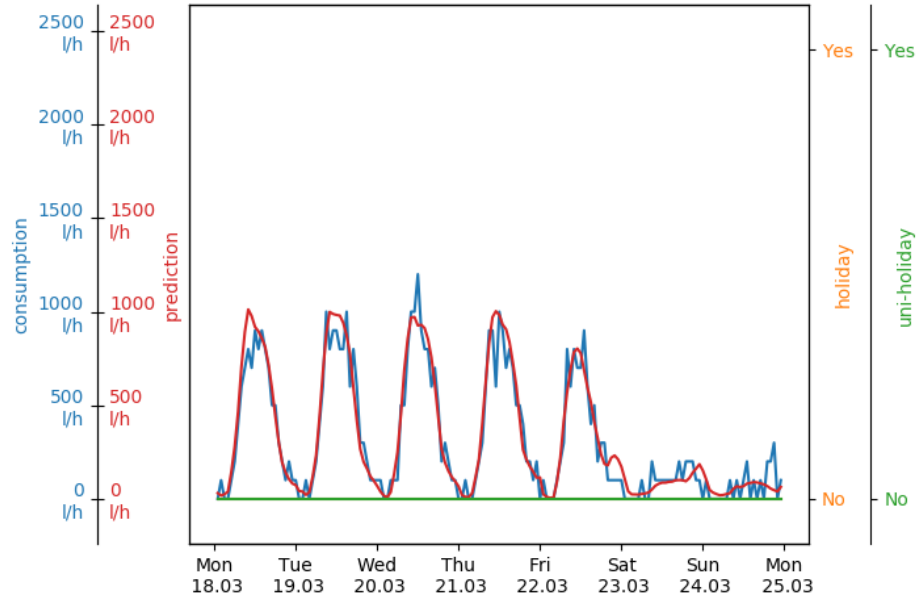
	random forest	neural network
R^2 (training)	0.8590	0.8556
R^2 (test)	0.8441	0.8500
R^2 (validation)	0.7733	0.7763
R^2 (leak)	-4.8426	-4.8404
rMSE (training)	117.52	118.92
rMSE (test)	127.31	124.87
rMSE (validation)	148.22	147.22
rMSE (leak)	777.16	777.02
average training time	500ms	15000ms

3.4.3.8 Results

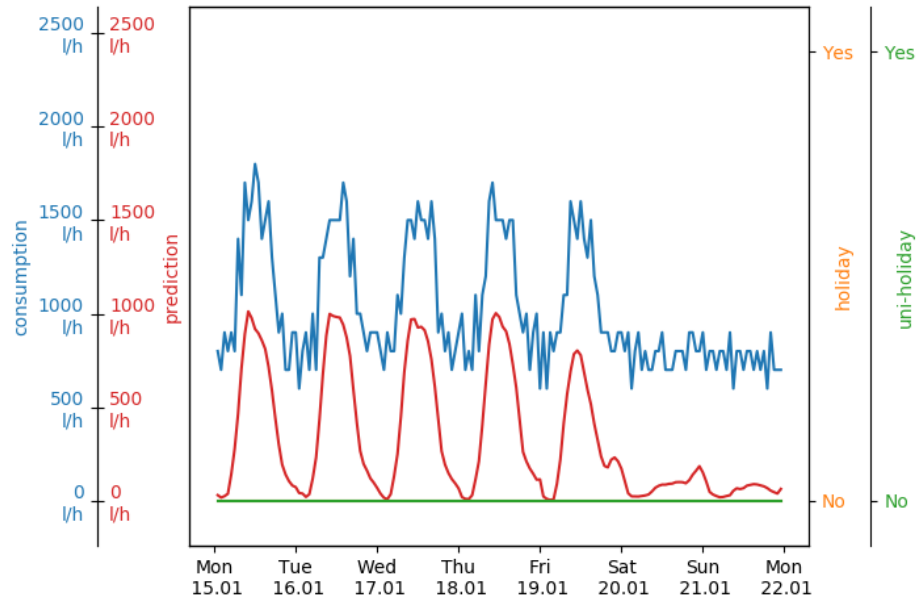
Table 3.2 shows the three parameter choices that performed best. Note that one of the network architectures appeared twice in the top three results. Consequently, we used this architecture for the subsequent experiments.

Table 3.3 contains the rMSE and the R^2 calculated on the training and test datasets, the separate validation set and the consumption trajectory recorded during the leakage at the beginning of 2018. The results show that the neural network outperforms the random forest on both the test and validation set by an extremely narrow margin. A more significant difference, however, can be seen in average training time. While training the random forest model takes around 500 milliseconds, training the neural network requires, on average, 15 seconds. Figure 3.4 shows the trajectories predicted by the neural network projected against actual consumption during regular and irregular system behavior. Note how the prediction does not follow the actual measurement during the leakage.

3 Traditional ML Methods for Dynamical System Simulation



(a) Prediction and actual measurements during regular system operation; the prediction and actual values match closely.



(b) Prediction and actual measurements during a leakage. Values constantly differ by approximately 600 l/h, which corresponds to the amount of water spilled.

Figure 3.4: The predicted and actual water consumption trajectories during nominal and faulty behavior.

Given the simplicity of the prediction approach, the results are surprisingly promising. There is, however, the potential for improvement, especially in generalization performance. Errors are consistently lower on unseen data selected from the same time frame as the training data (test data) than on unseen data from a later time (validation data). The most obvious approach is to engineer additional features from the data. For instance, the chosen features do not encode varying occupancy during semesters or represent that some holidays, such as Christmas, are less busy than others. One way to address this would be to construct a feature representing the time within the year. This feature, however, would have to be constructed carefully so that the model does not overfit on patterns in the training data. Other possibilities to improve prediction performance are to combine the weekday encoding with the holiday feature (i.e., mark holidays as Sunday), apply normalization on the input data, or explore other regression models.

3.4.4 Energy Demand Estimation

There is a large body of literature on data-driven energy estimation for buildings. Latest reviews of state-of-the-art approaches can be found in [WZS+18; SRG+18; AE18; FSF+20] and [SHF20b]. These reviews identify neural networks, regression trees, support vector machines and linear regression methods as the most popular data-driven modeling approaches. The authors of [SHF20b] present a review of 105 papers concerned with data-driven building energy estimation, addressing the research gaps in preceding literature reviews by including novel technologies (such as state-of-the-art deep learning methods), highlighting and analyzing the time-series property of energy data and providing a detailed summary of the input features used in the papers.

Most of the models in the literature use meteorological data, historical energy consumption features engineered from date and time, and occupancy as inputs to estimate the energy demand on various levels of aggregation, including single buildings, building blocks and entire districts. The applications cover a variety of building types ranging from residential structures to educational buildings, commercial offices and industrial complexes. Key objectives are estimating the total energy demand, electric power demand, or heating/cooling demand. Even when focusing on data-driven methods to estimate the electric energy demand of a single academic/mixed-use, office building, there is a substantial number of similar studies in the literature.

Deciding on a suitable algorithm for energy demand estimation can be difficult for practitioners. So we decided to develop a framework for testing and benchmarking ML models for time-series analysis and evaluate the performance of four popular algorithms for energy demand estimation. We compared the forecasting accuracy of a linear regression model, a random forest, a fully-connected, feedforward neural network, and a recurrent neural network. For each model, we used a forecasting approach to estimate a single energy demand value from a finite set of previously observed values as a baseline and tested a variety of additional input feature sets in terms of their influence on model performance.

3 Traditional ML Methods for Dynamical System Simulation

The forecasting problem can be characterized by two parameters, the lookback and the lookahead. The lookback denotes the number of past values the estimated target value is based on, and the lookahead expresses how many steps into the future the estimated target is located. In Figure 3.5 the blue graph represents the time series development.

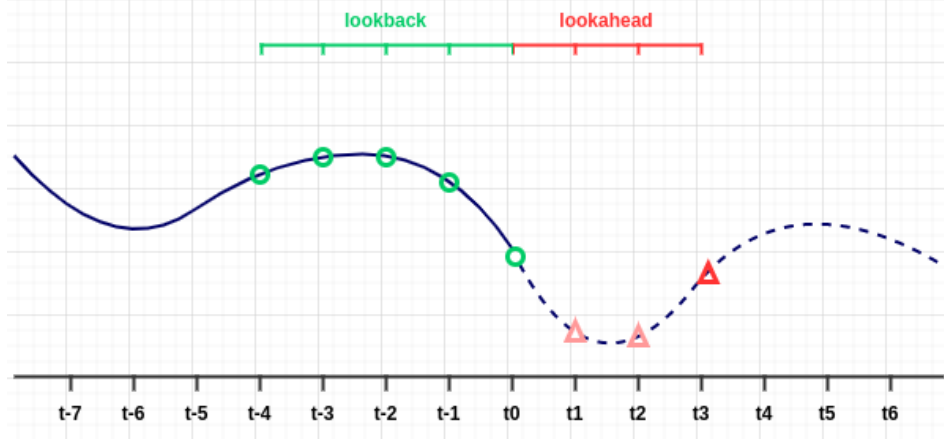


Figure 3.5: The lookback and lookahead. Note that the the model may estimate a single value at the end of the lookahead, or a series of values starting at the current time step.

For the current time t_0 , the solid line represents the past, and the dashed line represents the future. In this example, four values from the past in conjunction with the current value (marked with green circles) constitute the basis for projecting the value(s) (marked as a red triangles) of the time series for three steps into the future, which corresponds to a lookback of four and a lookahead of three. Depending on the model, intermediate values (marked with light red triangles in the plot) may or may not be part of the projection.

The baseline models infer forecasts from a single type of previously observed values, i.e., the models estimate forecasts of the hourly energy demand based on previous energy demand values. Each input vector \mathbf{x} is of size $[n, 1]$, where $n = \text{lookback } l + \text{one current state}$. The output y is a single scalar value, i.e. the expected energy demand c^i at time step $t + p$, where p is the lookahead.

$$\mathbf{x} = \begin{bmatrix} c^{t-l} \\ \vdots \\ c^t \end{bmatrix}, \quad y = c^{t+p} \quad (3.12)$$

It is possible to introduce additional information into the models by adding observations about the building's state that we suppose is correlated with the energy demand. Consider, for instance, the number of registrations for the courses held in the lecture hall located in the building, the time of day, the outside temperature, or the hourly water consumption of the building.

Using multiple characteristics to describe the system state turns the scalar components in the input vector \mathbf{x} into vectors. Consequently, the input vector turns into an input matrix \mathbf{X} of size $[n, m]$, where m corresponds to the number of characteristics, subsequently denoted features f_k^i . The model output y remains unchanged, as we are only interested in future energy demand.

$$\mathbf{X} = \begin{bmatrix} f_0^{t-l} & \cdots & f_m^{t-l} \\ \vdots & \ddots & \vdots \\ f_0^t & \cdots & f_m^t \end{bmatrix}, \quad y = c^{t+p} \quad (3.13)$$

Recurrent neural networks take input samples as two-dimensional matrices with a time axis and a feature axis. For the other models, however, this matrix has to be flattened to a vector \mathbf{x} , effectively removing any distinction between different features of the same time step and the same features of different time steps.

$$\mathbf{x} = [f_0^{t-l} \quad \cdots \quad f_m^{t-l} f_0^t \quad \cdots \quad f_m^t], \quad y = c^{t+p} \quad (3.14)$$

Be aware that the energy demand values may or may not be part of the input. In the subsequent section, we outline the data available to us.

3.4.4.1 Data

To compare and benchmark some of the most common energy demand estimation approaches, we used input features that are popular in the literature and very likely to be available to practitioners. For the baseline, we estimated future energy demand from historical energy demand data, as this is the bare minimum of data that has to be available to develop a supervised learning model.

The BATS department provided **energy demand trajectories** from smart sensors located in a five-floor mixed-use academic/office building accommodating offices, seminar rooms, laboratories and a lecture hall. The data is sampled at one-hour intervals and marked with a date and a timestamp. It was recorded between May 5, 2019, and July 21, 2020, with parts of the data being captured during the onset of the Covid-19 pandemic in Austria. Due to the measures imposed by the Austrian federal government, the university directorate restricted access to the academic facilities, suspended classroom teaching and mandated staff members to work from home whenever possible. This unusual situation allowed us to investigate how the models adjust to such significant changes in the target trajectories, without changing the boundary conditions, i.e., training and test data are from the same building, but occupancy and occupant behavior are different. We trained all models on data recorded before the measures came into effect and tested them on data from periods where the restrictions were in place.

The first logical extension was to use the data's date-and-time index to engineer a set of additional input features based on human expert input. Similar to the approach we applied for predicting the water supply trajectories in Section 3.4.3.3 we inferred an

apparent correlation between the amplitude of the energy demand, the time of day, the weekday, the occurrence of public holidays, and the semester breaks. Consequently, the **date and time features** include the time of day, the weekday, a Boolean value encoding the occurrence of public holidays, and a Boolean variable encoding semester breaks. The time of day and the numeric representation of the weekday assume periodic values, 0 to 23 (sampled hourly) and 0 to 6, respectively. To properly represent the cyclic nature of these features, they are encoded with a sine/cosine pair, as described in Section 3.4.3.4.

Meteorological conditions typically correlate with changes in the operational energy demand of a building [SHF20b; FSF+20]. Consequently, meteorological data is a popular input feature for energy demand models in the literature. Building operators and practitioners often have access to on-site weather stations, and even if they do not, weather data can be obtained from one of the many free or commercial weather APIs on the internet. We gathered the **weather data** through a free web API² that provides access to the open data collection published by the Austrian "Zentralanstalt für Meteorology und Geodynamik" (ZAMG). The weather station that recorded the data is located at the Graz Airport, about 8 km from the building site, and captures, among other metrics, hourly temperature measurements and the time of sunrise and sunset.

Literature suggests that occupant behavior is one of the governing factors in building energy demand [AHF+19; WXP+19] and that using **occupancy** as an input feature can improve the accuracy of energy estimation models [WXP+19]. The problem, however, is that accurate occupancy data can be hard to obtain, as it typically requires building operators to install and maintain additional sensors in the building and address potential privacy concerns. So instead of using actual, measured occupancy data, we used course and event schedules for the lecture hall in the building.

Finally, we decided to investigate whether using **water consumption data** as model input can help improve the accuracy of energy demand models. To the best of our knowledge, no paper in the literature has investigated the influence of this input feature before. However, we based our decision on the assumption that water consumption might implicitly encode occupancy and can thus help introduce knowledge into the model. The water consumption data for the building was provided by the BATS operators.

3.4.4.2 Algorithms

We developed four data models: two statistical regression models (linear regression and a random forest) and two neural networks (a fully-connected feedforward and a recurrent network). Subsequently, we describe the parameter settings and design choices for each model.

The **linear regression (LR)** model is relatively simple and requires no parameter settings. As the input samples have to be provided as a one-dimensional vector, multi-feature input matrices are flattened along the time axis in a preprocessing step.

²<http://at-wetter.tk>

The **random forest (RF)** consists of 100 estimators, with the mean squared error (MSE) as the splitting criterion and no limits on the maximum tree depth or the maximum number of leaf nodes. Splitting an internal node requires a minimum of two samples. Each leaf must contain at least one sample. The estimators use all input features when searching for the optimal split. The samples from the input set are bootstrapped randomly to build the estimators. The actual random forest requires the input samples to be in vector format. So for experiments with multiple input features, the input matrices are flattened into vectors along the time axis as described in Equation 3.14.

Both **neural network** models accept either a vector, with the value of one feature per lookback step, or a $[n, m]$ matrix with values of multiple features for each time step as input. Despite being capable of producing vectors or matrices containing single or multiple values for single or multiple time steps, we use the networks to forecast a single scalar energy demand value for one specific point in the future, which is the same output as the one from the statistical models.

All networks are trained on mini-batches of 72 samples using an *RMSprop* optimizer with the learning rate set to 0.001 and the ρ and ϵ parameters set to 0.9 and $1e^{-7}$ respectively. We set the maximum number of training epochs to 200 but implemented early stopping to avoid overfitting. For the early stopping mechanism, we use 20% of the samples (subsequently called the validation data) in the set to monitor convergence. Training stops when the MSE on the validation data does not decrease for 30 consecutive training epochs.

The **fully-connected neural network (NN)** architecture cannot explicitly distinguish between the feature and the time axis. So, the first network layer flattens input matrices with multiple features from multiple time steps into vectors, as described in Equation 3.14. The flattened input passes through two consecutive layers of 64 rectified linear units (ReLU) into a single output layer. The weights are initialized based on the layer's input shape [GB10] and the bias vectors set to 0. We used no regularization or constraints for the weights or biases.

The **recurrent neural network (RNN)** uses an architecture designed specifically for time series data. Consequently, the input matrices retain the explicit distinction between the feature and time axis. The inputs pass through a single layer of 32 long short-term memory units [HS97], activated with hyperbolic tangent functions, sigmoid functions for recurrent activation, and no dropout. The weights are initialized based on the shape of the input to the layer [GB10], the biases are set to 0, and the recurrent weight kernel uses an orthogonal initializer. We apply no regularization or constraints on the weights, the biases, or the recurrent weight kernel. The initial states are set to 0, and there is no dropout on the linear transformations of the input or the recurrent state. The LSTM layer outputs a single value into a dense layer with one unit.

3.4.4.3 Metrics

We use two scale-independent error metrics, the CV-rMSE and the R^2 to compare model performance. Both metrics are commonly used to assess energy demand models in the literature [SHF20b]. Additionally, the ASHRAE standards committee proposes the CV-rMSE as the de-facto standard goodness-of-fit indicator [ASH02].

3.4.4.4 Training, Test and Validation Data

We split the dataset along the time axis into a separate training and test set, using the older 80% of the data to train the models and the remaining, more recent 20% to assess generalization performance. Note that the more recent data has diverging usage patterns with readings from before and after the Covid-19 measures took effect. To monitor the convergence of the training process for the neural networks, we use a validation dataset consisting of a random subsample of 20% of the training data in each iteration.

We trained each of the four model types using six different feature set combinations: (i) energy demand, (ii) energy demand + weather data, (iii) energy demand + date and time features, (iv) energy demand + water consumption, (v) energy demand + course and event schedules and (vi) combination of all features. For each combination, we tested a lookback of 12, 48, and 72 hours to predict a single hourly energy demand value one, three, six, 12, and 24 hours ahead of the most recent value in the input vector.

3.4.4.5 Results

Results show that models trained on feature sets containing data from the course and event schedules perform poorly. It stands to reason that the number of registrations for the courses and events held in the building's lecture hall does not properly approximate the actual number of people in the building rather than the occupancy not affecting energy performance. The most likely reasons for this apparent discrepancy are that (i) attendance is not mandatory for many courses at university, (ii) single occurrences of lectures are sometimes canceled or cut short without updating the central registration system, and (iii) the restrictions imposed by the Covid-19 measures might not have been propagated to the registration system. Consequently, we omit the results from combining energy demand + course and event schedules and the combination of all available feature sets in the subsequent analysis.

Figure 3.6 shows the CV-rMSE between the forecasts and the actual energy demand trajectories from the test dataset, i.e., the more recent 20 % of the data. Each subfigure shows the results for all four model types; the results in the first row are from models that use a lookback of 24 hours, and the second row shows the CV-rMSE achieved by models using a lookback of 72 hours. The columns correspond to the four feature sets we examined in detail: historical energy demand values, energy demand + weather data,

Table 3.4: A comprehensive overview of the error between the forecast and actual energy demand trajectories from the test dataset for all models and the four most promising feature sets. All models and combinations use a lookback of 24 hours.

feature set		energy demand					energy demand, weather				
lookahead		1h	3h	6h	12h	24h	1h	3h	6h	12h	24h
LR	CV-rMSE	0.14	0.24	0.28	0.30	0.31	0.14	0.23	0.27	0.29	0.32
	R ²	0.78	0.40	0.16	0.07	-0.03	0.78	0.44	0.22	0.10	-0.04
RF	CV-rMSE	0.14	0.23	0.25	0.27	0.28	0.14	0.22	0.24	0.25	0.27
	R ²	0.78	0.44	0.33	0.24	0.16	0.79	0.47	0.39	0.32	0.22
NN	CV-rMSE	0.15	0.22	0.26	0.27	0.29	0.14	0.21	0.24	0.29	0.29
	R ²	0.77	0.51	0.30	0.25	0.13	0.79	0.55	0.41	0.11	0.15
RNN	CV-rMSE	0.15	0.24	0.28	0.29	0.28	0.14	0.22	0.25	0.30	0.29
	R ²	0.77	0.39	0.18	0.13	0.20	0.79	0.48	0.34	0.08	0.11
feature set		energy demand, date/time					energy dem., water cons.				
lookahead		1h	3h	6h	12h	24h	1h	3h	6h	12h	24h
LR	CV-rMSE	0.14	0.22	0.25	0.26	0.27	0.14	0.23	0.28	0.30	0.32
	R ²	0.79	0.47	0.32	0.28	0.27	0.79	0.43	0.20	0.08	-0.08
RF	CV-rMSE	0.14	0.19	0.21	0.20	0.23	0.15	0.23	0.25	0.24	0.29
	R ²	0.79	0.61	0.55	0.57	0.44	0.78	0.45	0.34	0.37	0.12
NN	CV-rMSE	0.14	0.22	0.25	0.27	0.28	0.14	0.21	0.25	0.27	0.29
	R ²	0.80	0.49	0.35	0.23	0.20	0.79	0.55	0.35	0.26	0.14
RNN	CV-rMSE	0.14	0.20	0.23	0.29	0.22	0.14	0.24	0.27	0.27	0.30
	R ²	0.79	0.56	0.43	0.11	0.51	0.78	0.38	0.21	0.22	0.06

3 Traditional ML Methods for Dynamical System Simulation

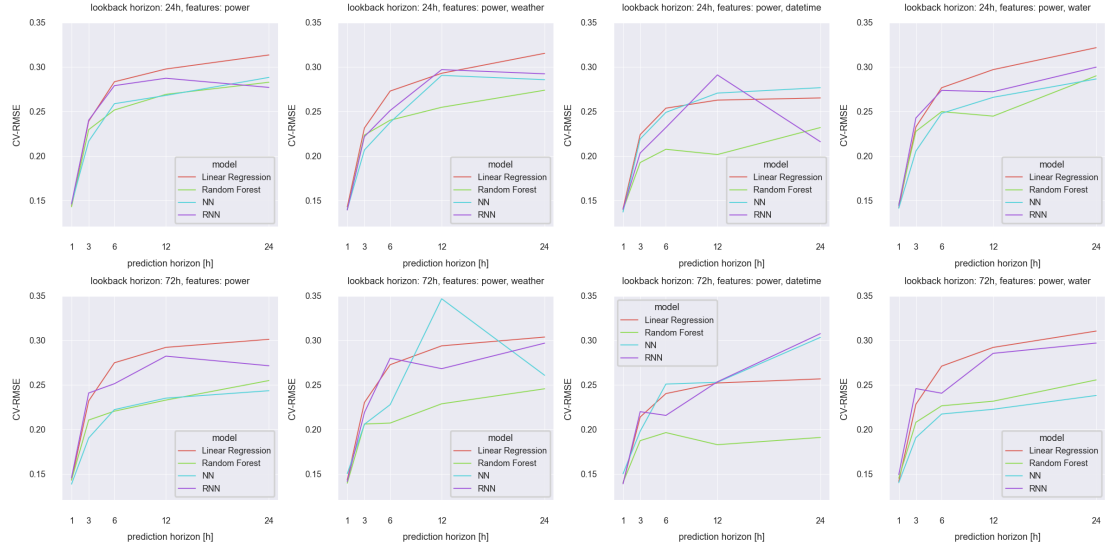


Figure 3.6: Comparison of the CV-RMSE for all model and feature combinations.

energy demand + date and time features, and energy demand + water consumption. Each subfigure shows the CV-rMSE along the y-axis for a lookahead of one, three, six, 12, and 24 hours. Table 3.4 provides a detailed listing of the results in terms of the CV-rMSE and the R^2 for models with a lookback of 24 hours.

The results show that for a lookahead of one hour, all models achieve a similar CV-rMSE of approximately 0.15, regardless of the choice of input feature set and lookback. Conversely, estimating the energy demand 24 hours into the future seems challenging for all models. There is, however, a more significant difference in performance between the models than for the one-hour-ahead forecast. For the three- to 12-hour-ahead forecast, differences in performance between the models and the feature set choices are most noticeable. Changing the lookback horizon from 24 to 72 hours slightly improves accuracy for three- to 24-hour ahead forecasts with the fully-connected neural network and the random forest when using historical energy demand alone or energy demand and water consumption, and when using energy demand + date/time features with the random forest model. Including water consumption causes no significant change in accuracy for either model or choice of lookback. Experiments show that the choice of input features and lookback horizon has a varied influence on the different models. We subsequently discuss the findings for each model in detail. There are, however, some general observations from all experiments:

- Lookback should be at least 24h.
- All models/features show similar performance for one-hour-ahead prediction.
- Using weather data or water consumption data does not increase accuracy in general (except for fully-connected NN).

Linear regression:

- Using date/time features significantly improves the results over using historical energy demand only.
- Changing the lookback from 24 to 72 hours causes no noticeable improvement.

Random forest:

- Using date/time features significantly improves the results over using historical energy demand only.
- A longer lookback improves performance.
- Results indicate that RF has the best generalization ability of all models.
- RF adjusts to changes in energy demand patterns, such as the ones introduced by the Covid-19 measures.
- RF is the model that is least sensitive to the choice of input features.

Fully-connected neural network:

- With a sufficiently long lookback, adding hourly water consumption to the input set improves over using historical energy demand data alone.
- When using historical energy demand data as the only input feature, the NN's performance is on par with the performance of the RF model.
- The choice of feature set and the choice of lookback interact with each other.
- The NN does not adapt well to changes in the energy demand patterns, such as the ones introduced by the Covid-19 measures.
- The NN is sensitive to the choice of input features.

Recurrent neural network:

- Date/time features significantly increase performance if the lookback and the lookahead are set to 24 hours.
- Performance does not increase with longer lookback horizons.
- The NN does not adapt well to changes in the energy demand patterns, such as the ones introduced by the Covid-19 measures.
- The RNN is sensitive to the choice of input features.

3.4.4.6 Conclusions

In general, results show that using features engineered from date and time affects prediction performance most significantly, regardless of the choice of model and lookback. Simple models, such as linear regression and random forests perform very well in terms of generalization ability and the ability to adjust to changes in the time series dynamics, as well as very little sensitivity to the choice of input features and lookback.

Especially the random forests show exceptional generalization performance for all choices of input features and lookbacks. Conversely, the neural networks perform well when forecasting based on historical energy demand, but are sensitive to the choice of additional features. Results indicate that the neural-network-based models could benefit from additional regularization; for instance, through dropout layers, L1 or L2 regularization, or batch normalization.

Another potential improvement to the neural-network-based models is the introduction of other, more advanced network topologies and layer types such as convolutional layers or encoder/decoder structures. Results corroborate that network-based models are powerful, but they require extensive tweaking and testing. Using water consumption data as an additional input to forecast energy demand shows some slight improvements when using network-based models, but the full potential of this combination of features is subject to follow-up research.

4 Theory-Informed Deep Learning for Dynamical System Simulation

In the previous chapter, we used theory-agnostic ML models to simulate the behavior of a dynamical system. However, whether ML is capable of solving a particular problem is chiefly governed by (i) the quality and quantity of the training data and (ii) how trustworthy the results have to be. Improving data quality and quantity can improve the generalization ability of a ML model. However, when modeling complex physical systems, it might be impossible to generate training data that is representative of all possible system states. We end up with underconstrained problems and models that learn seemingly correct relationships on the training data but fail to generalize.

Safety-critical applications require simulation interpretable, trustworthy simulation models. However, the black-box nature of ML models entails that ML models hide all causal relationships between input and output. Users cannot interpret the decision process of a black box model, which prevents black-box ML models from being trustworthy. Rudin [Rud19] requires an interpretable model to be constrained by the physical and structural laws of the domain. The emerging paradigms of *theory-guided data science*, *physics-informed neural networks*, and *physics-guided ML* are different approaches that introduce physical and structural constraints into ML. This chapter is based on [LSS+22] and describes neural-network-based architectures for dynamical system simulation.

4.1 Related Work

In the literature, there are several related but distinct surveys on the application of ML models for simulation. The reviews in [RDK+19; BNK20; BDC+18] and [CHB+18] provide an overview of potential use cases for ML approaches within different domains with limited emphasis on how to apply these approaches in practice.

The search for fast and accurate surrogate models for complex phenomena is closely related to the field of neural-network-based simulation. The authors of [KP20] present a comprehensive introduction to data-driven surrogate models, which includes the application of neural networks. For surveys focusing on the application of surrogate models for water resource modeling and building simulation, we refer the reader to [WE19] and [RTB12], respectively.

With a rising interest in addressing the inherent shortcomings of network-based simulation through the integration of prior (theoretical) knowledge, there is a growing number of related publications. The most relevant publications are [KAF+17], which introduces the term *theory-guided data science*, the taxonomic reviews in [vMS+20] and [vMB+21], and the review on hybrid, physics-guided ML techniques in [RS20].

Our work complements these surveys with a comprehensive review of neural networks for system simulation, as opposed to ML methods in general, and by providing concrete examples to highlight and contrast the similarities and differences between different modeling approaches.

4.2 The Initial Value Problem

The function $\mathbf{y}(t) : \mathbb{R} \rightarrow \mathbb{R}^n$ describes the state of the system at time t in terms of vector \mathbf{y}_t that contains a system's $n \in \mathbb{N}$ state variables. The behavior of the state variables over time is described by a set of differential equations. Using vector notation, the differential equations can be expressed through a multidimensional derivative function $f(t, \mathbf{y}(t))$. While this function describes how the state variables evolve, it is impossible to obtain $\mathbf{y}(t)$ directly. The multidimensional derivative function and an initial state \mathbf{y}_0 in the domain of the function constitute the initial value problem (IVP), formalized in Equation 4.1. The system state at time t , $\mathbf{y}(t)$, can be found by solving the initial value problem.

$$\dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t)), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad (4.1)$$

Depending on the nature of the differential equations, finding an exact algebraic solution for the IVP is not always possible. Hence it is common to approximate the continuous IVP numerically. There are many numerical solvers, each of which has its advantages and disadvantages. [WH91; HW96] provide detailed discussions of numerical solutions to ODEs and differential-algebraic systems of equations (DAE); [LeV07] discusses numerical solutions for systems of partial differential equations (PDE) and [MW01] presents an overview of advanced numerical integration schemes. The *forward Euler* is a simple first-order method; given the IVP and a constant step size $h > 0$, it computes a trajectory of estimated system states $[\hat{\mathbf{y}}(t_0) \dots \hat{\mathbf{y}}(t_m)]$, starting at some initial state \mathbf{y}_0 . There are multiple strategies to obtain the forward Euler method; one intuitive approach is to use discrete difference equations to approximate the continuous derivative.

$$\dot{\mathbf{y}}(t) = \frac{\mathbf{y}(t + \Delta t) - \mathbf{y}(t)}{\Delta t} \quad (4.2)$$

Equation 4.2 shows a forward difference approach. Substituting the time step with the step size parameter, $t + \Delta t = ht$, and rearranging yields an equation that approximates a future state of the system based on the current state and the multidimensional derivative function. For simplicity, it is convenient to express time as a discrete set so that the system state at time t can be referred to as \mathbf{y}_t .

$$\hat{\mathbf{y}}_{t+1} = \hat{\mathbf{y}}_t + hf(\hat{\mathbf{y}}_t) \quad (4.3)$$

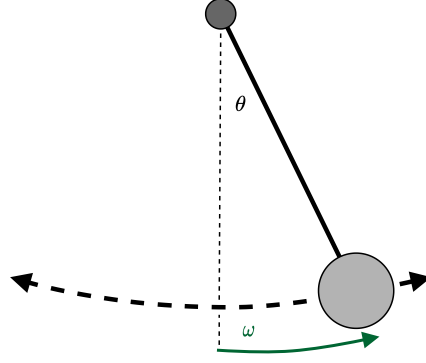


Figure 4.1: We use the ideal pendulum as a case study example to different network-based simulation approaches. The pendulum's system state is described by its angle, θ , and its angular velocity, ω .

The function $f(\hat{\mathbf{y}}_t)$ in Equation 4.3 is the multidimensional derivative function based on the set of differential equations describing the system.

4.3 Dependence on Time

In a time-varying system, future states depend on the sequence of states from some initial condition to the current state. This dependency can be expressed as f mapping both the current state \mathbf{y}_t and the current time t to a future state \mathbf{y}_{t+1} , as seen in Equation 4.4.

$$\mathbf{y}_{t+1} = \mathbf{y}_t + f(t, \mathbf{y}_t) \quad (4.4)$$

In a time-invariant model, a future state depends only on the current state \mathbf{y}_t , which implies that there is no explicit initial state in the system. Consequently, time-invariant models can generate trajectories starting from any arbitrary system state. Or, in other words, time-invariant models can be parameterized with different initial states. This invariance translates to the function f mapping the current state to the next state, without an explicit dependency on time t , as described in Equation 4.5.

$$\mathbf{y}_{t+1} = \mathbf{y}_t + f(\mathbf{y}_t) \quad (4.5)$$

Whether a system can be described by a time-invariant model depends on the nature of the system itself.

4.4 Case Study

In the subsequent sections, we use a simple dynamical system as a case study example to which we can apply the different network-based simulation approaches. The concept of

an *ideal pendulum* refers to a mathematical model of a physical pendulum and neglects the influence of friction, air resistance, or bending of the pendulum arm. A schematic representation of the systems can be seen in Figure 4.1. We can characterize the state of the ideal pendulum by its angle θ and its angular velocity ω . The differential equation in Equation 4.6 describes the system behavior over time.

$$\frac{\partial^2 \theta}{\partial t^2} + \frac{g}{l} \sin \theta = 0 \quad (4.6)$$

The variables g and l correspond to the constant gravitational acceleration and the length of the pendulum arm, respectively. Note that the angular velocity ω is the first derivative of the pendulum's angle θ . With this information, we can re-formulate the second-order differential equation in Equation 4.6 into two first-order differential equations and express them as a multidimensional derivative function $f(\mathbf{y}(t))$, seen in Equation 4.7.

$$f(\mathbf{y}(t)) = \begin{bmatrix} \frac{\partial \omega}{\partial t} \\ \frac{\partial \theta}{\partial t} \end{bmatrix} = \begin{bmatrix} -\frac{g}{l} \sin \theta \\ \omega \end{bmatrix} \quad (4.7)$$

The ideal pendulum is an *autonomous* system as there are no external influences on its dynamics. As seen in Equation 4.6, the changes to the state of the system are independent of time. Consequently, we can describe the system using a time-invariant model. These two characteristics simplify the simulation approach. However, more complex systems do not generally share these properties. We will address these issues in Section 4.8.3.

Solving the IVP for the ideal pendulum results in a trajectory of system states, like the one seen in Figure 4.2b. An alternative representation of the solution, a phase portrait, with the angle θ represented along the x-axis and the velocity ω along the y-axis, can be seen in Figure 4.2a. To obtain a phase portrait, the numerical solver evaluates the IVP

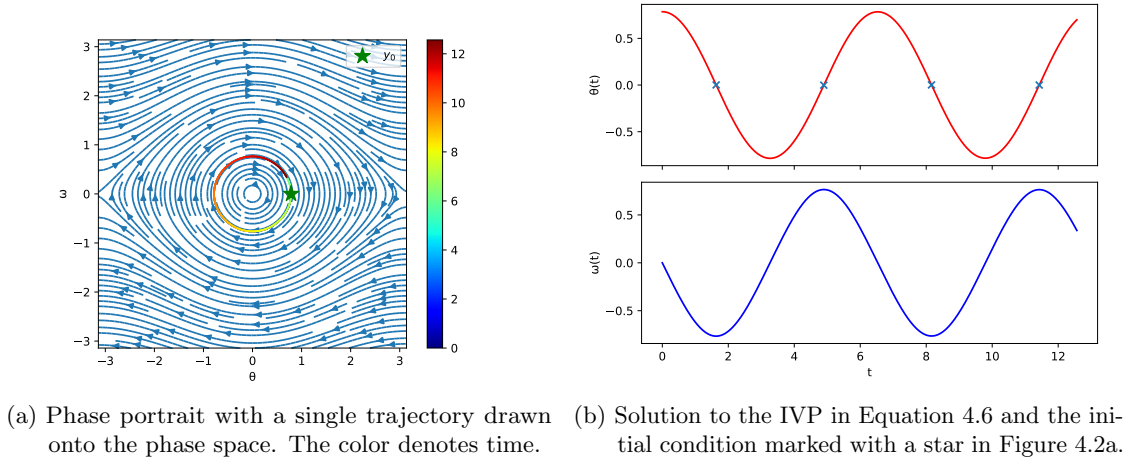


Figure 4.2: The ideal pendulum, when approximated using a numerical solver.

starting from different initial conditions. The circles in the phase portrait correspond to solutions repeating themselves or, formulated differently, an oscillation in time.

The ideal pendulum is a well-understood system that lends itself well to exemplifying simulation approaches. It is relatively simple to describe the system’s dynamics with differential equations, and there are many numerical solvers that can approximate an appropriate solution to the IVP. In real-world incarnations of a pendulum, there are more complicated interactions and phenomena such as friction, air resistance, bending of the pendulum arm, etc., and it is possible to develop more complex models accounting for these factors. However, it is challenging to design sufficiently complex models, and there is no guarantee that they properly account for all factors.

4.5 Neural Networks

The term *neural networks* encompasses a variety of model architectures that have shown the ability to solve a wide range of problems. The simulation approaches described in the subsequent sections use a single type of network, the *fully-connected* neural network, to solve regression problems by estimating trajectories of system states. For a more general discussion of deep learning, please refer to [Bis94] or [GBC+16].

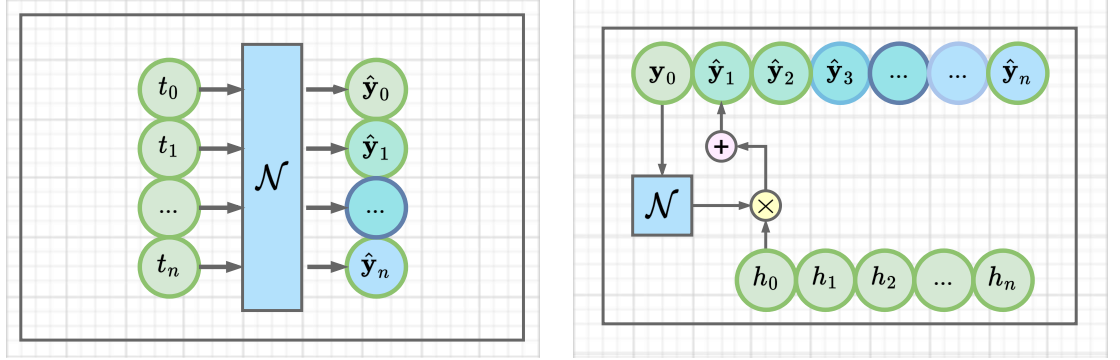
4.6 Model Taxonomy

One of the main challenges of studying neural networks is that, depending on the application, the terminology to describe certain concepts and ideas is not always consistent, especially in the intersection between deep learning, numerical simulation, and physics. Besides, the literature shows a tendency to emphasize the success of particular methods in solving specific problems instead of explaining their details and limitations. Therefore, it is often difficult to fully comprehend distinctive contributions to the field because the papers are hard to understand.

We propose a simple taxonomy based on two distinct categories of models: *direct-solution models* and *time-stepper models*. Direct-solution models do not perform explicit integration but instead approximate a mapping between a time instance and the system state at this instance. Conversely, time-stepper models employ an approach similar to the one applied by numerical solvers, where a future state is calculated from the current state of the system. The differences between direct-solution and time-stepper models affect how each type of model handles varying initial conditions and inputs.

Time-stepper models can simulate time-invariant systems starting at any arbitrary state, whereas direct-solution models solve the IVP for a single initial state. Hence, direct-solution models must be re-trained for the simulation to start at different initial conditions, while time-steppers can be parameterized with the new initial conditions.

The same is true for external inputs; it is possible to parameterize time-stepper models with different sets of external inputs, whereas direct-solution models must be re-trained. In more general terms, time-stepper models approximate the dynamics of the system, while direct-solution models solve the IVP given a set of initial conditions and external inputs.



(a) The direct-solution model. A network approximates a mapping between the time instance and the system state at this time instance. (b) The time-stepper model. A network \mathcal{N} estimates the derivative of the function describing the dynamics at multiple time instances that is then integrated by an Euler solver.

Figure 4.3: The Model Taxonomy. Direct-solution models produce a simulation trajectory without explicit numerical integration. Time-stepper models apply explicit integration, similar to regular numerical solvers.

4.7 Direct-Solution Models

Direct-solution models map a time instance t_k to the corresponding system state \mathbf{y}_k . To construct a direct-solution model, we train a network to approximate the solution along a set of *collocation points*, sampled from the true system. The network acts as a trainable interpolator and allows us to evaluate the solution at any arbitrary point in time.

In the literature, direct-solution models are commonly used to simulate systems, where the dynamics are governed by *partial differential equations* (PDE) instead of ODEs. One likely reason is that PDEs are typically more computationally expensive to solve numerically, which seems to be a stronger motivation to apply a data-driven approach instead. Conversely, many applications of ODEs require them to be solved for different initial conditions, which is somewhat cumbersome to implement with direct-solution neural network models. The main difference between a model solving a PDE and a model solving an ODE is that for the ODE the input is a time coordinate, whereas for the PDE the input consists of temporal and spatial coordinates. While there seem to be stronger arguments for using direct-solution models with PDEs, there is merit in applying them to solve ODEs.

The most prohibitive issue in training direct-solution models is to obtain enough training data for the model to reach appropriate accuracy and generalization performance. A naive approach to fitting a network to a set of collocation points, such as the one described in Section 4.7.2, is likely to fail in capturing the true underlying dynamics, without advanced regularization or densely sampled collocation points. There are, however, novel approaches, popularized by the introduction of *physics-informed neural networks* (PINN) [RPK19], that address these issues. In essence, PINN requires the use of automatic differentiation and the encoding of ODEs and PDEs into the architecture and the loss function of the network. In the remainder of this section, we will discuss different architectures for direct-solution models and apply them to simulate the ideal pendulum.

4.7.1 Network Architecture, Training, Evaluation

All direct-solution simulation models described in this section use a neural network consisting of three fully-connected layers of 32 neurons, activated by softplus functions and a single linear output layer. The weights are initialized based on the input shape to each layer [GB10] and biases as zeros. The layers do not use regularization or constraint functions for the weights, biases, or outputs.

We train each model using data from a single simulation trajectory sampled at equidistant collocation points, as seen in 4.4. The objective is to obtain a model that can estimate the system state at any arbitrary point despite only being shown the true values at the collocation points.

4.7.2 Vanilla Direct-Solution

Direct-solution models map a time instance t_k to an estimate of the system state $\hat{\mathbf{y}}_k$, i.e., the network learns a continuous function of time that can be evaluated at any arbitrary point.

$$\hat{\mathbf{y}}_k = \mathcal{N}(t_k) \quad (4.8)$$

To model the ideal pendulum, we can use the network with a single input and two outputs $\hat{\mathbf{y}}_k = [\hat{\theta}_k \ \hat{\omega}_k]$, and to obtain a trajectory from the model, we can evaluate it using multiple time instances. The invocations of the network are independent of each other, so it is possible to estimate multiple system states in parallel.

To train the model, we use the MSE between the estimated system state (\hat{Y}) and the true state (Y) in the collocation points, calculated for each state of the n state space variables of the m entries in the trajectory as the collocation loss function L_C .

$$L_C = \text{MSE}(\hat{Y}; Y) = \frac{1}{mn} \sum_{k=0}^{m-1} \sum_{i=0}^{n-1} (\hat{y}_{k,i} - y_{k,i})^2 \quad (4.9)$$

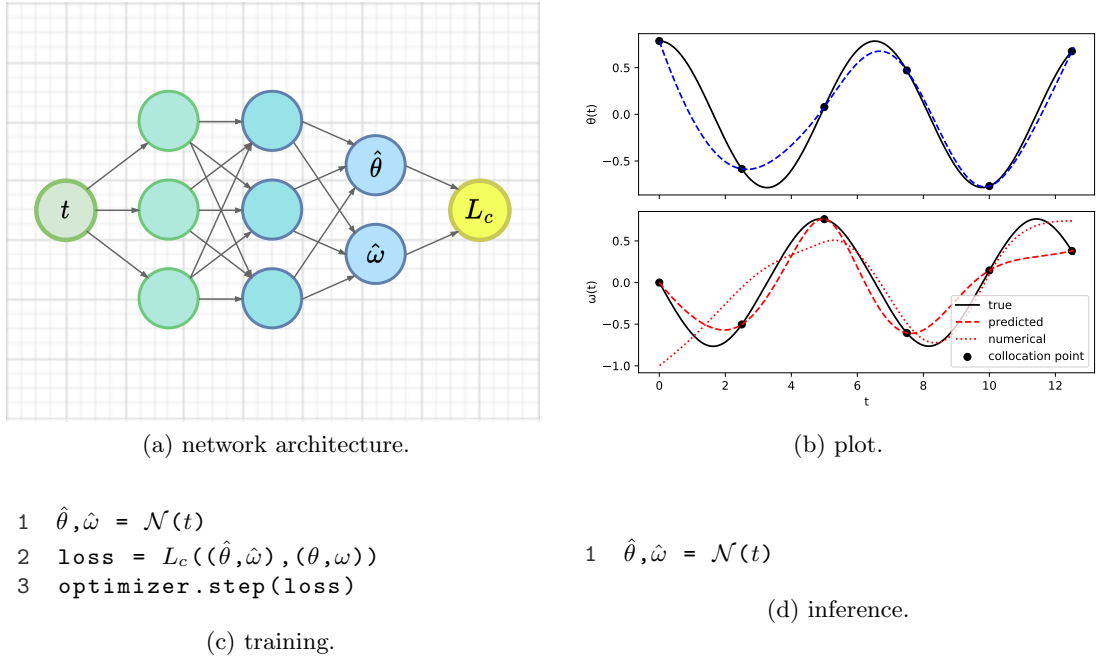


Figure 4.4: A vanilla direct-solution model. The neural network $\mathcal{N} : \mathbb{R} \rightarrow \mathbb{R}^2$, maps the time instances t to the estimated solution $\hat{\theta}, \hat{\omega}$. The black markers indicate the collocation points. Note that the estimates fit the true system in the collocation points reasonably well, but diverge significantly otherwise. Additionally, $\hat{\omega}$ does not behave like the actual numerical derivative of $\hat{\theta}$ either.

Note that the model learns to approximate the trajectory starting at a specific initial state and that it cannot be modified to obtain another trajectory starting at a different initial state without re-training. Naive direct-solution models are particularly sensitive to the quality and quantity of training data. Depending on the sampling strategy, it is likely that the network has to find a highly non-linear mapping from sparse data.

Consider the example in 4.4, while the estimates match the collocation points perfectly, generalization performance outside these points is exceptionally poor. It is important to note that this is not just one specific choice of training points that prevents the network from learning the true dynamics of the system, but many other samplings cause equally poor results. The most obvious way to mitigate this particular sampling issue is to increase the sampling rate. However, there are many cases where this is impossible, either because acquisition is expensive or impractical, or where the device capturing the data cannot be configured to a higher frequency.

In the deep learning community, there are many general-purpose regularization approaches designed to address this issue and improve the generalization performance of neural network models. There is, however, another, more domain-oriented approach to mitigating

the problem of inferring from partial data: using knowledge about the equations describing the dynamics of the system.

In the ideal pendulum, one part of the system state (the angular velocity ω) is the derivative of the other, which is a common relationship among systems that can be described by differential equations. However, a naive direct-solution model with two outputs does not leverage this property. As depicted in 4.4, the network correctly predicts the system state at the collocation point. However, the estimated values for the angular velocity $\hat{\omega}$ neither match the derivative of the estimated angle $\hat{\theta}$ nor the true value ω .

4.7.3 Automatic Differentiation in Direct-Solution models

One strategy to implement differential relationships between state variables is to use automatic differentiation. For the ideal pendulum, for instance, we can estimate the angular velocity by calculating the first derivative of the estimate for the pendulum’s angle $\hat{\theta}$ instead of having the network map it. To implement this approach, we construct a network with a single output, obtain the angular velocity through automatic differentiation with respect to time and then plug both state variables into the loss function to backpropagate and train the network. The trajectories in Figure 4.5 show a significant improvement in generalization performance.

Depending on the mode, automatic differentiation introduces additional computational cost. Reverse-mode automatic differentiation, also denoted backpropagation, as depicted in Figure 4.5 requires a pass of the computation graph, going from the output $\hat{\theta}$ to the input t . During training, this requirement is not an issue, as the optimizer needs to backpropagate to update the weights and the biases of the network. During inference, however, reverse-mode automatic differentiation introduces additional computation time and memory usage. To dispense with the separate backward pass, it is possible to use forward differentiation instead. With forward differentiation, the derivatives are evaluated when the input is passed through the network. However, not all deep learning frameworks support forward differentiation, most likely because the optimizer typically needs to evaluate the derivative of the loss with respect to the weights and biases of the network; which can be implemented more efficiently using reverse-mode differentiation.

4.7.4 Physics-Informed Neural Networks

In situations where we have access to the governing equations for a dynamical system, we can utilize them in an approach popularized as *physics-informed neural networks* (PINN) [RPK19]. One way of introducing information from the governing equations is to extend the loss function with a separate equation loss term L_{eq} that ensures the network’s estimates obey the constraints imposed by the equations.

Incorporating the equation loss constrains the search space of the optimizer to network parameters that result in solutions consistent with the governing equations. Note that

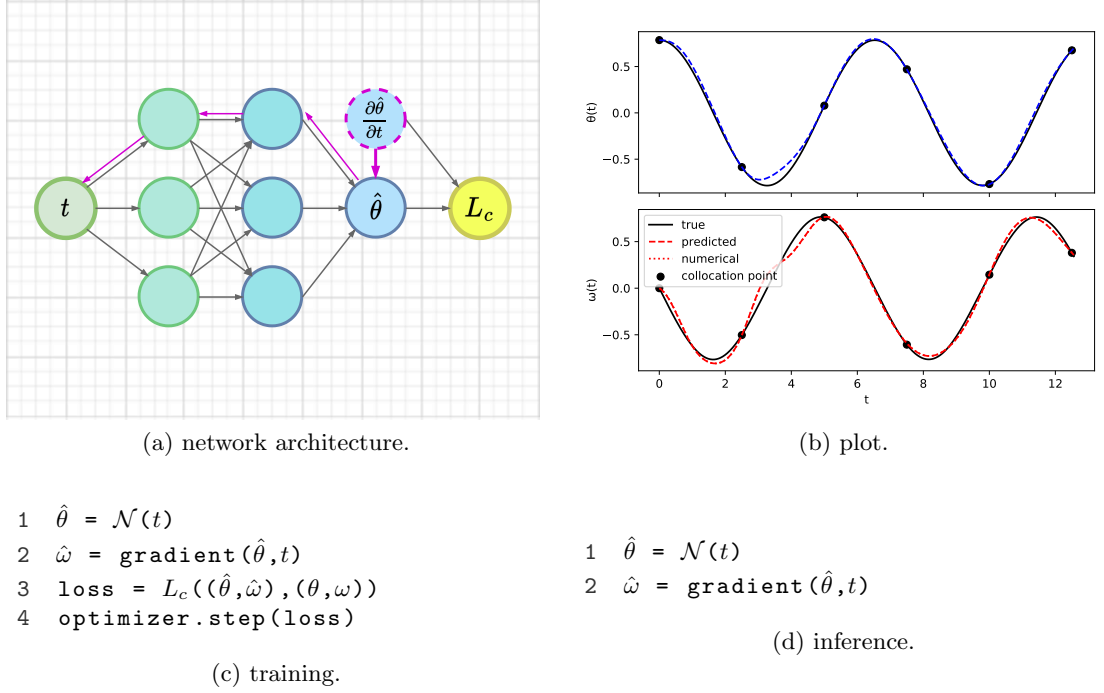


Figure 4.5: Automatic differentiation in a direct-solution model. The network $\mathcal{N} : \mathbb{R} \rightarrow \mathbb{R}$, maps the time instances t to the estimated solution for the pendulum's angle $\hat{\theta}$, and instead of the network predicting the angular velocity ω , it is obtained through automatic differentiation of the estimate for θ . This architecture ensures that the output that should be the derivative of the other behaves like a true derivative. Note that the generalization performance of the model increases significantly.

the equation loss guarantees that the estimated system state is a valid solution to the governing equations, but it does not necessarily mean that the estimate is also the correct solution for a particular temporal and spatial location. Hence, we need to combine it with the loss between the estimated and the true values of the trajectory at the collocation points to obtain an appropriate mapping.

The total loss for training the PINN then consists of the *collocation loss* L_c that penalizes the distance between the estimates and the true values in the collocation points, and the *equation loss* L_{eq} that penalizes discrepancies between the estimates and the governing equations.

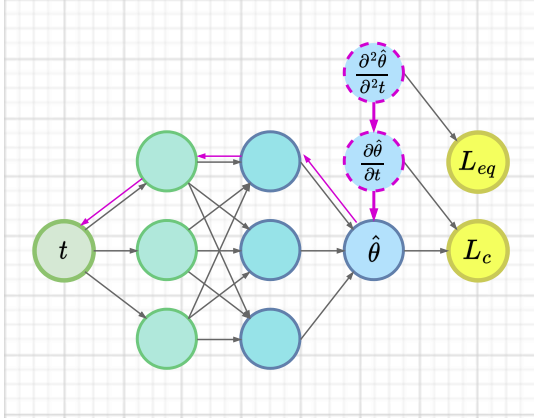
$$L_{PI} = L_c + L_{eq} \quad (4.10)$$

The authors of [RPK19] originally proposed this approach to solve PDEs, but we can easily apply it to solve ODEs by only considering a time dimension instead of temporal and spatial coordinates. For the ideal pendulum, we can use Equation 4.6 and square it

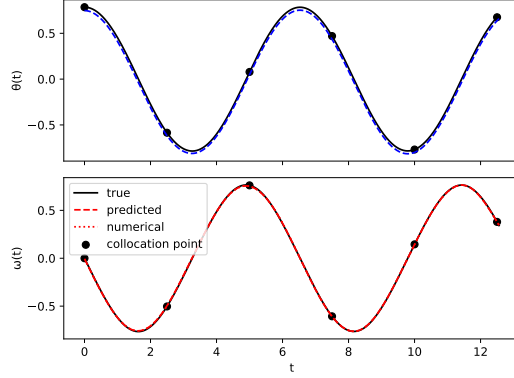
to ensure it is positive.

$$L_{eq} = \frac{1}{m} \sum_{k=0}^{m-1} \left(\frac{\partial \hat{\omega}_k}{\partial t} - \frac{g}{l} \sin \hat{\theta}_k \right)^2 \quad (4.11)$$

As the collocation loss, we use the squared distance between the estimated system state and the true system state, as we did for the vanilla direct-solution model and the model using automatic differentiation.



(a) network architecture.



(b) plot.

```

1   $\hat{\theta} = \mathcal{N}(t)$ 
2   $\hat{\omega} = \text{gradient}(\hat{\theta}, t)$ 
3   $\partial \hat{\omega} = \text{gradient}(\hat{\omega}, t)$ 
4   $\text{loss} = L_c((\hat{\theta}, \hat{\omega}), (\theta, \omega)) + L_{eq}(\hat{\theta}, \partial \hat{\omega})$ 
5  optimizer.step(loss)

```

(c) training.

```

1   $\hat{\theta} = \mathcal{N}(t)$ 
2   $\hat{\omega} = \text{gradient}(\hat{\theta}, t)$ 

```

(d) inference.

Figure 4.6: Physics-informed direct-solution model. The network $\mathcal{N} : \mathbb{R} \rightarrow \mathbb{R}$, maps the time instances t to the estimated solution for the pendulum's angle $\hat{\theta}$ and the estimate for the angular velocity $\hat{\omega}$ is obtained through automatic differentiation of $\hat{\theta}$. To ensure that the solution obeys the physics governing the dynamics of the pendulum, the network is trained by minimizing Equation 4.10.

We can use automatic differentiation to obtain an estimate of the angular velocity $\hat{\omega}$ and its derivative with respect to time $\frac{\partial \hat{\omega}}{\partial t}$ by differentiating the estimate of the pendulum's angle $\hat{\theta}$ twice, as depicted in Figure 4.6.

Using automatic differentiation in PINNs is particularly useful because, in physics, derivatives are typically part of the equations governing the dynamics of a system. Additionally, using automatic differentiation to obtain state variables from other state

variables ensures that the derivatives used in the equation loss are actual partial derivatives instead of estimates of the derivatives produced by the network.

Training PINN using a gradient-based optimizer requires special attention to the learning rate because the collocation loss and the equation loss converge at different speeds. To alleviate this issue, the authors of [WYP20; WTP20] propose weighing the separate terms in the combined loss function to ensure uniform convergence.

4.7.5 Hidden-Physics Networks

With access to the governing equations, it is possible to estimate properties of the system that are not present in the training trajectories. In real-world applications, there are many reasons why it can be difficult or even impossible to measure certain states of a dynamical system. These *hidden variables* can be estimated using *hidden physics neural networks* (HNN) [RYK20].

To describe the application of an HNN, we need to adjust our case study slightly; specifically, we suppose that the length of the pendulum’s arm l changes over time, and we do not have the necessary data to train the model directly. With access to the ODE describing the pendulum’s behavior, we can have the network estimate the length of the pendulum’s arm and use the equation loss from the PINN example to penalize estimates that violate the governing ODE.

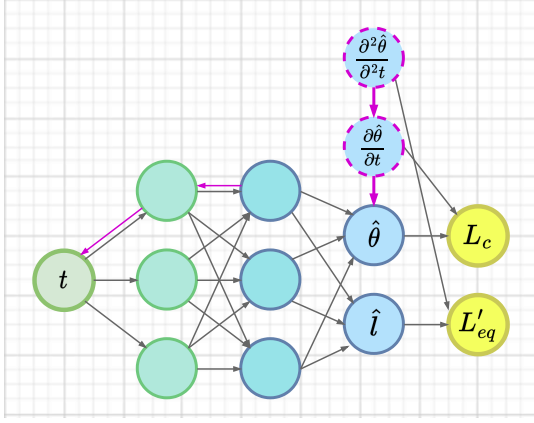
$$L'_{eq} = \frac{1}{m} \sum_{k=0}^{m-1} \left(\frac{\partial \hat{\omega}_k}{\partial t} - \frac{g}{\hat{l}} \sin \hat{\theta}_k \right)^2 \quad (4.12)$$

Note that we replaced the static, measured version of the length l with the length estimated by the network \hat{l} . The collocation loss remains unchanged as we do not have collocation data on the length. With the search space for \hat{l} confined to solutions that satisfy the governing equations and the estimates for the angle and the angular velocity of the pendulum constraint by the collocation loss, we can approximate the behavior of l reasonably well, as seen in 4.7.

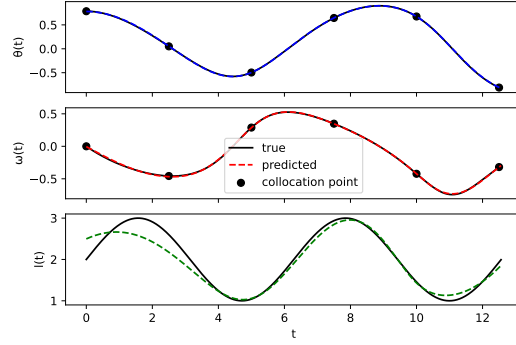
PINNs and HNNs apply the same principle, use a similar network architecture and enforce consistency with the governing equations of the system through a separate term in the loss function. The subtle difference is that in HNNs the value of the hidden variable is inferred implicitly by finding the correct mapping between time and estimates for the observable variables and then approximating the hidden variables within the search space constrained by the governing equations.

4.8 Time-Stepper Models

In the previous section, we used a neural network to directly map the time instances to system states, eliminating the need for numerical approximation. However, when



(a) network architecture.



(b) plot.

```

1   $\hat{\theta}, \hat{l} = \mathcal{N}(t)$ 
2   $\hat{\omega} = \text{gradient}(\hat{\theta}, t)$ 
3   $\partial \hat{\omega} = \text{gradient}(\hat{\omega}, t)$ 
4   $\text{loss} = L_c((\hat{\theta}, \hat{\omega}), (\theta, \omega)) + L'_{eq}(\hat{\theta}, \partial \hat{\omega}, \hat{l})$ 
5  optimizer.step(loss)

```

(c) training.

```

1   $\hat{\theta}, \hat{l} = \mathcal{N}(t)$ 
2   $\hat{\omega} = \text{gradient}(\hat{\theta}, t)$ 

```

(d) inference.

Figure 4.7: The hidden physics model. The neural network $\mathcal{N} : \mathbb{R} \rightarrow \mathbb{R}^2$, maps the time instances t to the estimated solution for the pendulum's angle $\hat{\theta}$ and the estimated length of the pendulum \hat{l} , which varies in time for illustrative purposes. Note that \hat{l} is not part of the collocation loss L_c because there is no training targets for it. Instead, the estimate for the pendulums length is tuned through the equation loss L'_{eq} .

we reconsider the approach applied by a numerical solver, such as the forward Euler method described in Section 4.2, we can see that analytically obtaining the derivative function constitutes the main challenge, especially when dealing with complex systems. A way to address this problem with a data-driven approach is to train a neural network to approximate the derivative function and then use a numerical solver to obtain a simulation trajectory through a step-wise approximation of the true solution of the IVP. We refer to models applying this approach as *time-stepper models* because they take multiple steps through time to simulate an entire trajectory. This approach combines the advantages of well-studied numerical solvers with the simplicity of neural networks.

Time-stepper models can be distinguished by (i) how the network approximates the derivatives and (ii) which integration scheme is used. Note that these two characteristics are relatively independent. Hence, the models we describe in this section do not constitute an exhaustive list but rather provide an overview of the models commonly encountered in the literature.

4.8.1 Network Architecture, Training, Evaluation

As time-stepper models approximate the dynamics of a system, rather than an actual sequence of states, they can produce simulations for different initial conditions. It is possible to train time-stepper models similar to direct-solution models, i.e., using one single trajectory of states. However, it is unlikely that this approach allows a time-stepper to generalize well enough to produce accurate simulations from different initial conditions than the ones used for training. To address this issue, we can use multiple training trajectories starting at different initial states. To obtain them, we can even cut a long trajectory into shorter ones and treat them as separate simulations, starting from different initial conditions. We can extend the loss function in Equation 4.9 to calculate the MSE over l trajectories, with m entries and n state variables.

$$L_{TS}(\hat{Y}, Y) = \frac{1}{l} \sum_{i=0}^{l-1} L_C(\hat{Y}, Y) \quad (4.13)$$

All time-stepper models we describe in this section use a deep, fully-connected neural network consisting of eight layers of 32 neurons with softplus activation functions, a single linear output layer, weight initialization based on the input shape [GB10], and no regularization or constraint functions on the weights and biases. We train each model on 100 single-step trajectories, with initial states between in $(-1, 1)$ for θ and ω . We choose the initial conditions for training using a Latin hypercube sampling strategy. To

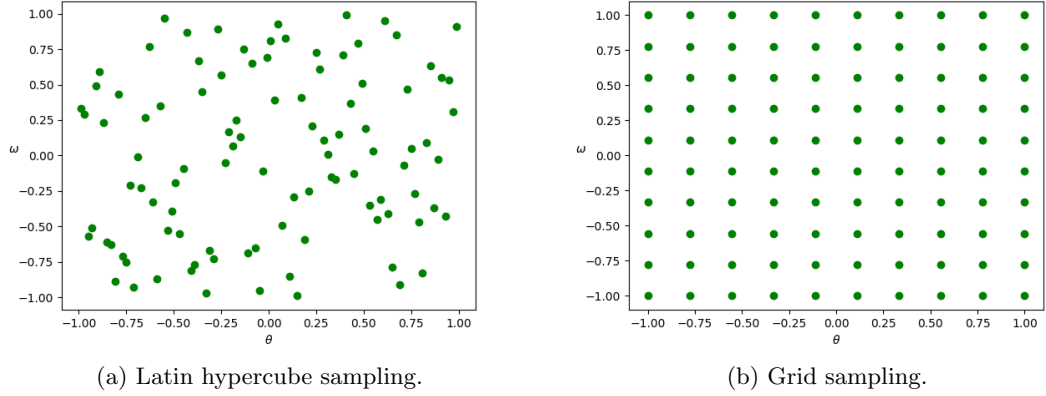


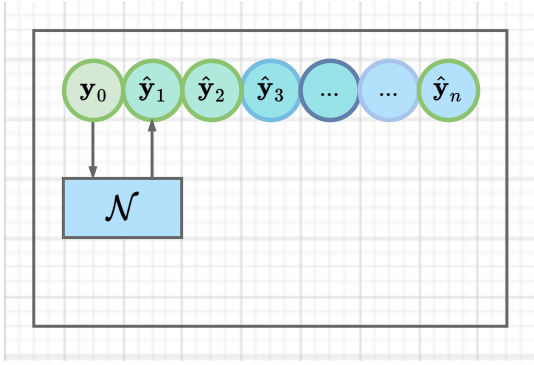
Figure 4.8: Two sampling strategies; near-random Latin hypercube sampling for the training and grid sampling for validation.

validate generalization performance, we select 100 initial states from a grid, simulate the trajectories for 4π seconds using the original ODE, and compare the outputs to the trajectories estimated by the models. Figure 4.8 visualizes the difference between Latin hypercube and grid sampling.

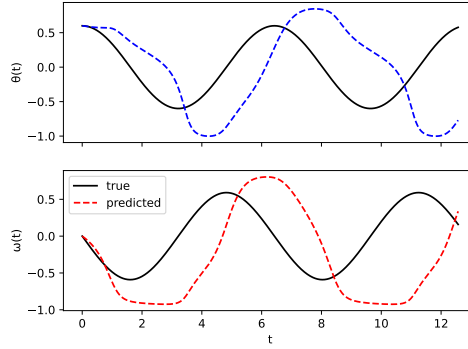
4.8.2 Integration Schemes

How the solver evaluates and integrates the derivative is a distinctive characteristic of the model. In addition to governing how the model simulates a trajectory, the numerical solver influences how to train the model. Minimizing an optimization criterion that is a function of the system state, for instance, causes a direct dependency on how the solver obtains the state from the derivative. We subsequently discuss how to use certain numerical solvers in time-stepper models and the integration strategy affects the performance of the models.

4.8.2.1 Direct Time-Stepper



(a) network architecture.



(b) plot.

```

1  $\hat{Y}[:,0,:] = Y_{t_0}$ 
2 for i in 0...m-1
3    $\hat{Y}[:,i+1,:] = \mathcal{N}(\hat{Y}[:,i,:])$ 
4   loss =  $L_{ts}(Y, \hat{Y})$ 
5   optimizer.step(loss)

```

(c) training.

```

1  $\hat{y}[0,:] = y_{t_0}$ 
2 for i in 0...m-1
3    $\hat{y}[i+1,:] = \mathcal{N}(\hat{y}[i,:])$ 

```

(d) inference.

Figure 4.9: Direct time-stepper model. The neural network $\mathcal{N} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, maps a system state θ_t, ω_t to the estimated next state $\hat{\theta}_{t+1}, \hat{\omega}_{t+1}$. These models typically generalize poorly outside the state space they were trained for. In time-stepper models the error accumulates, and the simulation becomes inaccurate.

In a direct time-stepper model a neural network approximates a mapping between subsequent system states:

$$\hat{\mathbf{y}}_{t+1} = \mathcal{N}(\mathbf{y}_t) \quad (4.14)$$

During training, we can divide a single trajectory with m entries into $m - 1$ single-step trajectories and map each initial state one step into the future in a single, parallel invocation of the network, and compare the vectorized network output to the actual next values in each trajectory, as depicted in Figure 4.9c. Note that we omit the m -th entry because it has no subsequent system state in the training trajectory. To obtain an entire simulation trajectory for a single initial condition during inference, however, we need to repeatedly introduce the predicted states into the network, as seen in Figure 4.9d. When we simulate the system for different initial conditions, we can parallelize the inference steps. Figure 4.9b shows that the simulation for a single initial condition diverges from the true trajectory after the first few steps.

4.8.2.2 Residual Time-Stepper

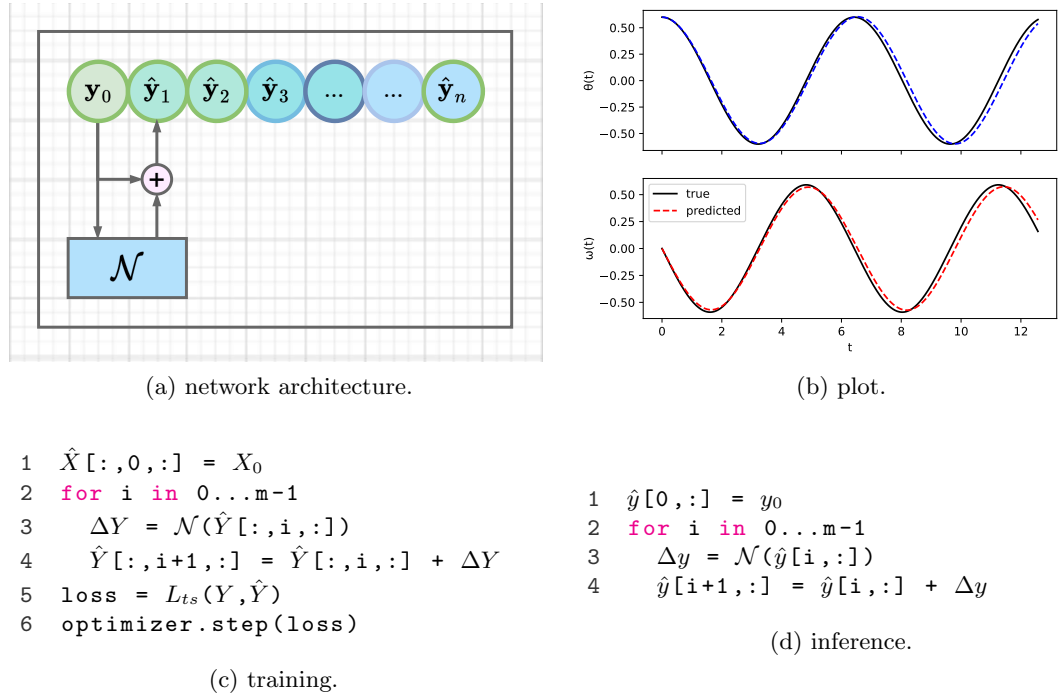


Figure 4.10: Residual time-stepper model. The output of the network $\mathcal{N} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, is added to the input system state θ_t, ω_t to obtain the next state $\hat{\theta}_{t+1}, \hat{\omega}_{t+1}$. Note that this approach generalizes significantly better than the direct time-stepper model.

Instead of directly approximating the mapping between consecutive states, we can train a network to estimate a derivative-like difference between each current and next state

that we can then add to the current state to obtain the next one.

$$\hat{\mathbf{y}}_{t+1} = \mathbf{y}_t + \mathcal{N}(\mathbf{y}_t) \quad (4.15)$$

This model architecture is known as *residual network* [HZR+15] in the deep learning community. Residual networks have been used successfully for many applications, including image classification and natural language processing. Note that this formulation closely resembles the discrete forward Euler method described in Equation 4.3 but misses an explicit, separate term for the step size. Instead, the network implicitly scales the derivative to account for the step size, if the training data is sampled at equidistant time steps. The key advantage of a residual architecture over a direct approach is that the residual network is expected to approximate the dynamics of the system instead of the actual sequence of system states, which can be easier. Figure 4.10 shows training and inference for the residual time-stepper model. Note how much closer the model output follows the true trajectory in Figure 4.10b.

4.8.2.3 Euler Time-Stepper

An obvious variation to the residual time-stepper approach is to explicitly scale the contribution of the residual, derivative-like block by a step size h_t , which results in a model that is very similar to a numerical forward Euler solver.

The scientific community has noted the close resemblance between ODEs and residual networks [QWX19], and several works subject residual networks to classical stability analysis [CMH+18; RH18; RH20]. It is straightforward to implement a forward Euler integration scheme; however, its simplicity typically precipitates lower accuracy compared to more sophisticated integration methods, such as Midpoint methods, linear multistep methods, or the family of Runge-Kutta methods. As result, the integration of advanced numerical solvers into time-stepper models has become a research interest. The authors of [RPK18], for instance, employ *linear multistep methods*, where they use several past states and their derivatives to estimate the next step. Linear multistep methods still only require a single network evaluation per step, which causes them to be efficient.

4.8.2.4 Neural Ordinary Differential Equations

Unlike the previous subsection headings, the term *neural ordinary differential equation* (NODE) [CRB+19] does not refer to a specific model architecture, but to the concept of interpreting a deep learning problem as a dynamical system. In NODEs, models combine a numerical solver with a neural network that approximates the derivative of the system. A schematic representation of a NODE model can be seen in Figure 4.12.

Originally introduced for image classification, NODEs were used to find ODEs that result in images of the same class converging to a cluster that can be separated from unrelated images. Consequently, in image classification, intermediate outputs of a NODE have no

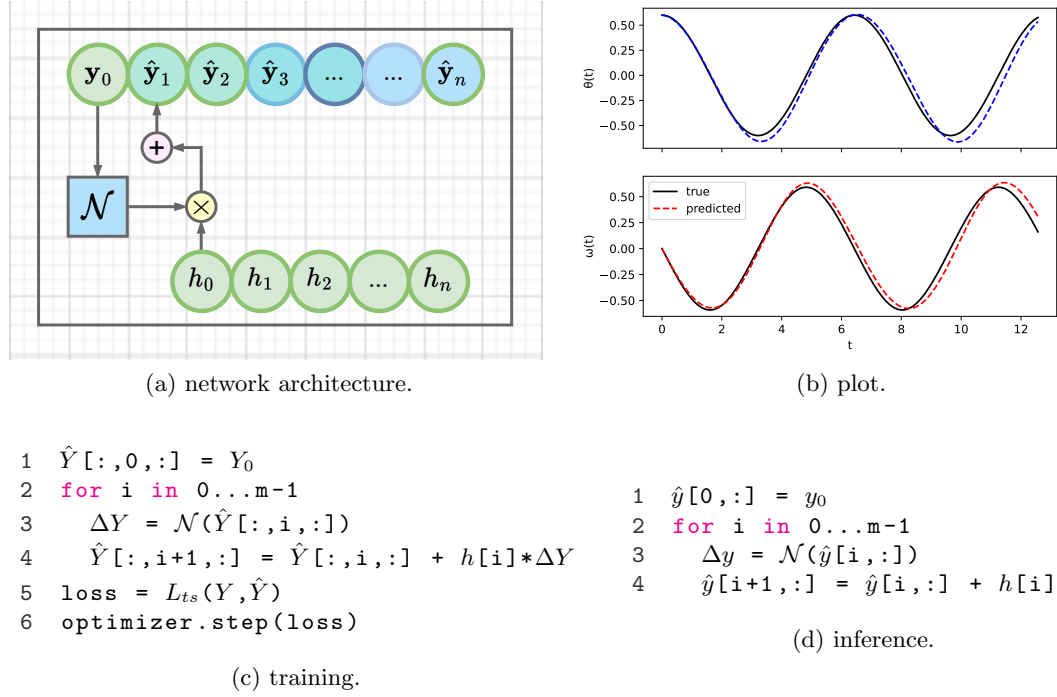
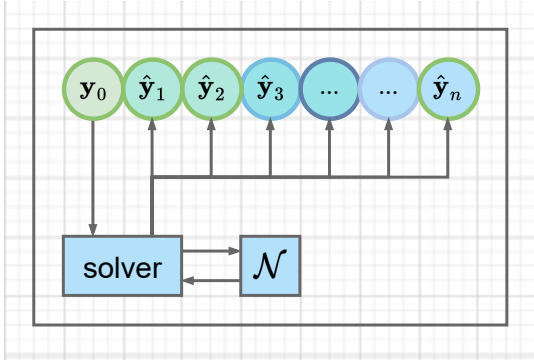


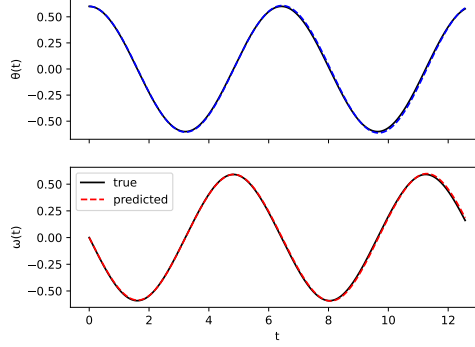
Figure 4.11: Euler time-stepper model. The output from the network $\mathcal{N} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is multiplied by a step size term h before being added to the input state to obtain the estimate for next state. In this example, explicitly modeling the step size only shows minimal improvements over the residual time-stepper. A possible reason is that the step size during training and during inference is the same.

inherent meaning, i.e., the only quantity of interest is the final system state \hat{y}_m and, because there are no intermediate values in the target data, it is impossible to minimize the single-step error.

NODEs have since been applied to a variety of problems. For instance, instead of specifying a discrete sequence of hidden layers, NODEs can be used as *continuous-depth models*, where the solver's step size can be tweaked to balance numerical precision and computational performance in continuous-depth residual networks and continuous-time latent variable models [CRB+19]. The numerical stability of NODEs is closely related to the stability of integration schemes in classical ODEs. Hence, to address convergence issues, it is possible to use Lyapunov stability theory [MPB+20] or spectral projections [QGM+20]. In [FJN+20] the authors address the computational overhead of NODEs over classical neural networks through stability regularization. The authors of [PMP+20] develop graph NODEs, incorporating prior knowledge to speed up training and increase accuracy. Other methods to increase performance include introducing



(a) network architecture.



(b) plot.

```

1  $\hat{Y} = \text{odeint}(\mathcal{N}, Y_0, t_{start}, t_{end}, "rk4")$ 
2  $\text{loss} = L_{ts}(Y, \hat{Y})$ 
3  $\text{optimizer.step}(\text{loss})$ 

```

(c) training.

```

1  $\hat{y} = \text{odeint}(\mathcal{N}, y_0, t_{start}, t_{end}, "rk4")$ 

```

(d) inference.

Figure 4.12: Neural ordinary differential equation (NODE) model. The term NODE refers to model architectures where a numerical solver integrates the derivatives approximated by a neural network. NODEs mimic the programming API of traditional numeric ODEs; hence the solver is independent of the neural network, and it is simple to switch between different solver types.

inductive biases in *Hamiltonian NODE* architectures [ZDC19] and penalizing higher order derivatives in the loss function [KBJ+20]. The authors of [LXS+19a; JB19; GLP19; LWC+20] generalize deterministic NODEs to *stochastic NODEs* in order to account for noise and uncertainties in dynamical systems.

The fundamental issue of interpreting NODEs as true ODEs is that NODEs might have trajectory crossings, implying that for the state at the intersection, the system can evolve in two different ways [OKH+20]. For NODEs to behave like true ODEs, there seems to be a threshold in the step size [OKH+20], i.e., if a neural network was trained with a particular step size, it will perform equally well or better when used with a smaller step size during inference. Regularization, such as ensuring a Lipschitz constant of below 1 can be used to guarantee that solutions are unique [BGC+19].

4.8.3 External Input

In the previous sections, we described how to use time-stepper models to simulate dynamical systems in which the system state determines the derivative. However, the dynamics of many real-world systems are affected by outside influences, such as external

forces or actuation signals from a controller. The architecture of time-stepper models allows us to conveniently introduce such external inputs at each evaluation of the derivative function. Subsequently, we will discuss two popular methods to integrate external, time-varying inputs into time-stepper models.

4.8.3.1 Neural State-Space Model

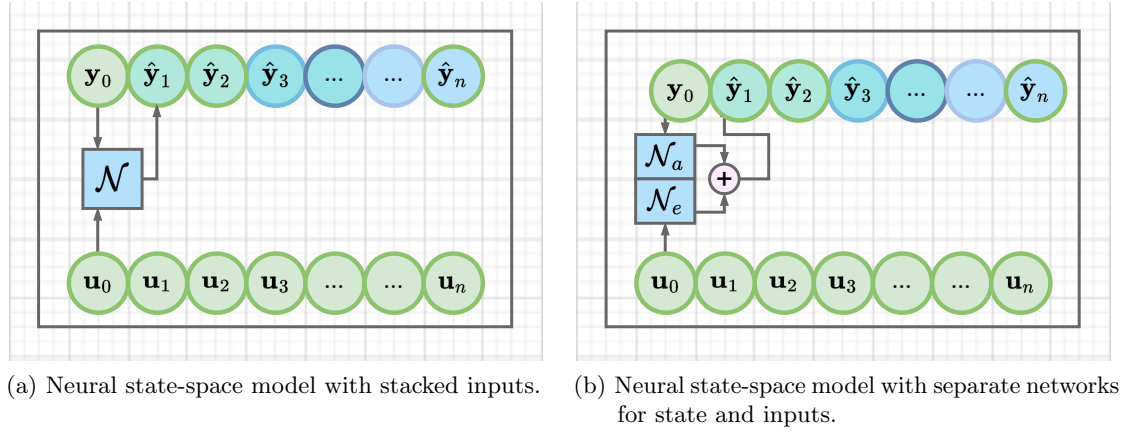


Figure 4.13: Two different architectures that allow for incorporating external inputs into time-stepper models.

An intuitive way of including external inputs \mathbf{u}_t when inferring the next state $\hat{\mathbf{y}}_{t+1}$, is to concatenate them with the state variables of the current state \mathbf{y}_t , as seen in Figure 4.13a.

$$\hat{\mathbf{y}}_{t+1} = \mathcal{N}([\mathbf{y}_t, \mathbf{u}_t]) \quad (4.16)$$

Note that the neural network \mathcal{N} determines the next system state. One possible motivation for combining external inputs and system states into one network input is the existence of parameter-varying systems, where the influence of external inputs depends on the current system state. It is important to highlight, that with this approach, the neural network cannot distinguish between external inputs and system states. Hence, an obvious alternative is to use two separate networks, \mathcal{N}_a and \mathcal{N}_e , for the autonomous and the forced parts of the dynamics and obtain the next system state by adding the outputs, as seen in Figure 4.13b.

$$\hat{\mathbf{y}}_{t+1} = \mathcal{N}_a(\mathbf{y}_t) + \mathcal{N}_e(\mathbf{u}_t) \quad (4.17)$$

The two-network approach is particularly suitable for systems in which the influence of the external inputs is independent of the system state because it enforces the separation into two independent neural network models.

From a system identification and control theory perspective, both architectures can be referred to as *state-space models* (SSM) [KS14; SL19; SN17; KWV+06]. The authors of [KSS16b; HLF+18; RSG+18; SWL+16] apply neural networks to model non-linear SSMs in a modeling paradigm we refer to as *neural state space models* (NSSM). There is research into the combination of neural networks with traditional, linear state transition dynamics yielding Hammerstein [OGJ+16b] and Hammerstein-Wiener architectures [JY08]. The authors of [FP20] use linear operators representing transfer functions as layers in neural networks. To address the issue of partially observable dynamics, the authors of [GWS+20; MB18] leverage encoder/decoder architectures. The work of [SVW+21; SDT20] and [DTC+20] showcases how to utilize physics-constraints for NSSM in a gray-box modeling approach. The authors of [OGJ+16a] compare traditional non-linear regressive models with neural architectures for non-linear system identification and conclude that neural networks are more salable and more straightforward to train.

4.8.3.2 NODEs with External Input

To be able to consider time-varying external input to NODEs, the authors of [DDT19; NBD+20] propose the introduction of additional, augmented variables into the state space. The authors of [MPP+21] describe a more general approach that uses additional neural networks to explicitly model the input dynamics. The fundamental challenge with external input to NODEs is that an adaptive step size numerical solver typically chooses the step size based on the dynamics of the derivative function. Hence, the solver's sampling may not coincide with the sampling of the external inputs, which can require interpolation of the external inputs to align them.

It is possible to represent external input as static parameters. For the pendulum, for instance, the length of the pendulum could be an external parameter of the model that remains constant throughout one simulation but can be changed for another one. The authors of [LP21] describe this approach as *parameterized* NODEs and use it to solve PDEs. In [KCL20], the authors apply the mathematical concept of *controlled DEs* in *neural controlled differential equations* (NCDEs), where the external inputs are modeled as a signal that drives the evolution the system state and the response of the system to the signal is approximated with a neural network.

4.8.4 Network Architectures

One of the reasons for the success of neural networks is their ability to accommodate domain-specific architectures. In this section, we describe (i) how to enforce energy-conserving dynamics with *Hamiltonian* and *Lagrangian* mechanics, (ii) showcase deep potential networks for energy conservation, and (iii) integrate graphs with time-stepper models to solve problems that can be represented as graphs naturally.

4.8.4.1 Hamiltonian and Lagrangian Networks

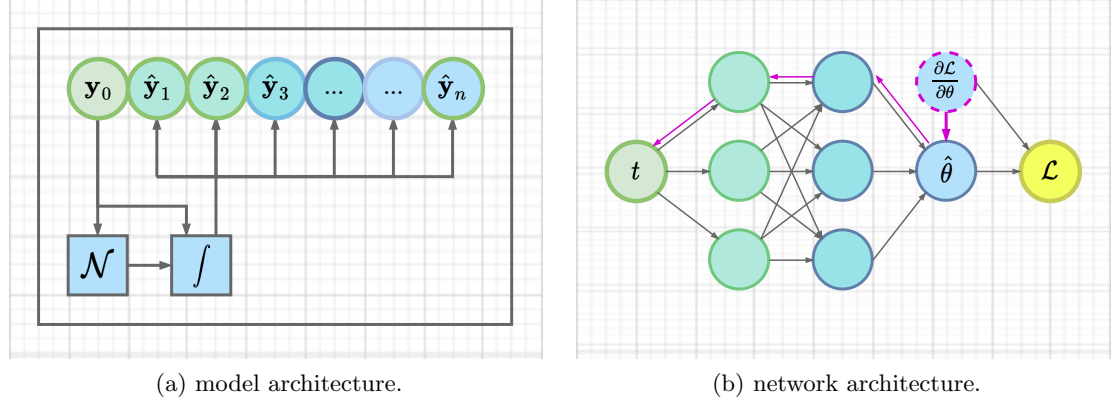


Figure 4.14: Lagrangian time-stepper. The Lagrangian, \mathcal{L} , is differentiated using AD to obtain the derivative of the system state.

In energy-conservative systems, actions happen due to energy transfers within the system. A frictionless pendulum with no external forces acting upon it, for instance, oscillates indefinitely without changing its total energy. Hamiltonian and Lagrangian mechanics are formulations of classical mechanics describing a system's energy. The Hamiltonian \mathcal{H} and Lagrangian \mathcal{L} are functions summarizing the kinetic T and the potential energy V of the system.

We define the Hamiltonian in terms of the concatenated state vector $\mathbf{x} = [\mathbf{q}, \mathbf{p}]$ consisting of generalized coordinates \mathbf{q} and generalized momenta \mathbf{p} .

$$\mathcal{H}(\mathbf{x}) = T(\mathbf{x}) - V(\mathbf{x}), \quad (4.18)$$

From the gradients of the energy function $H(\mathbf{x})$, we obtain a differential equation that includes a symplectic matrix S .

$$\dot{\mathbf{x}} = S \nabla \mathcal{H}(\mathbf{x}), \quad (4.19)$$

The difference between the Hamiltonian \mathcal{H} and the Lagrangian \mathcal{L} is the coordinate system; We define the concatenated state vector \mathbf{x} for the Lagrangian as $\mathbf{x} = [q, \dot{q}]$, where $\dot{p} = M(q)\dot{q}$, with $M(q)$ being a generalized mass matrix.

While the Hamiltonian and Lagrangian functions are mathematically elegant, the process of deriving them analytically can be grueling, especially for complex dynamical systems. There has been a rising interest in data-driven approximations of scalar-valued energy functions in recent years [LRP19; GDY19; ZDC19]. The key objective is approximate the Hamiltonian or Lagrangian by having a neural network to estimate the system states, differentiating these estimates with respect to the state variables and plugging the derivatives into Equation 4.18, as seen in Figure 4.14. The main advantage of

Hamiltonian [GDY19; TRJ+20] and Lagrangian [CGH+20; LRP19] neural networks is that the preservation of energy is ensured by the network architecture.

Research into the simulation of energy-preserving systems has led to the design of *symplectic integrators* for Hamiltonian systems and the subsequent development of specialized, *symplectic neural network* architectures [JZK+20].

4.8.4.2 Deep Potential Energy Networks

The concept of *deep potential energy networks* (DPEN) is similar to the idea behind Hamiltonian and Lagrangian networks, i.e., to develop a neural surrogate for the potential energy function $V(x)$ of a dynamical system. The distinguishing factor is that for DPENs, the kinetic energy terms are encoded explicitly into the time-stepper model through classical Newtonian laws of motion.

$$\tilde{\mathbf{x}}_{i+1} = \tilde{\mathbf{x}}_i + \tilde{\mathbf{v}}_i, \quad (4.20a)$$

$$\tilde{\mathbf{v}}_{i+1} = \tilde{\mathbf{v}}_i - \frac{\nabla \mathcal{V}(\mathbf{x})}{\mathbf{m}} \quad (4.20b)$$

In Equation 4.20, \mathbf{x}_i and \mathbf{v}_i are the position and velocity vectors of the system, the gradients of the potential function correspond to the interaction forces $F = -\frac{\nabla \mathcal{V}(\mathbf{x})}{\mathbf{m}}$, and \mathbf{m} is a vector of masses.

This modeling approach is commonly used for the simulation of molecular dynamics [Beh15; JLG16; WOW+19; UM18; UM19; ZHW+18a], and replaces quantum-chemistry calculations, such as density functional theory. In comparison to directly estimating the dynamics with a high-dimensional time-stepper model, approximating the scalar-valued potential function constitutes a significantly simpler regression problem. In addition, DPEN allow for the integration of prior knowledge into the architecture of the deep potential functions, for instance, by considering only local interactions between atoms [SKS+17], or by encoding spatial symmetries [GRI+20; ZHW+18b]. As a consequence, DPEN are remarkably scalable, enabling supercomputers to simulate up to 100 million atoms [JWC+20], while naive time-steppers would have to learn a mapping of 300 million dimensions.

4.8.5 Graph Time-Steppers

Many real-world systems, such as power grids, social networks, or molecular structures, can be represented as graphs. *Graph neural networks* (GNNs) leverage this fact by embedding or approximating the graph structure from the training data. GNNs represent a research interest in their own right, and we refer the reader to [BBC+21; BHB+18; ZCH+20; WPC+19; ZCZ18; SGT+09] for more general information on their structure and application. In the scope of this survey, we discuss how to use GNN-based time-stepper models to simulate dynamical systems [KFW+18; LWT+18].

The basic idea of utilizing GNNs in time-stepper models, i.e., to use a GNN pipeline to estimate the derivatives. The GNN pipeline consists of three steps: (i) encoding the current state of the system as a graph, (ii) processing the graph to obtain an updated, encoded system state, and (iii) decoding the updated system state.

Initial applications for GNN-based models include *interaction networks* [BPL+16] and *neural physics engines* [CUT+16] for small-scale dimensions, such as n-body problems, rigid-body collision, and non-rigid dynamics. After that, GNNs have been used in NODEs [SBC+19] to include control inputs [SHS+18; LWZ+18], dynamic graphs [RCF+20], and feature encoders that enable models to learn the dynamics of a system from visual signals [WZW+17]. State-of-the-art GNNs are typically trained using message-passing algorithms developed for quantum chemistry applications [GSR+17]. Message-passing aggregates the values of latent physical quantities, such as positions, charges, or velocities, and forwards them through the edges of neighboring nodes of the GNN. Thus, the message-passing algorithm efficiently encodes local, structure-preserving interactions from the real world. While initial implementations of GNN-based time-stepper models exhibited significant computational complexity, recent studies show that GNNs are, in fact, remarkably scalable [SGP+20] and adjust well to large dynamical systems with thousands of state variables and long prediction horizons.

4.8.6 Uncertainty

In the previous sections, we considered deterministic systems for which we had access to noise-free training data. In many real-world scenarios, however, it is likely that the measured states of a system do not correspond to the true system states because of measurement noise. Additionally, there are systems whose dynamics exhibit a certain degree of randomness, e.g., through unknown external forces acting on the system. For instance, the dynamics of a pendulum may be altered by environmental vibrations. In the subsequent sections, we discuss model architectures that explicitly incorporate uncertainty.

4.8.6.1 Deep Markov Models

Deep Markov models (DMM) [KSS16c; MAA19; AR18; LHC+19; FSP+16] combine neural networks with the formalisms of Markov models. The basic idea behind DMMs is similar to the concept behind state space models and employs a latent variable to model the underlying dynamics of the system. The key difference to state space models is that the mapping between the latent variable and the observable system state is probabilistic and that the evolution of the latent variable itself is not fully deterministic.

The relationship between the latent state vector \mathbf{z}_i and the observable state vector \mathbf{y}_i can be described by Equation 4.21.

$$\mathbf{z}_{i+1} \sim \mathcal{Z}(\mathcal{N}_t(\mathbf{z}_i)) \quad (\text{Transition}) \quad (4.21)$$

$$\mathbf{y}_i \sim \mathcal{Y}(\mathcal{N}_e(\mathbf{z}_i)) \quad (\text{Emission}) \quad (4.22)$$

\mathcal{Z} and \mathcal{Y} represent probability distributions (typically Gaussian distributions) whose parameters are approximated by the neural networks $\mathcal{N}_t(\mathbf{z}_i)$ and $\mathcal{N}_e(\mathbf{z}_i)$ (the mean and variance in case of a Gaussian distribution). Note that estimating the characteristic parameters is a convenient way to represent the probability density functions in actual numerical values. Figure 4.15 shows the inference step of a DMM.

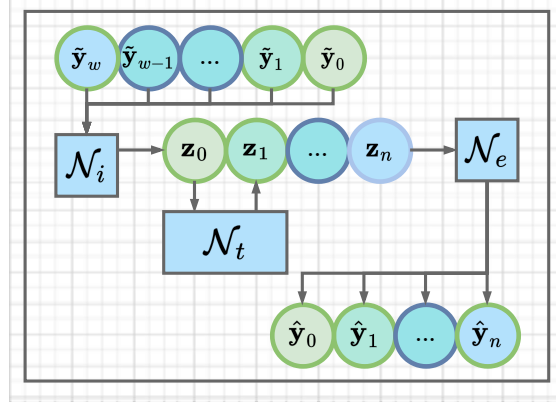


Figure 4.15: Deep Markov model with inference network. Based on a set of observed values $[\tilde{\mathbf{y}}_w, \tilde{\mathbf{y}}_{w-1}, \dots, \tilde{\mathbf{y}}_0]$, the inference network \mathcal{N}_i estimates a latent representation of the trajectory \mathbf{z}_0 . The transmission network \mathcal{N}_t maps the latent state forward in time to obtain a simulation trajectory in the latent space to produce a latent trajectory $[\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_n]$. The emission network \mathcal{N}_e maps each estimated system state from the latent space into the original, observable state space. The output of each network is a set of parameters for a probability distribution instead of an actual numeric estimate. Hence, we need to sample the distribution to obtain a value that can be fed to the next stage of the model.

Regarding the training of DMMs, there is the issue that we only have data for the observed state of the system \mathbf{y}_i but not for the latent state \mathbf{z}_i . A means to address this matter is to use *variational inference* (VI), a universal method to estimate the parameters of probability density functions from data. For a more detailed discussion of this approach, we refer the reader to [KSS16c]. To train DMMs, it is also necessary to adapt the backpropagation algorithm. Fortunately, many popular open-source libraries, such as TensorFlow Probability [DLT+17], or the Pytorch-based Pyro [BCJ+19] library, provide a stochastic counterpart [RMW14] to the backpropagation algorithm.

The authors of [RM16; GHL+21] replace classical Gaussian distributions with more expressive and more computationally expensive deep normalizing flows. The work

in [QBT19] combines DMMs with graph structures to encode inductive biases. DMMs can be used in a wide variety of applications from climate models [CPL+18], through molecular dynamics [WMP+18] to uncertainty quantification in generic time series models [MAL15].

4.8.6.2 Latent Neural ODEs

Latent neural ordinary differential equations (latent NODEs) [CRB+19] extends NODEs to allow a separation between the discretization of the observations and the representation of the dynamics in a continuous-time, generative approach. To do so, latent NODEs apply an encoder network \mathcal{N}_e that obtains a unique, latent representation \mathbf{z}_0 that determines each simulation trajectory from a set of observations. It is convenient to use recurrent network architectures to encode the series of observations because they can handle a variable number of input samples. From the local, initial latent state \mathbf{z}_0 , an ODE solver can generate a trajectory of latent states that is then mapped back into the observable space by a separate decoder network \mathcal{N}_d to obtain the actual simulation trajectory. A schematic representation of this pipeline is depicted in Figure 4.16.

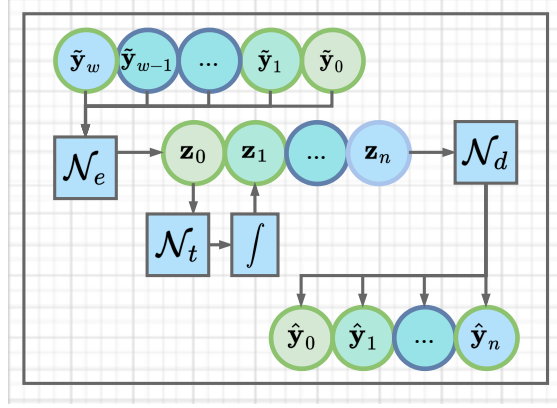


Figure 4.16: Latent NODEs. The encoder network \mathcal{N}_e estimates a latent representation of the system’s initial state from a set of observations of the system $[\tilde{\mathbf{y}}_w, \tilde{\mathbf{y}}_{w-1}, \dots, \tilde{\mathbf{y}}_0]$. Similar to a regular NODE, a neural network and a numerical solver estimate a simulation trajectory of the system, albeit in the latent space $[\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_n]$. The latent solution is then mapped back into the original state space by the decoder network \mathcal{N}_d .

Latent NODEs treat the state variables as continuous, allowing them to generate simulation trajectories with discretizations that are different from the training data’s. By separating the observed system state \mathbf{y}_i from the dynamics, we can apply wider neural networks capable of generating more complex latent trajectories. This separation, however, ensues an inference problem that requires estimating unknown initial conditions of the hidden states for both deterministic [LKS15; SVW+21] and stochastic time-steppers [KSS15; LKS15; KSS17; CKD+15].

4.8.6.3 Bayesian Neural Ordinary Differential Equations

Bayesian neural ordinary differential equations (BNODEs) [DCD+21] introduce stochastic *Bayesian neural networks* (BNN) [JBB+20] into NODEs. *Bayesian* characteristic of BNNs refers to the network's parameters being expressed as probability density functions rather than exact numeric values. The network's weights, for example, can be approximated by a multivariate Gaussian distribution. To obtain a simulation trajectory, we parameterize the model with a numeric value drawn from the distribution.

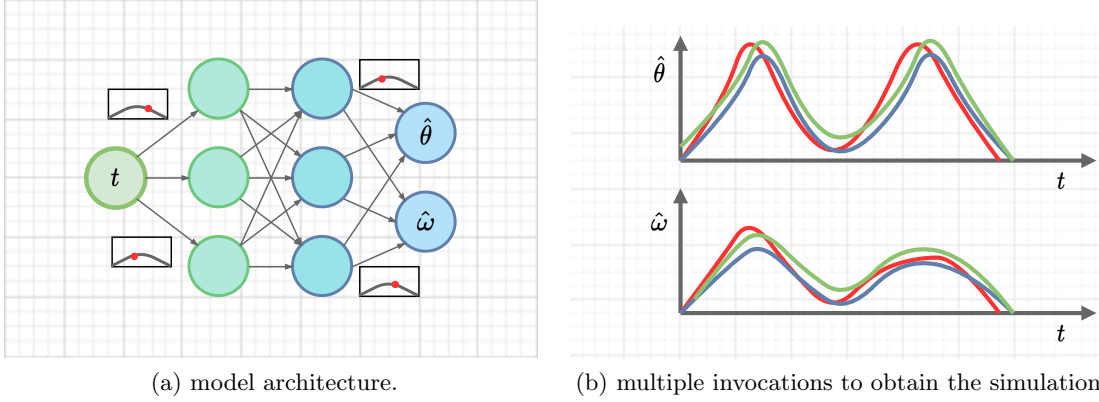


Figure 4.17: Bayesian Neural Ordinary Differential Equations (BNODEs). In BNODEs the parameters of the neural network are described by probability distributions. The distributions are typically sampled multiple times and simulated for each value, which results in multiple trajectories of the same system simulation. We can use this set of trajectories to express uncertainty bounds of the system or average them to obtain a single simulation trajectory.

With this approach, it is possible to represent an estimate of the uncertainty in the model's output by generating multiple trajectories using different samples drawn from the distribution. The inference process can be seen in Figure 4.17. From the set of trajectories, we can infer confidence bounds and a mean.

The main issue of BNNs and their application in BNODEs is that they require specialized training algorithms that do not scale well to large networks. In [JBB+20, Sec 8], the reader can find an overview of alternative methods to introduce sources of stochasticity that do not require specialized training algorithms.

4.8.6.4 Neural Stochastic Differential Equations

The term *stochastic differential equations* (SDEs) refers to ODEs that include stochastic terms in their dynamics. Like Markov processes, SDEs often include a deterministic drift

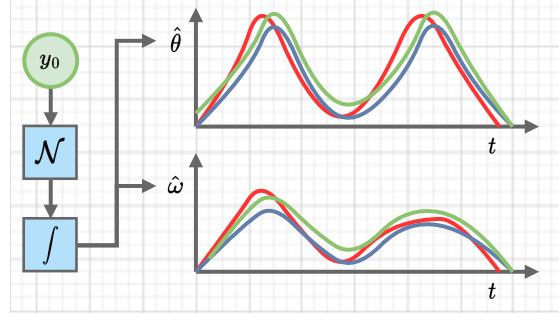


Figure 4.18: Neural stochastic differential equations (NSDEs). In this example we use a neural network N to approximate the deterministic drift term of the SDE and model the diffusion term as a Wiener process. To account for different realizations of the stochastic Wiener process, we solve the SDE multiple times and obtain multiple simulation trajectories.

term and a stochastic diffusion term. This stochastic "white noise" is commonly described by a derivative of the continuous-timed *Wiener Process*, shown in Equation 4.23

$$dX = f(x(t))dt + g(x(t))dW_t. \quad (4.23)$$

SDEs are frequently expressed in *differential* rather than derivative form because stochastic processes are often continuous but not differentiable. Expressing Equation 4.23 as an integral equation, we obtain Equation 4.24.

$$y(t) = y_0 + \int_0^t f(y(s))ds + \int_0^t g(y(s))dW_s. \quad (4.24)$$

Just like many ODEs, SDEs often do not have an analytical solution. However, to solve an SDE numerically, we need different algorithms than to solve deterministic ODEs. As the issue of solving SDEs is quite comprehensive, we refer the reader to [KP92, Chapter 9] for a more extensive discussion.

In *neural stochastic differential equations* (NSDEs) [LXS+19b], we can consider SDE solvers as a black-box method to simulate systems with stochastic dynamics. There are three principal approaches to integrating neural networks into SDEs. If the stochastic diffusion term is known, the network can approximate the deterministic drift term [OVV20; LXS+19b]. Alternatively, we can use the network to parameterize both the drift and the diffusion term [HHL+18]. The authors of [XCL+21] incorporate BNNs into NSDEs and model the evolution of the system states and the network parameters as SDEs.

NSDEs are a powerful tool for modeling uncertainty, however, they are inherently more complex than their deterministic counterparts. Hence, it is a key research interest to examine more computationally efficient mechanisms such as injecting noise or dropouts in NODEs can achieve similar effects as a fully stochastic approach.

Table 4.1: Comparison of direct-solution and time-stepper models.

name	advantages	limitations
direct-solution	<ul style="list-style-type: none"> + can be applied to PDEs easily + no discretization of temporal and spatial coordinates + no accumulation of error during simulation + parallel evaluation during simulation 	<ul style="list-style-type: none"> - fixed initial state - fixed temporal and spatial domain - difficult to incorporate inputs
time-stepper	<ul style="list-style-type: none"> + variable initial states + variable temporal and spatial domain + leverage knowledge from numerical simulation + can incorporate inputs easily 	<ul style="list-style-type: none"> - difficult to apply to PDEs - accumulation of error during simulation - no parallel evaluation during simulation

4.9 Discussion

Given such a wide choice of model architectures, the obvious question is how to pick the most suitable one for a given application. The most fundamental distinction categorizes the models into (i) the ones where a neural network approximates the solution directly, as described in Section 4.7, and (ii) the ones where a neural network approximates the dynamics of the system, as described in Section 4.8. Both approaches have their inherent advantages and disadvantages. Table 4.1 provides a comparison between these two modeling paradigms.

In the literature, different implementations of direct-solution and time-stepper architectures often appear to be profoundly different models. In reality, however, many architectures are closely related. This observation becomes apparent once the different realizations are applied to solve the same problem, e.g., estimate the dynamics of an ideal pendulum. For direct-solution models, the differences are often caused by different approaches to integrate prior knowledge about the system we want to approximate. For instance, the difference between the vanilla solution and the models using automatic differentiation (AD) is the way we estimate state variables that are related to each other, i.e., in the AD solution, we apply automatic differentiation to obtain a state variable that we know to be the derivative of the other. PINNs combine automatic differentiation with physics-based regularization in the loss function.

For time-stepper models, the differences can be expressed in terms of their network architectures and the numerical integration scheme. Consider, for instance, the close resemblance between a residual time-stepper and the Euler time-stepper. However, the ability to tune the network architecture to the problem the model is supposed to solve and the flexibility to choose different numerical solvers, allow time-stepper to simulate a wide range of physical phenomena.

The ideas behind different models can be transferred to each other, which facilitates the development of novel architectures and approaches. Hence, it is challenging to define guidelines for which models should be applied for which application without limiting the models' potentials. Instead, we suggest that the reader carefully considers the

characteristics of the application and how prior knowledge or inherent structures can be leveraged with each model architecture.

4.9.1 Open Questions

Benchmarking the model architectures on a dataset with various dynamical systems would be extremely useful. A benchmark could provide practitioners with a starting point when selecting models for specific applications. However, it can be difficult to compile a representative collection of systems from different disciplines, such as physics, chemistry, and engineering. Additionally, it would be essential for the dataset to provide a fair comparison between the different architectures of network-based models, as well as between network-based approaches and traditional simulation methods.

Similarly, the research field would benefit from a standardized procedure to evaluate a model's ability to approximate a dynamical system for all possible initial conditions. In safety-critical applications, for instance, it is essential to ensure that a simulation model provides a reasonable estimate of the system state for all operational conditions. Given the inherent diversity of dynamic systems, it stands to reason that some will be more difficult to approximate than others. An established validation procedure would help researchers to understand the architectures' capabilities and limitations. The development of standardized validation procedures is closely related to the selection of error metrics for dynamical system simulation, which is another interesting research topic in itself.

While many model architectures apply slight variations of the same basic ideas, the training process is often more nuanced. Hence, rule-of-thumb-like guidelines for the training process and the requirements on quantity and quality of the training data would be a useful contribution to the field of data-driven simulation of dynamical systems. These guidelines could help practitioners decide whether a data-driven modeling approach is feasible in a given situation and which models are most suitable. Additionally, it would be worthwhile to investigate the best way to approach the training/test/validation split and how to prepare trajectories for training. For time-steppers, for example, trajectories can be split in order to balance training time and accuracy. Training guidelines have to consider the choice of loss functions and error metrics, which are closely related to the topic of model validation.

5 Co-Simulation

Many complex, real-world systems consist of distinct, interactive components and sub-systems. In the previous chapters, we used ML methods to implement a monolithic simulation model, and in many instances, monolithic models can provide a sufficiently accurate representation of the system behavior. Since complex cyber-physical systems often combine components from various engineering disciplines, there is a plethora of component models. With the paradigm of co-simulation, we can leverage the strengths of different modeling tools, environments, and formalisms [CLB+19] introduced by partial system models. To combine partial models into a system simulation, we have to integrate different notations and toolchains [GTB+19]. For a definition of co-simulation terminology and a comprehensive overview of co-simulation frameworks, we refer the reader to [GTB+17] and [GTB+19]. This chapter is based on in [LTS+21b; TCD+21] and [ASF+23]. In the subsequent sections, we discuss coupling mechanisms and frameworks for co-simulation and present an open-source tool to generate simulators that implement the *functional mockup interface*.

5.1 Coupling Techniques

There are three principal coupling techniques for coupling different simulators [TFL+21] into a co-simulation:

- One-to-one coupling: Communication between different simulators requires explicit implementation of the connections. S-function, for instance, connects MATLAB function blocks with Simulink models, and TRNSYS 155 connects MATLAB to the TRNSYS environment.
- Middleware coupling: Employing dedicated middleware is a very flexible means of coupling simulators. Typically, the middleware orchestrates the simulation and handles data exchange between the simulators. The Building Control Virtual Test Bed environment [Wet11], for example, allows for the run-time coupling between EnergyPlus, MATLAB, Simulink, and Dymola and provides additional post-processing functionality such as visualization.
- Standard interface coupling: The formalization of standardized co-simulation interfaces allows for direct coupling between all simulators that implement the same interface. The Functional Mock-up Interface (FMI) and the High-Level Architecture (HLA) are well-known examples of co-simulation standards.

Standard interfaces provide a flexible framework for co-simulation. Given the growing popularity and capability of data-driven models there is an increasing interest in integrating them with more traditional simulation tools.

5.2 The Functional Mockup Interface

The *functional mockup interface* (FMI) has become a popular standard for continuous time, discrete event, and hybrid co-simulation [SGE+19]. The popularity of the FMI standard has led the developers of modeling tools to integrate FMI compatibility into their software. For instance, it is possible to create, export and import functional mockup units (FMUs) based on modeling languages such as Modelica or MATLAB using *OMEdit* [AT10], *Simulink* [Doc20] or *20-sim* [BV13]. While these tools cover a large share of traditional modeling applications, there is a rising interest in specialized, purpose-built FMUs. Creating an FMU from scratch, however, is cumbersome and requires:

- a detailed understanding of the FMI standard
- the code to be implemented and compiled in a C-compatible language
- cross-compilation to support all target-platforms
- manual creation and synchronization of the top-level model description file
- correct packaging of all assets

These requirements have spawned a series of libraries that support the manual development of FMUs. Each of these libraries typically supports one specific programming language, meaning that a user has to install and understand multiple libraries to use different programming languages. Because the FMI standard is based on the C language, FMUs generally have to be implemented in C or a C-compatible language such as C++, Rust, or Fortran. Hence, most libraries and tools favor languages derived from C or languages where interoperability with C is easy to implement. Table 5.1 presents a selection of libraries for FMU development and simulation.

In addition to development tools, some frameworks allow users to combine the functionality implemented by an FMU with other, user-developed code. *FMI++* and *fmpy*, for instance, enable users to import and simulate FMUs in a Python environment and insert additional Python code into the simulation loop without having to repack the FMU. *FMU-proxy*, introduced in [HSH+19], provides a client/server architecture that separates FMI's C-API from the actual implementation of an FMU. So-called "proxy servers" host a set of FMUs and allow clients to access these FMUs through language-agnostic, cross-platform *remote procedure calls* (RPC). Servers support *model exchange* (ME) and *co-simulation* (CS) but support only CS through their API. Depending on the server implementation, clients can choose the solver that a server uses for wrapping the ME models. Clients can be implemented in any language that can wrap the required RPCs. To generate a "proxified" version of an FMU, the authors provide a command line tool called *fmu-proxify*.

Table 5.1: Overview of libraries to import and simulate and/or export FMUs.

	language	import	export
FMUSDK	C	yes	yes
FMI++ [WME+13]	C/C++	yes	yes
FMI4j [HZS+18]	Java	yes	yes
JavaFMI [EH14]	Java	yes	yes
JFMI	Java	yes	
PythonFMU	Python		yes
PyFMU	Python		yes
OvertureFMU	VDM		yes

In [LTS+21a] and [SMT+21] we introduce the a tool called *universal functional mockup unit* (UniFMU), which allows users to build FMUs from arbitrary code in any programming language; it supports Python, C#, and Java out of the box. The tool uses a precompiled binary wrapper that implements the methods specified in the FMI standard's C-headers to spawn a process that executes the FMU's actual code. UniFMU uses RPCs, similarly to FMU-proxy, however, UniFMU ships with a set of template FMUs in three popular languages and provides other additional features such as the possibility to select virtual environments for running the FMU or to containerize the FMU in a Docker container.

5.3 Practical Example

To exemplify the process of using UniFMU, we consider the case of modeling a robotic arm coupled to a controller, as depicted in Figure 5.1. The example demonstrates how to employ UniFMU to implement different modeling formalisms, i.e., how to simulate a continuous robot and a discrete controller.

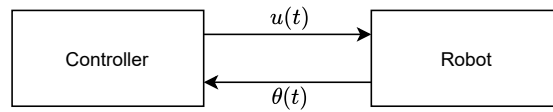


Figure 5.1: Connection between controller and robot model.

In the subsequent sections, we describe the mathematical concepts behind the robot arm and its controller, present an Python implementation of a model of the robot arm and a PID controller, and showcase how generate and simulate two FMUs wrapping both components. Additionally, we discuss the basic functionality of the UniFMU tool and how to use virtualization to package runtime dependencies.

5.3.1 Robotic Arm

We model the robotic arm as a controlled inverted pendulum and describe its state by its angle θ , the angular velocity ω , and the current running through the coils of the electrical motor i moving the arm. We can describe the dynamics of the robotic arm by Equation 5.1.

$$f(x) = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{i} \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{K \cdot i - b \cdot \omega - m \cdot g \cdot l \cdot \cos(\theta)}{J} \\ \frac{u \cdot V_{abs} - R \cdot i - K \cdot \omega}{L} \end{bmatrix} \quad (5.1)$$

The derivative of the angle $\dot{\theta}$ is, per definition, equal to the velocity of the arm ω . The derivative of the angular velocity $\dot{\omega}$ is determined by the torque coefficient $K = 7.45 \text{ s}^{-2} \text{ A}^{-1}$, the current i , the motor-shaft friction $b = 5.0 \text{ kg} \cdot \text{m}^2 \cdot \text{s}^{-1}$ and the gravity acting on the arm, denoted by $m \cdot g \cdot l \cdot \cos(\theta)$, with $m = 5.0 \text{ kg}$, $g = 9.81 \text{ ms}^{-2}$, $l = 1.0 \text{ m}$. The change in current is determined by the input from the controller u , the voltage across the coils $V_{abs} = 12.0 \text{ V}$, the resistance $R = 0.15 \text{ } \Omega$ and the motor's inductance $L = 0.036 \text{ H}$.

5.3.2 Controller

To control the robot arm, we use a proportional-integral-derivative (PID) controller [ÅM10] and describe its continuous formalization as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \dot{e}(t) \quad (5.2)$$

The function $e(t)$ estimates the error between the desired and the actual value of the variable we want to control. The terms K_p , K_i , and K_d are coefficients that weigh the proportional and derivative terms. For our example, the controller minimizes the error between the desired angle $\theta^*(t)$ and the true angle $\theta(t)$ of the robot arm. We can define the error function as $e(t) = \theta^*(t) - \theta(t)$.

In practice, most controllers are digital. Hence, the derivatives and integrals in Equation 5.2 have to be replaced by discrete approximations. There are several ways to do this; the simplest is to replace derivatives by first-order differences, as seen in Equation 5.3a and the integrals by sums, as seen in Equation 5.3b.

$$\dot{e}(t_k) \approx \dot{e}_k = \frac{e_k - e_{k-1}}{T}, \quad (5.3a)$$

$$\int_0^{t_k} e(t_k) \approx E_k = \sum_{n=1}^N e_{k_n} \cdot T \quad (5.3b)$$

$e_k = e(t_k)$, T is the sampling time and $N = t_k/T$ is the number of samples between time 0 and t_k . After replacing the continuous definitions in Equation 5.2, we obtain a discrete approximation, shown in Equation 5.4, that we can implement on a discrete controller.

$$u_k = K_p e_k + K_i E_k + K_d \dot{e}_k \quad (5.4)$$

5.3.3 Robotic Arm FMU

The robotic arm FMU is implemented using a numerical solver provided by the *SciPy* [VGO+20] Python package. The general procedure for solving an ODE using Scipy is to define a function that evaluates the derivative for a given combination of state and time. Using Equation 5.1 as a reference, we can define the function $f(\cdot)$ as shown in Listing 5.1.

Listing 5.1: Implementation of the `fmi2DoStep` method for the robotic arm FMU.

```

1  def do_step(self, current_time, step_size, no_step_prior):
2      def f(t, y):
3          theta, omega, i = y
4          tau=self.k1*np.cos(theta)
5          domega=(self.K*i-self.b*omega-tau)/self.J
6          di=(self.V-self.R*i-self.K*omega)/self.L
7          dtheta=omega
8          return dtheta, domega, di
9      res=solve_ivp(f, (current_time, current_time+step_size), y0)
10     self.theta, self.omega, self.i=res.y[:, -1]
11     return Fmi2Status.ok

```

With the definition of the derivative $f(t, y)$, we can use `solve_ivp` to obtain the solution for the next state. This implementation is remarkably flexible, as `solve_ivp` allows us to specify any numerical solver provided by the Scipy package or to implement a custom solver to numerically integrate the ODE. Per default, the function uses a Runge-Kutta integrator. With the solution to the IVP, we can assign the newly estimated state to the FMU, where other methods, orchestrators, and FMUs can access it.

An advantage of UniFMU's approach of separating the FMU code from the FMI interface is that it can be tested and debugged with standard testing and debugging tools. For instance, to test the implementation of the robotic arm's dynamics, we can write a small test program in Python. The test program invokes the `do_step` several times, starting at an initial state of $\theta_0 = \omega_0 = i_0$, with a constant control input of $u(t) = 0$. You can see the angle of the arm decreasing from 0 to -1 in Figure 5.2 over 10 seconds.

5.3.4 Controller FMU

The controller implements a simple algorithm that estimates the control signal for the motor based on the difference between the desired and the actual angle. Similarly to how Equation 5.1 was translated into Python code describing the derivative of the state, we

5 Co-Simulation

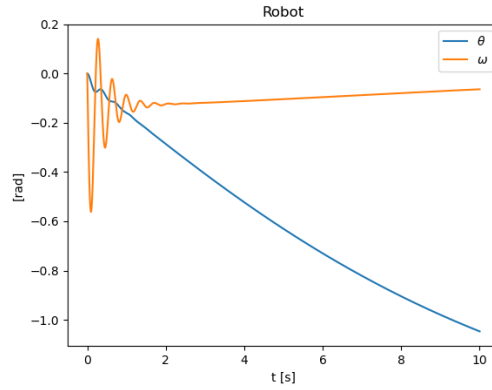


Figure 5.2: Standalone test of robotic arm, with values $\theta_0 = \omega_0 = i_0 = u(t) = 0$.

use the control policy Equation 5.4 as a reference to implement the controllers' `do_step` function shown in Listing 5.2.

Listing 5.2: Implementation of the `fmi2DoStep` method for the controller-FMU.

```

1  def do_step(self, current_time, step_size, no_step_prior):
2      err=self.setpoint_t-self.measured_t
3      self.I=self.I+err*step_size
4      D=(err-self.p_err)/step_size
5      self.u=self.Kp*err+self.Ki*self.I+self.Kd*D
6      self.p_err = err
7      return Fmi2Status.ok

```

In most practical situations, controllers run on a processing unit where updates to the output happen at a fixed rate determined by the controller's clock frequency and the number of operations needed at each update. An implicit assumption of our example model is that the solver's step size matches the controller's update rate. For small step sizes, the discrete approximation implemented by the model remains relatively accurate. For larger step sizes, the accuracy of the discretization scheme decreases and can ultimately cause the closed-loop system to become unstable. A means to reduce the discretization error is to use a more sophisticated discretization scheme, such as Tustin's method [FPE20]. However, for the sake of this demonstration, we use a sufficiently small step size to mitigate this issue. Alternatively, as for the robotic arm, we can also use any external library, such as the *python-control* [FGM+21], to model the controller instead.

Again, as for the other FMU, we can test our implementation with a small test program that invokes the `do_step` function several times. To examine how the controller behaves in a closed-loop simulation, we use a simple linear model, described by the ODE $\dot{\theta} = u$, as a surrogate for the robotic arm. Executing the Python test program, we obtain the step-response depicted in Figure 5.3 from the closed-loop system.

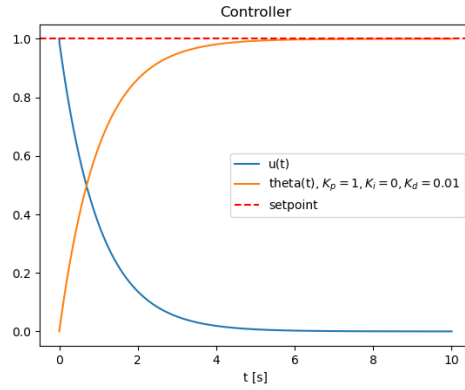


Figure 5.3: Standalone test of the controller FMU with using linear model for plant $\dot{\theta} = u$, step size = 0.001, setpoint = 1.0

5.3.5 Creating an FMU in UniFMU

UniFMU offers a command line interface (CLI) and can be installed through Python’s package installer *pip*, or from source following the instructions in the official repository at GitHub¹. Note that, even though UniFMU is available through *pip*, the FMUs generated with UniFMU do not require a Python environment during runtime unless the FMUs themselves are implemented in Python. To generate an FMU, we invoke the tool’s subcommand **generate** and supply it with the language the FMU is implemented in and the name it should have; for a Python-based FMU with the name **example** the command is:

```
1 unifmu generate python example
```

This command generates an FMU with the structure shown in Figure 5.4. The **binaries** directory contains a precompiled wrapper for Windows, Linux, and macOS that implements the methods specified in the FMI’s C-headers and relays them to the actual implementation of the FMU found in the **resources** directory. **model.py** defines a class that declares a set of methods that correspond to the ones in the FMI standard, such as FMI’s **fmi2DoStep**, which is implemented by the Python method **do_step**. It is important to note that it is not the binaries that define the behavior of an FMU generated with UniFMU but the file(s) within the **resources** directory. Hence, the binaries are the same for all FMUs, regardless of the language used to implement the FMU’s behavior. The binaries for Linux, Windows, and macOS are pre-compiled and shipped with the tool, so users do not need to set up a compiler toolchain. As of 2022, a GUI is a work in progress.

The basic architecture of an FMU generated by UniFMU separates it into a compiled binary that can be invoked through calls to its FMI-compatible interface and the actual

¹<https://github.com/INTO-CPS-Association/unifmu>

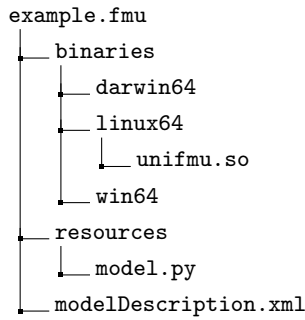


Figure 5.4: The directory structure for a Python FMU. Note that we omit some of the autogenerated files for simplicity.

implementation of the simulation unit. Upon invocation, the binary wrapper spawns a separate process providing a runtime environment to the implementation. This translation layer allows the spawned process to run code in any arbitrary language, including interpreted languages, such as Python, or languages that use automatic garbage collection, such as Java. Message exchange between the wrapper and the implementation happens through RPCs. The example implementation for Python, Java, and C# that ship with the tool use the *gRPC* [gRP] backend, but it is possible to implement the RPCs with other backends as well.

5.3.6 Docker Support

Regardless of the programming language used for the implementation, running the FMU requires the host machine to provide the necessary runtime environment. For a Python-based FMU, for instance, the host machine has to supply a compatible version of the Python runtime and all libraries referenced in the code. Especially when running an FMU on a machine different from the one it was implemented on, manually ensuring that all runtime dependencies are fulfilled can be tedious. One way to alleviate this problem is to virtualize the runtime environment and ship it with the FMU. UniFMU uses the *Docker* virtualization engine to *containerize* the runtime dependencies.

To create a Dockerized FMU, the user can append the `--dockerize` switch to UniFMU's `generate` subcommand, as shown in Listing 5.3. The functionality is available for Linux, Windows, and macOS and all languages that the tool supports.

Listing 5.3: Creating a dockerized FMU with UniFMU.

```
1 unifmu generate python --dockerize robot
```

5.3.6.1 Building a Docker Image

The `Dockerfile` provides instructions to build a snapshot of the virtualization environment, referred to as a *Docker image*. In Listing 5.4, you can see an excerpt from the `Dockerfile` that builds the runtime environment for the FMU simulating the robotic arm. Docker provides a wide range of pre-built images for common environments through its *container image library* called *Docker Hub* ² In the first line in the FMU's `Dockerfile`, we select a pre-built Python 3.8 image from Docker Hub as the base image for the FMU. The second line installs the required Python packages through the package manager `pip`. For simplicity, we replaced the dependencies of the UniFMU python backend with three dots. The third line instructs Docker to copy the `container_bundle` directory into the image. The `container_bundle` contains all files the FMU needs during runtime, such as the actual model implementation and all user-generated files and dependencies.

Listing 5.4: Dockerfile used to assemble the image used by FMU instances.

```
1 FROM python:3.8
2 RUN pip install ... scipy roboticstoolbox-python matplotlib
3 COPY container_bundle resources
4 ...
```

5.3.6.2 Instantiating a Dockerized FMU

Figure 5.5 depicts how the FMU modeling the robot arm is built and instantiated when an orchestration algorithm calls a method in the FMU's binary. The steps are as follows: First, the binary wrapper ensures that the image declared in the `Dockerfile` has been built. If this is not the case, it will automatically invoke the `Dockerfile` to build the image. From the image, the wrapper instantiates a Docker container. The container has access to all dependencies listed in the `Dockerfile`, such as Python packages installed through `pip` and everything inside the `container_bundle`. Note that each instance of an FMU is executed within a separate container and removed after use to ensure that multiple instances of an FMU do not share their state or influence each other in any other way. Per default, the images are not automatically rebuilt on each invocation of the FMU to improve performance. However, it is simple to change this behavior by changing the command in the wrapper's configuration without having to recompile the binary.

5.3.7 Simulation

So far, we have only tested the Python implementation of the simulation models within the FMUs without invoking the binary wrapper. Hence, the next step is to run a simulation using an FMI orchestration algorithm. To do so, we set up a co-simulation test bench, using the *INTO-CPS tool-chain* [LFW+16].

²<https://hub.docker.com/>

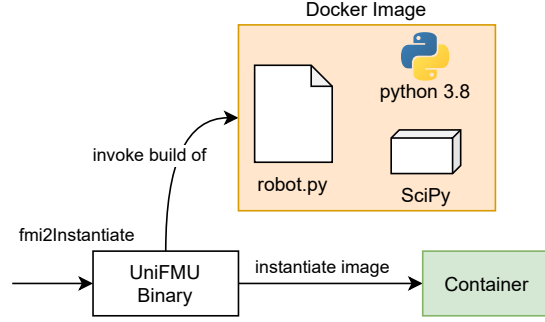
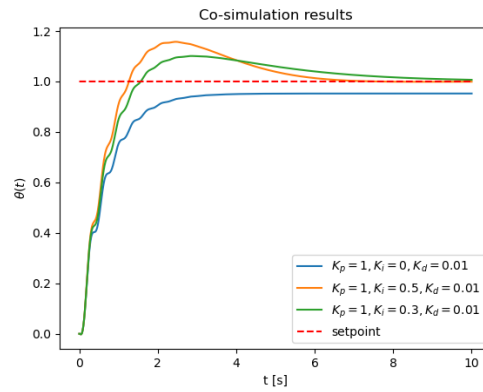


Figure 5.5: Deployment of a model inside the Docker container.

Figure 5.6: Co-simulation of the controller and the robot with the *setpoint* = 1. Experiments of varying the controller parameters are shown.

We use a fixed step-size solver with a step-size of 0.001 seconds, set the desired angle to 1 radian, and plot the arm's angle θ as a function of time for various values of the controller's coefficients K_p , K_i and K_d . You can see the respective plots in Figure 5.6. The simulation where the integral term K_i is set to 0 shows a significant steady-state error, whereas the others converge within 10 seconds. However, it is worth noting that the latter overshoot the setpoint before converging.

We can balance the tendency to overshoot and the size of the steady-state error by tweaking the coefficients according to the application's requirements. Besides the possibility of manually adjusting the coefficients, there are heuristic methods to tune PID controllers to the desired accuracy. However, we consider applying these methods beyond the scope of this demonstration and leave it to the reader to extend and adapt the setup.

5.3.8 Discussion

Co-simulation is a promising paradigm for solving modeling problems in a variety of domains and applications. Co-simulation standards provide a remarkably flexible framework to integrate a diverse set of different modeling tools. The FMI standard is among the most popular interfaces for model exchange and co-simulation.

With the growing popularity of machine-learning frameworks for scripting languages such as Python, the integration of new programming languages into the FMI ecosystem has become a key research interest. However, FMI requires communication through a C-API, which causes difficulties in implementing FMUs in languages that cannot be compiled into C-compatible binaries. UniFMU addresses this issue by providing a generic C-binary that handles all communication between FMI calls to the FMU and the FMU’s actual implementation. Access to high-level programming languages, such as Python, allows developers to leverage a large ecosystem of scientific libraries that significantly simplify the development of simulation models. Consider, for instance, the implementation of a robot arm’s dynamics, shown in Listing 5.1. The ODE is declared and solved in eight lines of code.

Loosening the restrictions on the choice of programming languages and paradigms simplifies co-simulation and causes it to be more accessible to practitioners. Additionally, separating the FMU implementation from its interface to the FMI methods allows developers to use standard debugging and development tools. Consider, for instance, the test programs we used to verify the FMUs in the example setup. In our experience, the ability to test individual models before orchestrating them in a co-simulation significantly reduces the likelihood of encountering issues when integrating the models.

One pitfall of using FMUs that are not compiled is that the host machine has to provide all runtime dependencies. UniFMU addresses this issue by providing a means to virtualize the runtime environment inside a Docker container. With this virtualization layer, the FMU provides almost the same portability as a compiled FMU even if the actual implementation uses a non-compiled language.

While UniFMU provides a convenient, accessible means to implement specialized FMUs, it is clear that the additional translation layers introduce computational overhead. However, the results in [HSH+19] indicate an almost constant per RPC invocation, meaning that reducing the number of invocations, e.g., through grouping subsequent calls to the `fmi2DoStep` method, is a possible means to reduce the overhead. The extent of the performance deterioration and the potential additional penalties through Dockerization would have to be investigated in future research.

Another pitfall of UniFMU’s flexibility is that it does not validate consistency between the model implementation and the model description file. Addressing this issue is not trivial, as it would entail implementing a means of automatically generating code for each target language individually.

6 End-to-End Integration

In the previous chapters, we have often alluded to the requirements and applications of data-driven modeling approaches. Dynamical, cyber-physical systems, such as modern buildings, typically consist of complex, interacting subsystems that generate large quantities of system data and require careful configuration, maintenance, and control to operate efficiently. Hence, they are particularly well-suited to provide us with the data and the use cases for data-driven modeling approaches, hybrid models, and co-simulations.

In this chapter, we showcase the development of an integrated IoT platform that serves as a data source and application use case for the modeling approaches and results we described in the other chapters. The content of this chapter is based on a survey paper on IoT middleware [ASK+22] and a conference paper on the development and integration of an IoT middleware platform that collects operational data and hosts a variety of building automation services [SAH+22].

6.1 Background

Control strategies, monitoring applications, demand and production forecasts, predictive maintenance, and automatic fault detection typically require operational data from the buildings they should be developed for, as well as the technical prerequisites to integrate the services with the building control systems. However, it is often difficult to integrate heterogeneous computational components, such as sensors or actuators from different vendors because of compatibility issues between their communication protocols and data formats; a problem that is typically addressed using middleware platforms as translation layers. While there is a growing number of commercial and non-commercial IoT middleware solutions, there are still comparatively few production-ready software stacks for the energy sector [ASK+22].

In the literature, there are examples of the application of middleware systems that unify the data collection process based on the FIWARE platform [AMS+19; TGR+19a]. FIWARE is designed to solve two problems: i) enable interoperability between devices and subsystems and ii) provide practitioners with open standards to prevent them from falling into a vendor lock. The FIWARE initiative provides a curated set of open-source components for IoT middleware platforms, and while they are designed to work out of the box, integrating advanced services still requires software development expertise, somewhat challenging the benefit of pre-built solutions. The FIWARE components

generally cover a wide variety of applications; however, by trying to be generic, they might not meet all requirements a specific use case presents.

In our work, we take one step behind the generic implementation and develop a minimalist, use-case-specific platform based on the standards, data models, and ontologies used by the FIWARE platform that can be integrated seamlessly with FIWARE components and other software that implements the same standards. This IoT middleware system, denoted *inframonitor*, is used at the Graz University of Technology and covers a wide range of applications from data collection and data persistence to machine-learning-based data analysis services, notification, and visualization services. The software components and configurations are available for practitioners to examine, use and adapt on GitHub¹.

6.2 Technologies and Related Work

The need to efficiently manage scalable IoT infrastructure has generated considerable interest in the development of integrated IoT platforms, especially in the fields of smart cities and smart services. The ultimate goal of such platforms is to enable the connection between multiple services and components, allowing the users to track, manage, and monitor the system service easily.

6.2.1 IoT Middleware Surveys

The authors of [AGM+15] provide an overview of different technologies, protocols, and applications in IoT systems and showcase existing IoT systems. In [FZ20], the reader can find a detailed analysis of 63 IoT management solutions and an evaluation framework that highlights the specifics, advantages, and disadvantages of each approach. In the evaluation process, the authors compile a list of questions to explain the process lifecycle, the modeling activities, and the modeling languages in each solution. The most common commercial and non-commercial, open-source frameworks and solutions for smart cities and smart energy services, such as Apache Kafka, Cisco Kinetic, and FIWARE are described in [ASK+22].

6.2.2 Open Standards

The FIWARE Foundation [FIW21] provides a diverse range of pre-built open-source services and components that can either be used as a single unified platform or as separate, independent components in any other configuration. All FIWARE components adhere to the open protocols, standards, and reference models defined by the *Smart Applications REFerence* (SAREF) [DHR15] ontology and the *Next Generation Service Interface - Linked Data* (NGSI-LD) [Con21] information model and API.

¹github.com/tug-cps/inframonitor-public

The SAREF ontology specifies the building blocks of smart applications, their relations, and interactions, creating a *shared model of consensus* ontology that can be separated and recombined for any specific application in a modular, reusable, maintainable, and extensible manner. The NGSI-LD information model and API, developed by OMA SpecWorks [OMA21] constitute a meta-model that defines the key concepts of linked data and the necessary operations to exchange context information about entities between different components, systems, and stakeholders in applications such as smart cities, smart industries, or any other cyber-physical systems. FIWARE uses a RESTful HTTP API that implements the NGSI-LD context information model [FIW20].

The authors of [HDS+20] utilize FIWARE components to build the *FISMEP* IoT platform for energy services. FISMEP uses the *FIWARE Orion Context Broker* to manage context information and FIWARE's *QuantumLeap* service in combination with a separate time series database to store and access historical, spatial-temporal data. The authors of [CSB+19] present three real-world use cases for the FIWARE platform: i) a global IoT market, ii) analytics in smart cities, and iii) IoT-augmented autonomous driving. They analyze the benefits and the challenges of using FIWARE components, such as the lack of native support for time series data, or the data privacy and security concerns caused by limited control over shared data usage.

In [TGR+19b], the authors develop a multi-layered FIWARE-based IoT platform to manage energy data from sensors in multiple buildings by re-using, adapting, and extending different open-source components in a modular, extensible software stack built around the FIWARE context broker, the FIWARE IoT agent, and the short-time historical data collection service *Comet*.

6.2.3 Graz University of Technology

Most supply systems at the Graz University of Technology have been connected to digital metering devices. However, depending on who installed them, the sensor devices collect data into different information systems that run incompatible software stacks based on conflicting protocols and data models. The operational data was largely unused until 2019/2020 when an interdisciplinary research group on campus developed machine-learning models to monitor water supply [SSP+20] and subsequently investigated methods to predict electrical energy demand [SEL+21] in two exploratory projects.

To use the results for automatic monitoring and control of the system, as well as for additional services such as centralized data collection and data persistence, data visualization, and notification services, it was necessary to implement the appropriate software infrastructure.

In the subsequent section, we describe the software stack we developed, starting from the connections to the sensors, through the two levels of data storage to the service architecture. Additionally, we describe a data analysis service that we use to estimate the energy production on one of the photovoltaic systems on campus.

6.3 Implementation

The components of the system are designed to be modular. Each service, including the databases and data brokers, is containerized using the *docker* virtualization engine. Virtualization achieves three objectives i) portability, which helps with testing and deployment, ii) modularity, which decouples components and allows us to extend and change the system efficiently, and iii) security, because we can easily restrict communication between the components, even when they are deployed on the same server.

6.3.1 Sensors

The sensor devices are operated and maintained by different groups of operators, the BATS operators, a building automation company, an electrical engineering company, and the occupants of one of the offices on campus. Consequently, the sensors connect through different gateways, including BACnet servers, a KNX fieldbus, and a LoRaWAN. The sensors capture energy demand and water consumption, production of the photovoltaic systems, room temperatures, humidity, CO₂ concentration, and for one building, the number of people entering and exiting the building.

6.3.2 Data Collection

We run a *mosquitto* MQTT broker [Lig17] as the primary data collection entry point. The MQTT protocol provides lightweight publish/subscribe message exchange with low bandwidth and low power usage [AGM+15], efficient distribution, secure connections via TLS/SSL, and reliable delivery [Lig17]. The protocol supports a wide range of devices and has been identified as an essential means of communication for IoT platforms by more than 90% of experts in a recent survey [ASK+22].

The tree structure of the MQTT protocol allows us to assign separate top-level branches to each data provider, i.e., all sensors maintained by one group or company are grouped under one branch, named by an *API key*. Grouping IoT devices into services and assigning an API key to each group is an approach also used by FIWARE's IoT agent setup. The key-based grouping provides three advantages for us i) the API key encodes who operates the sensor, ii) devices from different operators can send under the same id; a feature that might be necessary if it is difficult or impractical to change the id a device uses to identify itself, and iii) it requires that senders must have a valid API key to write messages to the broker. Communication between the broker and clients is secured through certificate-based SSL/TLS, with the certificates placed in a directory on the server that is mounted into the docker container running the MQTT broker.

On the server, we subscribe a python-based *paho-mqtt*-client, subsequently denoted *MQTTAgent*, to all updates on the broker. The client runs a demultiplexer that, based on the message origin encoded through the API key, invokes the necessary decoding and

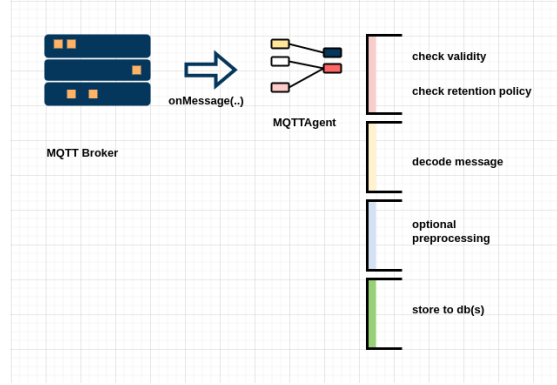


Figure 6.1: The MQTTAgent handling sensor updates.

preprocessing operations to obtain the sensor update for the database. The MQTTAgent does not process MQTT messages without payload. MQTT messages can be *retained*, allowing subscribers to obtain the last update even if they subscribe after the message was sent. The MQTTAgent can be set to skip retained messages or handle them; per default, they are skipped. We support two message formats; *ultralight* and *JSON*.

The ultralight 2.0 format [FIW22] is a lightweight, text-based format of key/value pairs, where keys and values are separated by a pipe symbol. Decoding is straightforward and can be implemented using string splitting and indexing operations in plain python. To decode JSON payloads we use the native library provided by python. The decoded payload is forwarded to a data sink that processes and stores the update in the database(s). A schematic representation of the pipeline can be seen in 6.1.

Besides collecting data from on-site devices via the MQTTAgent, we use another dockerized client, the HTTPAgent, to poll an external web API for the current weather forecast and cache it in our database. We use the weather forecast in many of our ML models, e.g., in the ones predicting PV production. To reduce the number of calls to the weather API, we store the current forecast in our database every hour and have all services, including the ML models, collect it from there.

6.3.3 Data Persistence

To store both ontological and operational information about the building control system, we use two databases: an ontology database, containing a relational representation and the current state (e.g., current sensor values) of the system, and a time series database with historical sensor values. The ontology database is implemented as a MongoDB, a source-available, NoSQL JSON-like document-based database. The time series database is an InfluxDb, an open-source, high-performance database for real-time data collection and retrieval.

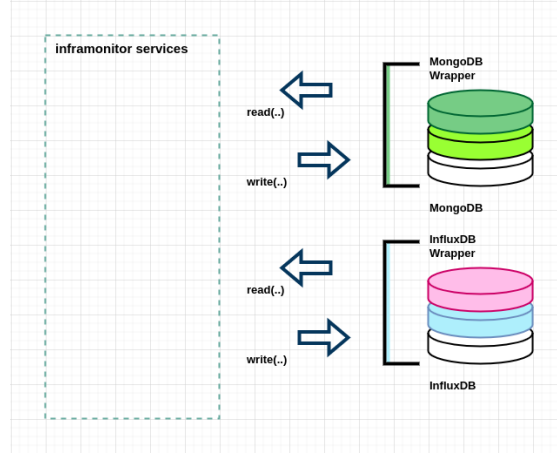


Figure 6.2: The database wrappers.

To connect to the databases, we use clients written in python, wrapping the CRUD operations such that the database can easily be exchanged for another one, without having to change the way services connect to them. MongoDB is schemaless, hence it neither requires nor enforces a predefined data structure. Instead, the MongoDB wrapper for the ontology database encodes the database schema.

We use the NGSI-LD context information management information model [ETS21]. Consequently, we require each entry to have an NGSI-LD id, consisting of a prefix explicitly declaring the information model, the entry type, and an alphanumeric identifier for the entry. An example of an energy meter with the device id 0001A would look as follows:

```
1 urn:ngsi-ld:energy:0001A
```

As per the NGSI-LD data model, we require each entry to explicitly specify its type in a separate field, and that the type in this field matches the type in the id. The MongoDB wrapper ensures that these fields are present when an entry is created and that these fields are not changed when an entry is updated.

Additionally, the wrapper adds a timestamp to when an entry was created or last updated if the field is not specified in the payload of the entry. Any additional formatting of the data entries is delegated to the service invoking the MongoDB wrapper. The simplest, valid, newly created entry for an energy sensor with the device id 0001A is as follows:

```
1 {
2   "_id": "urn:ngsi-ld:energy:0001A",
3   "type": "energy",
4   "dateCreated": {
5     "type": "DateTime",
6     "value": "2022-09-22 12:00:00"
7   }
8 }
```

The InfluxDB wrapper handles CRUD operations for the time series database. Data points in the InfluxDB are identified by their timestamp, *field* key and value, and a set of *tag* keys and values, and grouped into *measurements* [Inf20]. We denote all time series *sensor-readings* because we do not need additional separation between groups of device metrics. We currently use three types of fields: reading, prediction, and status. However, the wrapper imposes no restrictions on the field keys or values. We use the tag keys and values to assign readings, statuses, and predictions with the sensor they are associated with, but it is possible to use additional tags for filtering with the InfluxDB wrapper.

6.3.4 Database Updates

Typically, the MQTTAgent writes decoded sensor updates to the ontology database. To obtain sensor value trajectories of equidistant entries, a service denoted MongoAgent, transfers the values from the ontology database to the time series database at a specified interval. For energy and water meters, we update the time series database every 15 minutes. At this interval, we retain sufficient information about the overall trends without overfitting on short-term peaks, e.g., spikes in water consumption when flushing toilets. For the other sensor devices, capturing metrics, such as CO₂, temperature, humidity, window and door contacts, or operation statuses, new sensor values are stored in the ontology database and the time series database on every update. Figure 6.3 shows a graphical representation of the setup.

The requested update policy for each sensor is specified through its type, i.e., sensor types are assigned an update policy. The sensor types are specified in the ontology database. If the type was not set manually, e.g., when provisioning the sensor, the MQTT agent tries to infer the sensor type from known patterns in the sensor name. Note that the patterns we use are specific to our use case, but it stands to reason that in any other system there are similar patterns are likely. In case the sensor was not provisioned before the MQTTAgent receives the first update from it and the type cannot be inferred from the sensor name, the type defaults to *UnknownSensor*. The default policy for sensors of unknown type is to update the time series database on every change.

6.3.5 Data Management and Data Access

To access and administrate the databases we provide two endpoints: one for maintainers and operators, and one for external project partners. Both interfaces are tailored toward our use cases. The admin interface is only available via SSH and allows maintainers to create, update, delete, and export entities in the ontology database grouped as sites (i.e., the campuses), buildings, and sensors, and to change, add and delete API keys for the MQTT broker.

External partners can retrieve data from the ontology database and the time series database through an HTTP REST API. The container providing the API runs a Flask

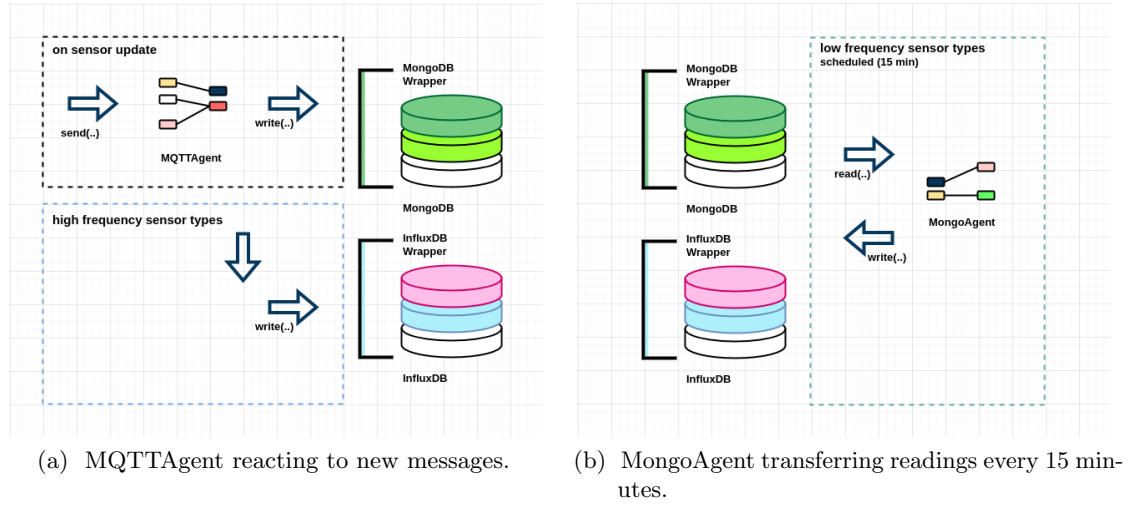


Figure 6.3: The MQTT data collection pipeline. In 6.3a you can see the MQTTAgent, subscribed on the broker, reacting to new messages. For high-frequency sensors, i.e., the ones where we want to retain each update in the time series database the MQTT agent forwards them to the InfluxDB directly. In 6.3b you see the MongoAgent transferring the sensor status to the InfluxDB every 15 minutes.

micro web server that converts between HTTP requests and a Web Server Gateway Interface (WSGI) environment and outgoing WSGI responses to the appropriate HTTP responses [Pal22]. The API allows external partners to easily access information from both databases in the same manner, without the need for them to know about the architectural separation.

6.3.6 Data Analysis

On top of the data collection and data persistence, we developed automatic monitoring and maintenance services. These include automated supervision services for water consumption, energy demand, and PV production. We applied a similar approach to the one described in [SSP+20], i.e., to predict the expected nominal consumption and production values using regression models and then compare them to the actual values according to heuristics specified in coordination with the building operators. The monitoring process hence consists of two stages, and to decouple these stages we implemented it in two services: a prediction service and a monitoring service. This separation has three main advantages: (i) the nominal value can be predicted at any time, i.e., the actual value does not need to be available at the time of prediction, allowing us to predict in advance; (ii) the predictions can be used for other services such as visualization

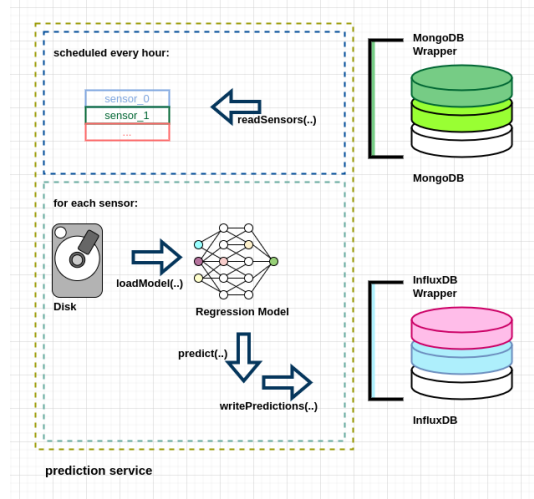


Figure 6.4: The prediction service. The service obtains a list of all sensors from the ontology database, scans for a model on the disk, loads the model, predicts the expected, nominal sensor value, and stores it in the database.

or optimal control strategies, and (iii) it is straightforward to swap the regression models or the monitoring heuristics independently of each other.

We developed, trained, and validated the regression models for the *prediction* service within a Python framework based on the code and the conceptual ideas from [SEL+21]. The framework is available on GitHub at <https://github.com/tug-cps/datamodels>.

We export sensor data from the inframonitor to train the models on a different machine than the one running the inframonitor services. This way we are (i) more flexible in developing and tweaking the models, e.g., by training them on a machine with more computational power or by having operators train them on their machines, and (ii) we do not interfere with the operation of the inframonitor system. The model architectures and parameters for a trained and validated model are stored in a directory.

Note that the required input data depends on the model characteristics. Consider, for example, a model predicting the water consumption for the current hour based on the weekday, current time of day, and occupancy and another model predicting the energy demand for the next hour based on the energy demand in the last ten hours. To obtain and preprocess the input data for the model as generically as possible, we have the models bootstrap the process, i.e., we add a function that handles data collection and preprocessing to the model directory. This way, the information on what input data the model requires is contained within the model, and the prediction service does not have to know about the specifics of a model to invoke it. To deploy a model to the inframonitor, we push the model directory to the server.

Currently, we use a one-hour resolution for the estimated water consumption, energy

6 End-to-End Integration

demand, and PV production values. Hence, we schedule the prediction service to run every full hour. The service uses the MongoDB wrapper to obtain a list of all sensors in the ontology database and then scans a location on the server for corresponding model directories. If a model directory exists, the prediction service loads the model using the Python framework, uses the model-specific function from the directory to obtain and preprocess the input, invokes the model, and writes the results to the time series databases using the InfluxDB wrapper. A simplified, schematic representation of the workflow can be seen in 6.4.

To interpret the relationship between the predicted and actual values, we use a service denoted *monitor*. The monitor infers a status based on a set of strategies we developed in coordination with the building operators. The strategies for the water consumption sensors, for instance, check for four potential fault cases: (i) a new, measured value exceeds the highest prediction by a factor of 1.3, (ii) three consecutive sensor updates exceed the corresponding predicted values by a factor of 1.3, (iii) the consumption value never reaches the lowest predicted values during the night, when we expect the buildings to be empty, and iv) the sensor values do not change for more than 24 hours. To obtain the statuses, the monitor pulls the past 24 hours of predicted and actual values from the time series database every hour, applies the strategies, and encodes the results into a status code. The status code is a bitmask with the faults represented by a sequence of numbers with a separate bit for each case. This representation is compact and retains all information within a single value. The status codes for each pair of predicted and actual values are stored in the time series database.

From the status codes, we generate status reports and send out alerts to building operators with a separate *alert* service. The alert service, running every hour to match the resolution of the prediction and monitoring services, reads the status codes from the past two hours and generates a human-readable status report, separating faults into new ones, persisting ones, and corrected ones by comparing the bitmasks between the current and the previous status codes. The service sends reports to subscribed operators via mail.

6.3.7 Data Visualization

For operators and building occupants, we provide data visualization in the format of historic time series trajectories, real-time status monitors, and predicted sensor readings. The *visualization* service is implemented as dockerized *Grafana* web application. Grafana is a powerful open-source data visualization and data analytics service that allows for extensive customization. It connects natively to the InfluxDB for time series data and accesses data from the ontology database through a third-party plugin. An example dashboard for a mixed-use academic office building is shown in 6.5.



Figure 6.5: A Grafana dashboard. This showcase example visualizes predicted and actual water consumption, predicted occupancy as well as predicted hourly PV production and actual daily PV production.

6.4 Discussion

Efficient, open, scalable IoT infrastructure has become a key interest for building operators. While there are commercial providers for IoT middleware solutions, it can be difficult for practitioners to choose a suitable platform for their needs, especially, when they require very specific services and functionalities. Within the European Union, the non-commercial, open FIWARE platform has received considerable support from EU institutions, academia, and industrial partners. Based on the scientific literature, it is reasonable to assume that there are many concepts and ideas behind the FIWARE platform that address the main challenges in the development and operation of IoT middleware platforms.

The principal design decisions, such as the use of the NGSI-LD data model, the containerized separation of components as well as the use of two separate databases for the ontology and the time series data, respectively have proven to result in flexible, modular architectures that can be connected with a wide range of devices. However, based on our experience and the experience of many of our project partners, there are issues with the implementation and interoperability in the curated set of components provided by the FIWARE initiative, such as problems with connecting the ontology database wrapper (the *Orion context broker*) to the update service for the time series database, data inconsistencies through the use of the IoT agent, or difficulties with implementing access control.

Additionally, we found that the configuration is somewhat cumbersome for several components, with the configuration files alone often being significantly larger and more complex than many of the implementations we built ourselves. Besides, it is worth noting that the Orion broker, does not validate the data that is sent to the database. Hence

6 *End-to-End Integration*

it is necessary to perform inspections, such as checking if an entity id has the correct format manually, which requires additional wrapping of a component that already acts as a wrapper for the database.

We understand that many of these issues might be addressed by manually debugging, testing, and adapting the code. However, we like to point out that this solution is time-intensive and requires the expertise of experienced software developers, and that spending this time and knowledge to implement a solution that is more tailored explicitly toward the actual application might be preferable, especially when it is possible to achieve the same level of openness and interoperability by using the same open protocols and data models that FIWARE uses.

7 Conclusion and Future Extensions

Simulation is a principal interest for many domains concerned with cyber-physical systems, such as building energy systems, HVAC, or smart grids, industrial processes, autonomous automobile systems, or robotics. Through the increasing availability of operational data, the development of user-friendly, open-source frameworks, and the increase in computational power, data-driven simulation methods have gained popularity. In the remainder of this section we revisit the research objectives formulated in Section 2.1 and reflect on how the work in this thesis addresses them.

RO₁ *Apply well-established machine learning methods to predict the behavior of dynamical systems in buildings.*

RO₂ *Compare prediction and forecasting approaches.*

In Chapter 3, we discussed the application of machine learning methods for modeling energy and water consumption patterns in buildings. Results show that both time-series-based forecasting and feature-based prediction methods are promising approaches even though their means to encode the inherent temporal structure of the data are fundamentally different.

For the prediction approach, the temporal relationship between system states was encoded into the features we engineered from the timestamps. Hence, this is where we anticipate the highest potential for improvement, e.g., through additional research into extending the set of temporal features with variables that capture monthly or yearly seasonalities and potentially permanent shifts and trends in the trajectories. We theorize that it is possible to extract this meta-information through additional statistical data analysis.

With forecasting models, we achieved relatively accurate results even when we used model architectures that are unaware of the temporal relationship between the input features, such as linear models, random forests, and fully-connected neural networks. The fully-connected and the recurrent neural network performed well at estimating the future system state from previous states only if the lookback is sufficiently long, i.e., around 24 hours. However, while preliminary results are promising, we acknowledge that additional processing, such as normalization or network regularization, as well as other network architectures, should be examined. Furthermore, we propose to have the forecasting models estimate multiple future states of the system and evaluate their accuracy for each time step separately, such that the same model can be used to obtain different future states.

7 Conclusion and Future Extensions

The time series we approximated in Chapter 3 are governed by human behavior. Hence, the regression problems that the models solved were inherently stochastic. However, when simulating trajectories of system states for more complex dynamical processes, traditional, purely data-driven models may fail, especially when training data is scarce. Consider, for instance, the direct-solution neural network model for the ideal pendulum we discussed in Section 4.7.2: While the model fits the trajectory well in the collocation (training) points, it fails to generalize.

With access to prior information about a dynamical system, it is possible to develop simulation models that combine theory-based and data-based approaches to leverage their respective advantages. The paradigms of informed machine learning, theory-guided data science, and physics-informed neural networks have spawned many novel approaches toward data-driven simulation.

RO₃ *Survey theory-informed neural-network architectures for the simulation of dynamical systems and apply them to a case study example.*

RO₄ *Identify common mechanisms in theory-informed neural-network architectures and derive a taxonomy.*

In Chapter 4, we presented the reader with a comprehensive analysis of the state of the art in dynamical system simulation using neural networks. The literature covers a wide variety of applications, and proposes a myriad of models that can be used to solve ODEs or PDEs, simulate graph-like systems, incorporate external stimuli, quantify uncertainty, or use well-known numerical solvers within their architectures.

From our analysis, we conclude that many of these models apply similar approaches, and/or can be combined to obtain new model architectures. Consider, for example, the application of automatic differentiation, described in Section 4.7.3 and the architecture of the physics-informed neural network in Section 4.7.4: both models obtain one system state by deriving the other. Many model architectures successfully address the inherent shortcomings of purely data-driven models by improving generalization performance, providing mechanisms to ensure plausible and reliable behavior, or allowing practitioners to quantify uncertainties explicitly. However, it is still difficult to properly assess the potential of each architecture without a benchmark dataset containing a collection of dynamical systems that is representative of a variety of application domains.

While theory-guided and physics-informed models address some of the intrinsic issues of data-driven simulation, the need to simulate systems of ever-increasing complexity requires increasingly complex models. Additionally, complex systems often comprise heterogeneous subsystems, and to simulate them it is often preferable or even necessary to model each subsystem with a separate, specific model and combine the submodels in a co-simulation. While the scientific literature proposes various coupling and synchronization techniques for simulation units, standard interface coupling using the functional mockup interface (FMI) has become one of the most popular methods.

RO₅ *Develop a framework that generates co-simulation-ready software components from arbitrary models.*

In Chapter 5, we presented the UniFMU tool, a framework that allows the generation of portable, cross-platform, FMI-compliant simulator units from code in any programming language. With access to scripting languages such as Python, practitioners can utilize an entire ecosystem of machine learning frameworks such as TensorFlow or Pytorch and other scientific libraries that simplify the development of specialized simulation models. By decoupling the FMI API implementation from the actual simulation model, practitioners can test and debug the model’s code using standard development and debugging tools. By virtualizing models in Docker containers, the simulation units can be run on any host machine that provides a Docker service, without additional runtime requirements.

While we have not had any performance issues, it is still necessary to assess the communication overhead the tool introduces. Additional overhead might be less of an issue when using a comparatively slow language such as Python MATLAB or Java, which is why we have not encountered problems during our tests. However, in certain simulations, speed might be a limiting factor hence it should be considered. Related work could include a means to automatically synchronize the FMI-compatible model description with the actual implementation, which is a functionality we deliberately omitted from UniFMU’s core functionality because to automatically generate and update the description the tool would have to validate the implementation, which can be not trivial and introduce massive overhead.

Typically, the overarching objective of simulating cyber-physical systems is to integrate the simulation models with the system. Thus, it is necessary to consider interactions between sensors, data persistence, data analysis, and higher-level services, such as operation, supervision, and maintenance in a cyber-physical system. The end-to-end data flow in a complex system, such as the university campus we described in Section 3.4.1 and Section 6.2.3, consists of a variety of subsystems, such as HVAC, water supply, and the energy system, and passes through several heterogeneous computational components. As such, buildings can be considered an excellent use case for data-driven models to estimate human behavior, physics-informed or theory-guided simulation to model complex physical interactions, and co-simulation to obtain a high-level model of the entire system.

RO₆ *Develop a scalable, modular software platform that facilitates data collection and simulator integration.*

In Chapter 6, we described an internet of things (IoT) middleware platform that facilitates communication between computational components, handles data collection and persistence, and integrates high-level data analysis and visualization services. The main motivation behind the development of this platform was to have an efficient, scalable, and modular framework that allows us to collect operational data necessary to implement data-driven simulation methods and to feed back the simulation results into the system.

In recent years, research into IoT middleware has gained popularity, and there are many commercial and non-commercial middleware solutions. To assess the state of the art, and to identify the requirements of practitioners and operators, we conducted an expert survey. The platform we described in Section 6.3 is based on the results from this survey, an extensive literature review, and inputs from the building operators at the Graz University of Technology (TUG). Besides growing and extending the features and services for the TUG, we have since adapted the components we developed to other IoT use cases as well.

We consider our software stack a modular set of open components that will remain under continuous development per design. However, the main open issues are (i) automatic testing and validation of the components, (ii) security and privacy in applications where third-parties are involved, and (iii) integration testing based on specific real-world use cases. Our literary research, the expert survey and the case study show that the development and validation of open ontologies, data models and data representation require additional research and a closer collaboration between practitioners and academia.

7.1 Open Questions

In this final section of the thesis we briefly outline the most relevant open research questions that we derive from the conclusions in the previous section.

- With building processes formulated as dynamical regression problems, we can use a wide variety of machine learning methods. Results show that even relatively simple approaches can be used to automate building control, if the features are chosen correctly. This intersection between expert-guided data selection and the application of generic, well-established machine learning models warrants additional research into more advanced knowledge integration, preprocessing and feature selection.
- While many novel, advanced neural network architectures from the literature successfully address the inherent shortcomings of purely data-driven models by improving generalization performance, providing mechanisms to ensure plausible and reliable behavior, or allowing practitioners to quantify uncertainties explicitly,

it is necessary to develop representative benchmark datasets to properly assess the architecture's potential.

- Co-simulation is a promising paradigm that allows practitioners to apply the most suitable simulation framework for each component in a system. Software frameworks such as *UniFMU* introduce a plethora of new languages and libraries into the co-simulation ecosystem and provide practitioners with an accessible tool to develop highly specialized simulation units. One important objective, however, is to assess and potentially improve the runtime overhead introduced by these tools and to provide additional automation and validation within the framework's environment.
- Many of the issues in IoT systems are solved on a technical level. However, the development of data representations, ontologies, data models, and meta-data information frameworks remains largely unsolved.

Bibliography

- [ACB21] Adoubi Vincent De Paul Adombi, Romain Chesnaux, and Marie-Amélie Boucher. “Review: Theory-guided Machine Learning Applied to Hydrogeology—State of the Art, Opportunities and Future Challenges”. In: *Hydrogeology Journal* 29.8 (Dec. 2021), pp. 2671–2683. ISSN: 1431-2174, 1435-0157. DOI: 10.1007/s10040-021-02403-2.
- [AE18] Kadir Amasyali and Nora M. El-Gohary. “A Review of Data-Driven Building Energy Consumption Prediction Studies”. In: *Renewable and Sustainable Energy Reviews* 81 (Jan. 2018), pp. 1192–1205. ISSN: 13640321. DOI: 10.1016/j.rser.2017.04.095.
- [AEK+15] A. Allouhi, Y. El Fouih, T. Kousksou, A. Jamil, Y. Zeraouli, and Y. Mourad. “Energy Consumption and Efficiency in Buildings: Current Status and Future Trends”. In: *Journal of Cleaner Production* 109 (Dec. 2015), pp. 118–130. ISSN: 09596526. DOI: 10.1016/j.jclepro.2015.05.139.
- [AGM+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.
- [AHF+19] Milad Ashouri, Fariborz Haghighat, Benjamin C.M. Fung, and Hiroshi Yoshino. “Development of a Ranking Procedure for Energy Performance Evaluation of Buildings Based on Occupant Behavior”. In: *Energy and Buildings* 183 (Jan. 2019), pp. 659–671. ISSN: 03787788. DOI: 10.1016/j.enbuild.2018.11.050.
- [ÅM10] Karl Johan Åström and Richard M Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. In English. 2010. ISBN: 978-1-4008-2873-9.
- [AMS+19] Victor Araujo, Karan Mitra, Saguna Saguna, and Christer Åhlund. “Performance Evaluation of FIWARE: A Cloud-Based IoT Platform for Smart Cities”. In: *Journal of Parallel and Distributed Computing* 132 (Oct. 2019), pp. 250–261. ISSN: 07437315. DOI: 10.1016/j.jpdc.2018.12.010.
- [APP+90] David K Arrowsmith, Colin M Place, CH Place, et al. *An introduction to dynamical systems*. Cambridge university press, 1990.
- [AR18] Maren Awiszus and Bodo Rosenhahn. “Markov Chain Neural Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2180–2187.

Bibliography

- [AS22] Kari Alanne and Seppo Sierla. “An Overview of Machine Learning Applications for Smart Buildings”. In: *Sustainable Cities and Society* 76 (Jan. 2022), p. 103445. ISSN: 22106707. DOI: 10.1016/j.scs.2021.103445.
- [ASF+23] Qamar Alfalouji, Thomas Schranz, Basak Falay, Sandra Wilfling, Johannes Exenberger, Thorsten Mattausch, Claudio Gomes, and Gerald Schweiger. “Co-Simulation for Buildings and Smart Energy Systems — A Taxonomic Review”. In: *Simulation Modelling Practice and Theory* (Apr. 2023), p. 102770. ISSN: 1569190X. DOI: 10.1016/j.simpat.2023.102770. (Visited on 04/25/2023).
- [ASH02] ASHRAE Standards Committee. “ASHRAE Guideline: Measurement of Energy and Demand Savings”. en. In: *ASHRAE Guideline 14-2002* (2002).
- [ASK+22] Qamar Alfalouji, Thomas Schranz, Alexander Kümpel, Markus Schraven, Thomas Storek, Stephan Gross, Antonello Monti, Dirk Müller, and Gerald Schweiger. “IoT Middleware Platforms for Smart Energy Systems: An Empirical Expert Survey”. In: *Buildings* 12.5 (Apr. 2022), p. 526. ISSN: 2075-5309. DOI: 10.3390/buildings12050526.
- [AT10] Syed Adeel Asghar and Sonia Tariq. “Design and Implementation of a User Friendly OpenModelica Graphical Connection Editor”. In: (2010), p. 81.
- [Bac00] Alexander Backlund. “The definition of system”. In: *Kybernetes* (2000).
- [BBC+21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges”. In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478.
- [BCJ+19] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. “Pyro: Deep Universal Probabilistic Programming”. In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 973–978.
- [BDC+18] Keith T. Butler, Daniel W. Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. “Machine Learning for Molecular and Materials Science”. In: *Nature* 559.7715 (July 2018), pp. 547–555. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-018-0337-2.
- [Beh15] Jörg Behler. “Constructing High-Dimensional Neural Network Potentials: A Tutorial Review”. In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1032–1050. DOI: 10.1002/qua.24890. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/qua.24890>.
- [BGC+19] Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. “Invertible Residual Networks”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 573–582.

- [BHB+18] Peter W. Battaglia et al. “Relational Inductive Biases, Deep Learning, and Graph Networks”. In: *CoRR* abs/1806.01261 (2018). arXiv: 1806.01261.
- [Bir27] George David Birkhoff. *Dynamical systems*. Vol. 9. American Mathematical Soc., 1927.
- [Bis94] Chris M. Bishop. “Neural Networks and Their Applications”. In: *Review of Scientific Instruments* 65.6 (June 1994), pp. 1803–1832. ISSN: 0034-6748, 1089-7623. DOI: 10.1063/1.1144830.
- [BK20] Sharu Bansal and Dilip Kumar. “IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication”. In: *International Journal of Wireless Information Networks* 27.3 (Sept. 2020), pp. 340–364. ISSN: 1068-9605, 1572-8129. DOI: 10.1007/s10776-020-00483-7.
- [BK22] Steven L. Brunton and Jose Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge, United Kingdom, New York, NY: Cambridge University Press, 2022. ISBN: 978-1-00-909848-9.
- [BLT+12] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. “Viewpoints, Formalisms, Languages, and Tools for Cyber-Physical Systems”. In: *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*. Innsbruck Austria: ACM, Oct. 2012, pp. 49–54. ISBN: 978-1-4503-1805-1. DOI: 10.1145/2508443.2508452. (Visited on 05/02/2023).
- [BNK20] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. “Machine Learning for Fluid Mechanics”. In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214.
- [BOA+11] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. “The Functional Mockup Interface for Tool Independent Exchange of Simulation Models”. In: *The 8th International Modelica Conference, Technical University, Dresden, Germany*. June 2011, pp. 105–114. DOI: 10.3384/ecp11063105.
- [Bon17] Giuseppe Bonaccorso. *Machine learning algorithms*. Packt Publishing Ltd, 2017.
- [Bot19] Alexei Botchkarev. “A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms”. In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14 (2019), pp. 045–076. ISSN: 1555-1229, 1555-1237. DOI: 10.28945/4184.
- [BPL+16] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. “Interaction Networks for Learning about Objects, Relations and Physics”. In: *CoRR* abs/1612.00222 (2016). arXiv: 1612.00222.

Bibliography

- [BS02] Michael Brin and Garrett Stuck. *Introduction to Dynamical Systems*. Cambridge, U.K ; New York: Cambridge University Press, 2002. ISBN: 978-0-521-80841-5.
- [BSQ+21] 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”. In: *14th Modelica Conference 2021*. Sept. 2021, pp. 335–341. DOI: 10.3384/ecp21181335.
- [BV13] Controllab Products B.V. <http://www.20sim.com/>. 20-sim official website. Jan. 2013.
- [CCC+19] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. “Machine Learning and the Physical Sciences”. In: *Reviews of Modern Physics* 91.4 (Dec. 2019), p. 045002. ISSN: 0034-6861, 1539-0756. DOI: 10.1103/RevModPhys.91.045002.
- [CD14] T. Chai and R. R. Draxler. “Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)? – Arguments against Avoiding RMSE in the Literature”. In: *Geoscientific Model Development* 7.3 (June 2014), pp. 1247–1250. ISSN: 1991-9603. DOI: 10.5194/gmd-7-1247-2014.
- [CDL16] Xiaodong Cao, Xilei Dai, and Junjie Liu. “Building Energy-Consumption Status Worldwide and the State-of-the-Art Technologies for Zero-Energy Buildings during the Past Decade”. In: *Energy and Buildings* 128 (Sept. 2016), pp. 198–213. ISSN: 03787788. DOI: 10.1016/j.enbuild.2016.06.089.
- [CE19] Debaditya Chakraborty and Hazem Elzarka. “Advanced Machine Learning Techniques for Building Performance Simulation: A Comparative Analysis”. In: *Journal of Building Performance Simulation* 12.2 (Mar. 2019), pp. 193–207. ISSN: 1940-1493, 1940-1507. DOI: 10.1080/19401493.2018.1498538.
- [CGH+20] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. “Lagrangian Neural Networks”. In: (July 2020). arXiv: 2003.04630 [physics, stat].
- [CHB+18] Travers Ching et al. “Opportunities and Obstacles for Deep Learning in Biology and Medicine”. In: *Journal of The Royal Society Interface* 15.141 (Apr. 2018), p. 20170387. DOI: 10.1098/rsif.2017.0387.
- [Cho+15] François Chollet et al. *Keras*. <https://keras.io>. Accessed 2022.09.06. 2015.
- [Chw03] Dorota Chwieduk. “Towards sustainable-energy buildings”. en. In: *Applied Energy* 76.1-3 (Sept. 2003), pp. 211–217. ISSN: 03062619. DOI: 10.1016/S0306-2619(03)00059-X. URL: <https://linkinghub.elsevier.com/retrieve/pii/S030626190300059X> (visited on 01/28/2021).

- [CKD+15] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. “A Recurrent Latent Variable Model for Sequential Data”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015.
- [CLB+19] Fabio Cremona, Marten Lohstroh, David Broman, Edward A. Lee, Michael Masin, and Stavros Tripakis. “Hybrid Co-Simulation: It’s about Time”. en. In: *Software & Systems Modeling* 18.3 (June 2019), pp. 1655–1679. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-017-0633-6.
- [CMH+18] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. “Multi-Level Residual Networks from Dynamical Systems View”. In: (Feb. 2018). arXiv: 1710.10348 [cs, stat].
- [Con21] Context Information Management (CIM) ETSI Industry Specification Group (ISG). *Context Information Management (CIM); NGSI-LD API*. https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01_04_01_60/gs_cim009v010401p.pdf. Accessed 2022.10.04. 2021.
- [CPL+18] Zhengping Che, Sanjay Purushotham, Guangyu Li, Bo Jiang, and Yan Liu. “Hierarchical Deep Generative Models for Multi-Rate Multivariate Time Series”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, July 2018, pp. 784–793.
- [CRB+19] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. “Neural Ordinary Differential Equations”. In: (Dec. 2019). arXiv: 1806.07366 [cs, stat].
- [CSB+19] Flavio Cirillo, Gurkan Solmaz, Everton Luis Berz, Martin Bauer, Bin Cheng, and Erno Kovacs. “A Standard-Based Open Source IoT Platform: FIWARE”. In: *IEEE Internet of Things Magazine* 2.3 (Sept. 2019), pp. 12–18. DOI: 10.1109/iotm.0001.1800022. URL: <https://doi.org/10.1109/iotm.0001.1800022>.
- [CTG17] Chao Chen, Jamie Twycross, and Jonathan M. Garibaldi. “A New Accuracy Measure Based on Bounded Relative Error for Time Series Forecasting”. In: *PLOS ONE* 12.3 (Mar. 2017). Ed. by Zhong-Ke Gao, e0174202. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0174202.
- [CUT+16] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. “A Compositional Object-Based Approach to Learning Physical Dynamics”. In: *CoRR* abs/1612.00341 (2016). arXiv: 1612.00341.

Bibliography

- [CWJ21] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. “The Coefficient of Determination R-squared Is More Informative than SMAPE, MAE, MAPE, MSE and RMSE in Regression Analysis Evaluation”. In: *PeerJ Computer Science* 7 (July 2021), e623. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.623.
- [DCD+21] Raj Dandekar, Karen Chung, Vaibhav Dixit, Mohamed Tarek, Aslan Garcia-Valadez, Krishna Vishal Vemula, and Chris Rackauckas. “Bayesian Neural Ordinary Differential Equations”. In: (Mar. 2021). arXiv: 2012.07244 [cs].
- [DDT19] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. *Augmented Neural ODEs*. 2019. arXiv: 1904.01681 [stat.ML].
- [dGL+16] Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. “Mean Absolute Percentage Error for Regression Models”. In: *Neurocomputing* 192 (June 2016), pp. 38–48. ISSN: 09252312. DOI: 10.1016/j.neucom.2015.12.114.
- [DHR15] Laura Daniele, Frank den Hartog, and Jasper Roes. “Created in Close Interaction with the Industry: The Smart Appliances REference (SAREF) Ontology”. en. In: *Formal Ontologies Meet Industry*. Ed. by Roberta Cuel and Robert Young. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2015, pp. 100–112. ISBN: 9783319215457. DOI: 10.1007/978-3-319-21545-7_9.
- [DLT+17] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. “TensorFlow Distributions”. In: *CoRR* abs/1711.10604 (2017). arXiv: 1711.10604.
- [Doc20] Simulink Documentation. *Simulation and Model-Based Design*. 2020. URL: <https://www.mathworks.com/products/simulink.html>.
- [DR98] I. Drezga and S. Rahman. “Input variable selection for ann-based short-term load forecasting”. In: *IEEE Transactions on Power Systems* 13.4 (1998), pp. 1238–1244. ISSN: 08858950. DOI: 10.1109/59.736244.
- [dRA+18] Mauro A. A. da Cruz, Joel Jose P. C. Rodrigues, Jalal Al-Muhtadi, Valery V. Korotaev, and Victor Hugo C. de Albuquerque. “A Reference Model for Internet of Things Middleware”. In: *IEEE Internet of Things Journal* 5.2 (Apr. 2018), pp. 871–883. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2796561. (Visited on 05/02/2023).
- [DTC+20] Jan Drgona, Aaron R. Tuor, Vikas Chandan, and Draguna L. Vrabie. *Physics-Constrained Deep Learning of Multi-Zone Building Thermal Dynamics*. 2020. arXiv: 2011.05987 [cs.LG].

- [EC19] European Commission and Climate Action DG. *Going climate-neutral by 2050: a strategic long-term vision for a prosperous, modern, competitive and climate-neutral EU economy*. English. OCLC: 1140133232. 2019. ISBN: 978-92-76-02079-0.
- [EH14] Jose Evora-Gomez and Jose Juan Hernández. “JavaFMI: Interoperability for Integrating Simulations”. In: (2014). DOI: 10.13140/RG.2.2.12431.48805.
- [ETS21] ETSI. *ETSI Context Information Management; NGSI-LD*. https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.01_60/gs_cim009v010401p.pdf. Accessed 2022.12.06. 2021.
- [Eur20] European Environmental Agency. *Annual European Union greenhouse gas inventory 1990–2018 and inventory report 2020: Submission under the United Nations Framework Convention on Climate Change and the Kyoto Protocol*. Tech. rep. European Commission, DG Climate Action European Environment Agency, 2020, p. 997. URL: <https://www.eea.europa.eu/publications/european-union-greenhouse-gas-inventory-2020>.
- [FGL+15] John Fitzgerald, Carl Gamble, Peter Gorm Larsen, Kenneth Pierce, and Jim Woodcock. “Cyber-Physical Systems Design: Formal Foundations, Methods and Integrated Tool Chains”. In: *2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering*. Florence, Italy: IEEE, May 2015, pp. 40–46. ISBN: 978-1-4673-7043-1. DOI: 10.1109/FormaliSE.2015.14. (Visited on 05/02/2023).
- [FGM+21] Sawyer Fuller, Ben Greiner, Jason Moore, Richard Murray, René van Paassen, and Rory Yorke. “The Python Control Systems Library (python-control)”. In: *60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 4875–4881.
- [FIW20] FIWARE Foundation. *FIWARE NGSIv2*. <http://fiware.github.io/specifications/ngsiv2/stable>. Accessed 2022.10.06. 2020.
- [FIW21] FIWARE Foundation. *The FIWARE Foundation*. <https://www.fiware.org/foundation>. Accessed 2022.10.04. 2021.
- [FIW22] FIWARE Foundation. *The Ultralight 2.0 Protocol*. <https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual.html>. Accessed 2022.10.04. 2022.
- [FJN+20] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. *How to Train Your Neural ODE: The World of Jacobian and Kinetic Regularization*. 2020. arXiv: 2002.02798 [stat.ML].
- [FP20] Marco Forgione and Dario Piga. *dynoNet: A Neural Network Architecture for Learning Dynamical Systems*. 2020. arXiv: 2006.02250 [cs.LG].

Bibliography

- [FPE20] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. eng. Eighth edition, global edition. Harlow, United Kingdom: Pearson Education Limited, 2020. ISBN: 978-1-292-27452-2.
- [FPL14] John Fitzgerald, Ken Pierce, and Peter Gorm Larsen. “Co-Modelling and Co-Simulation in the Engineering of Systems of Cyber-Physical Systems”. In: *2014 9th International Conference on System of Systems Engineering (SOSE)*. Adelaide, SA: IEEE, June 2014, pp. 67–72. ISBN: 978-1-4799-5227-4. DOI: 10.1109/SYSOSE.2014.6892465. (Visited on 04/24/2023).
- [FSF+20] Soheil Fathi, Ravi Srinivasan, Andriel Fenner, and Sahand Fathi. “Machine Learning Applications in Urban Building Energy Performance Forecasting: A Systematic Review”. In: *Renewable and Sustainable Energy Reviews* 133 (Nov. 2020), p. 110287. ISSN: 13640321. DOI: 10.1016/j.rser.2020.110287.
- [FSP+16] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. “Sequential Neural Models with Stochastic Layers”. In: *arXiv preprint arXiv:1605.07571* (2016). arXiv: 1605.07571.
- [FZ20] Mahdi Fahmideh and Didar Zowghi. “An exploration of IoT platform development”. In: *Information Systems* 87 (Jan. 2020), p. 101409. DOI: 10.1016/j.is.2019.06.005. URL: <https://doi.org/10.1016/j.is.2019.06.005>.
- [GB10] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.
- [GBC+16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*. Vol. 1. MIT press Cambridge, 2016.
- [GDY19] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. “Hamiltonian Neural Networks”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 15379–15389.
- [GHL+21] Anubhab Ghosh, Antoine Honoré, Dong Liu, Gustav Eje Henter, and Saikat Chatterjee. “Robust Classification Using Hidden Markov Models and Mixtures of Normalizing Flows”. In: *CoRR* abs/2102.07284 (2021). arXiv: 2102.07284.
- [GK96] P.D. Gader and M.A. Khabou. “Automatic Feature Generation for Hand-written Digit Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.12 (Dec. 1996), pp. 1256–1261. ISSN: 01628828. DOI: 10.1109/34.546262.

- [GLP19] Batuhan Güler, Alexis Laignelet, and Panos Parpas. *Towards Robust and Stable Deep Learning Algorithms for Forward Backward Stochastic Differential Equations*. 2019. arXiv: 1910.11623 [stat.ML].
- [GRI+20] Xiang Gao, Farhad Ramezanghorbani, Olexandr Isayev, Justin S. Smith, and Adrian E. Roitberg. “TorchANI: A Free and Open Source PyTorch-Based Deep Learning Implementation of the ANI Neural Network Potentials”. In: *Journal of Chemical Information and Modeling* 60.7 (2020), pp. 3408–3415. DOI: 10.1021/acs.jcim.0c00451. eprint: <https://doi.org/10.1021/acs.jcim.0c00451>.
- [gRP] gRPC Authors. *gRPC: A high-performance, open source universal RPC framework*. <https://grpc.io>. Accessed 2022.11.06. (Visited on 11/06/2022).
- [GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. “Neural Message Passing for Quantum Chemistry”. In: *CoRR* abs/1704.01212 (2017). arXiv: 1704.01212.
- [GTB+17] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. *Co-simulation: State of the art*. Tech. rep. Feb. 2017. URL: <http://arxiv.org/abs/1702.00686>.
- [GTB+19] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. “Co-Simulation: A Survey”. In: *ACM Computing Surveys* 51.3 (May 2019), pp. 1–33. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3179993.
- [GWS+20] Daniel Gedon, Niklas Wahlström, Thomas B. Schön, and Lennart Ljung. *Deep State Space Models for Nonlinear System Identification*. 2020. arXiv: 2003.14162 [eess.SY].
- [HDS+20] Maliheh Haghighi, Alberto Dognini, Thomas Storek, Radu Plamanescu, Ulrike Rahe, Stefan Gheorghe, Mihaela Albu, Antonello Monti, and Muller Dirk. “Open Smart Energy Eco-System for the Future”. In: *IOP Conference Series: Earth and Environmental Science* 588.2 (Nov. 2020), p. 022048. DOI: 10.1088/1755-1315/588/2/022048. URL: <https://doi.org/10.1088/1755-1315/588/2/022048>.
- [HHL+18] Pashupati Hegde, Markus Heinonen, Harri Lähdesmäki, and Samuel Kaski. “Deep Learning with Differential Gaussian Process Flows”. In: (Oct. 2018). arXiv: 1810.04066 [cs, stat].
- [HK06] Rob J. Hyndman and Anne B. Koehler. “Another Look at Measures of Forecast Accuracy”. In: *International Journal of Forecasting* 22.4 (Oct. 2006), pp. 679–688. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2006.03.001.

Bibliography

- [HLF+18] Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. “Learning Latent Dynamics for Planning from Pixels”. In: *CoRR* abs/1811.04551 (2018). eprint: 1811.04551.
- [HMW+20] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [Ho95] Tin Kam Ho. “Random decision forests”. In: *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. Vol. 1. IEEE Computer Society, 1995, pp. 278–282. ISBN: 0818671289. DOI: 10.1109/ICDAR.1995.598994.
- [HR20] Max Roser Hannah Ritchie and Pablo Rosado. “Energy”. In: *Our World in Data* (2020). <https://ourworldindata.org/energy>.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: <https://doi.org/10.1162/neco.1997.9.8.1735> (visited on 01/14/2021).
- [HSH+19] Lars Ivar Hatledal, Arne Styve, Geir Hovland, and Houxiang Zhang. “A Language and Platform Independent Co-Simulation Framework Based on the Functional Mock-Up Interface”. In: *IEEE Access* 7 (2019), pp. 109328–109339. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2933275.
- [HW96] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 14. Springer-Verlag Berlin Heidelberg, 1996. ISBN: 3-540-60452-9.
- [HZR+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: (Dec. 2015). arXiv: 1512.03385 [cs].
- [HZS+18] Lars Ivar Hatledal, Houxiang Zhang, Arne Styve, and Geir Hovland. “FMI4j: A Software Package for Working with Functional Mock-up Units on the Java Virtual Machine”. In: *The 59th Conference on Simulation and Modelling (SIMS 59), 26-28 September 2018, Oslo Metropolitan University, Norway*. Nov. 2018, pp. 37–42. DOI: 10.3384/ecp1815337.
- [Inf20] InfluxData Inc. *InfluxDB glossary*. <https://docs.influxdata.com/influxdb/v1.8/concepts/glossary/>. Accessed 2022.11.05. 2020.
- [JB19] Junteng Jia and Austin R. Benson. “Neural Jump Stochastic Differential Equations”. In: *CoRR* abs/1905.10403 (2019). arXiv: 1905.10403.
- [JBB+20] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. “Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users”. In: (July 2020). arXiv: 2007.06823 [cs, stat].

- [JBS+19] Mohammadreza Javadiha, Joaquim Blesa, Adria Soldevila, and Vicenc Puig. “Leak Localization in Water Distribution Networks using Deep Learning”. In: Institute of Electrical and Electronics Engineers (IEEE), Sept. 2019, pp. 1426–1431. DOI: 10.1109/codit.2019.8820627.
- [JLG16] Bin Jiang, Jun Li, and Hua Guo. “Potential Energy Surfaces from High Fidelity Fitting of Ab Initio Points: The Permutation Invariant Polynomial - Neural Network Approach”. In: *International Reviews in Physical Chemistry* 35.3 (2016), pp. 479–506. DOI: 10.1080/0144235X.2016.1200347. eprint: <https://doi.org/10.1080/0144235X.2016.1200347>.
- [JWC+20] Weile Jia, Han Wang, Mohan Chen, Denghui Lu, Lin Lin, Roberto Car, Weinan E, and Linfeng Zhang. *Pushing the Limit of Molecular Dynamics with Ab Initio Accuracy to 100 Million Atoms with Machine Learning*. 2020. arXiv: 2005.00223 [physics.comp-ph].
- [JY08] Jeen-Shing Wang and Yi-Chung Chen. “A Hammerstein-Wiener Recurrent Neural Network with Universal Approximation Capability”. In: *2008 IEEE International Conference on Systems, Man and Cybernetics*. Oct. 2008, pp. 1832–1837. DOI: 10.1109/ICSMC.2008.4811555.
- [JZK+20] Pengzhan Jin, Aiqing Zhu, George Em Karniadakis, and Yifa Tang. “Symplectic Networks: Intrinsic Structure-Preserving Networks for Identifying Hamiltonian Systems”. In: *CoRR* abs/2001.03750 (2020). arXiv: 2001.03750.
- [KAF+17] Anuj Karpatne, Gowtham Atluri, James H. Faghmous, Michael Steinbach, Arindam Banerjee, Auroop Ganguly, Shashi Shekhar, Nagiza Samatova, and Vipin Kumar. “Theory-Guided Data Science: A New Paradigm for Scientific Discovery from Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.10 (Oct. 2017), pp. 2318–2331. ISSN: 1041-4347. DOI: 10.1109/TKDE.2017.2720168.
- [KBJ+20] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. *Learning Differential Equations That Are Easy to Solve*. 2020. arXiv: 2007.04504 [cs.LG].
- [KCL20] Patrick Kidger, Ricky T. Q. Chen, and Terry Lyons. ““Hey, That’s Not an ODE”: Faster ODE Adjoints with 12 Lines of Code”. In: (Sept. 2020). arXiv: 2009.09457 [cs, math].
- [KFW+18] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. *Neural Relational Inference for Interacting Systems*. 2018. arXiv: 1802.04687 [stat.ML].
- [KP20] Slawomir Koziel and Anna Pietrenko-Dabrowska. “Basics of Data-Driven Surrogate Modeling”. In: *Performance-Driven Surrogate Modeling of High-Frequency Structures*. Cham: Springer International Publishing, 2020, pp. 23–58. ISBN: 978-3-030-38925-3. DOI: 10.1007/978-3-030-38926-0_2.

Bibliography

- [KP92] Peter E Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. 1992. ISBN: 978-3-662-12616-5.
- [KPL+18] Jiheon Kang, Youn Jong Park, Jaeho Lee, Soo Hyun Wang, and Doo Seop Eom. “Novel leakage detection by ensemble CNN-SVM and graph-based localization in water distribution systems”. In: *IEEE Transactions on Industrial Electronics* 65.5 (May 2018), pp. 4279–4289. ISSN: 02780046. DOI: 10.1109/TIE.2017.2764861.
- [KS14] Andreas Kroll and Horst Schulte. “Benchmark Problems for Nonlinear System Identification and Control Using Soft Computing Methods: Need and Overview”. In: *Applied Soft Computing* 25 (2014), pp. 496–513. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2014.08.034.
- [KSS15] Rahul G. Krishnan, Uri Shalit, and David Sontag. *Deep Kalman Filters*. 2015. arXiv: 1511.05121 [stat.ML].
- [KSS16a] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. “ExploreKit: Automatic Feature Generation and Selection”. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. Barcelona, Spain: IEEE, Dec. 2016, pp. 979–984. ISBN: 978-1-5090-5473-2. DOI: 10.1109/ICDM.2016.0123.
- [KSS16b] Rahul G. Krishnan, Uri Shalit, and David Sontag. “Structured Inference Networks for Nonlinear State Space Models”. In: (Dec. 2016). arXiv: 1609.09869 [cs, stat].
- [KSS16c] Rahul G. Krishnan, Uri Shalit, and David Sontag. “Structured Inference Networks for Nonlinear State Space Models”. In: (Dec. 2016). arXiv: 1609.09869 [cs, stat].
- [KSS17] R. Krishnan, U. Shalit, and D. Sontag. “Structured Inference Networks for Nonlinear State Space Models”. In: *AAAI*. 2017.
- [KV15] James Max Kanter and Kalyan Veeramachaneni. “Deep Feature Synthesis: Towards Automating Data Science Endeavors”. In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Campus des Cordeliers, Paris, France: IEEE, Oct. 2015, pp. 1–10. ISBN: 978-1-4673-8272-4. DOI: 10.1109/DSAA.2015.7344858.
- [KWV+06] Gaëtan Kerschen, Keith Worden, Alexander F. Vakakis, and Jean-Claude Golinval. “Past, Present and Future of Nonlinear System Identification in Structural Dynamics”. In: *Mechanical Systems and Signal Processing* 20.3 (2006), pp. 505–592. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2005.04.008.
- [Lee08] Edward A. Lee. “Cyber Physical Systems: Design Challenges”. In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. Orlando, FL, USA: IEEE, May 2008, pp. 363–369. ISBN: 978-0-7695-3132-8. DOI: 10.1109/ISORC.2008.25. (Visited on 04/24/2023).

- [LeV07] Randall J LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Vol. 98. Siam, 2007. ISBN: 0-89871-629-2.
- [LFW+16] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzson, Jörg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Stylianos Basagiannis, and Andrey Sadovykh. “Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project”. In: *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. 2016, pp. 1–6. DOI: 10.1109/CPSData.2016.7496424.
- [LHC+19] Dong Liu, Antoine Honoré, Saikat Chatterjee, and Lars K Rasmussen. “Powering Hidden Markov Model by Neural Network Based Generative Models”. In: *arXiv preprint arXiv:1910.05744* (2019). arXiv: 1910.05744.
- [Lig17] Roger Light. “Mosquitto: server and client implementation of the MQTT protocol”. In: *The Journal of Open Source Software* 2 (May 2017). DOI: 10.21105/joss.00265.
- [LKS15] I. Lenz, Ross A. Knepper, and A. Saxena. “DeepMPC: Learning Deep Latent Features for Model Predictive Control”. In: *Robotics: Science and Systems*. 2015.
- [LP21] Kookjin Lee and Eric J. Parish. “Parameterized Neural Ordinary Differential Equations: Applications to Computational Physics Problems”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 477.2253 (Sept. 2021), p. 20210162. DOI: 10.1098/rspa.2021.0162.
- [LRP19] Michael Lutter, Christian Ritter, and Jan Peters. “Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning”. In: (July 2019). arXiv: 1907.04490 [cs, eess, stat].
- [LS17] Edward A. Lee and Sanjit A. Seshia. *Introduction to embedded systems: a cyber-physical systems approach*. Second edition. Cambridge, Massachusetts: MIT Press, 2017. ISBN: 9780262533812.
- [LSS+22] Christian Møldrup Legaard, Thomas Schranz, Gerald Schweiger, Ján Drgoňa, Basak Falay, Cláudio Gomes, Alexandros Iosifidis, Mahdi Abkar, and Peter Gorm Larsen. “Constructing Neural Network-Based Models for Simulating Dynamical Systems”. In: *ACM Computing Surveys* (Nov. 2022), p. 3567591. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3567591.
- [LTS+21a] Christian Legaard, Daniella Tola, Thomas Schranz, Hugo Macedo, and Peter Larsen. “A Universal Mechanism for Implementing Functional Mock-up Units”. In: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SciTePress, 2021, pp. 121–129. ISBN: 978-989-758-528-9. DOI: 10.5220/0010577601210129.

Bibliography

- [LTS+21b] Christian Legaard, Daniella Tola, Thomas Schranz, Hugo Macedo, and Peter Larsen. “A Universal Mechanism for Implementing Functional Mock-up Units.” in: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Online Streaming, — Select a Country —: SCITEPRESS - Science and Technology Publications, 2021, pp. 121–129. ISBN: 978-989-758-528-9. DOI: 10.5220/0010577601210129.
- [LWC+20] Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. *Scalable Gradients for Stochastic Differential Equations*. 2020. arXiv: 2001.01328 [cs.LG].
- [LWT+18] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. “Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids”. In: *CoRR* abs/1810.01566 (2018). arXiv: 1810.01566.
- [LWZ+18] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake. “Propagation Networks for Model-Based Control under Partial Observation”. In: *CoRR* abs/1809.11169 (2018). arXiv: 1809.11169.
- [LWZ12] Yanli Liu, Yourong Wang, and Jian Zhang. “New Machine Learning Algorithm: Random Forest”. In: *Information Computing and Applications*. Ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Baoxiang Liu, Maode Ma, and Jincai Chang. Vol. 7473. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 246–252. ISBN: 978-3-642-34061-1. DOI: 10.1007/978-3-642-34062-8_32.
- [LXS+19a] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. *Neural SDE: Stabilizing Neural ODE Networks with Stochastic Noise*. 2019. arXiv: 1906.02355 [cs.LG].
- [LXS+19b] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. “Neural SDE: Stabilizing Neural ODE Networks with Stochastic Noise”. In: (June 2019). arXiv: 1906.02355 [cs, stat].
- [MAA19] Mohammed Kyari Mustafa, Tony Allen, and Kofi Appiah. “A Comparative Review of Dynamic Neural Networks and Hidden Markov Model Methods for Mobile On-Device Speech Recognition”. In: *Neural Computing and Applications* 31.2 (2019), pp. 891–899.
- [Mak93] Spyros Makridakis. “Accuracy Measures: Theoretical and Practical Concerns”. In: *International Journal of Forecasting* 9.4 (Dec. 1993), pp. 527–529. ISSN: 01692070. DOI: 10.1016/0169-2070(93)90079-3.

- [MAL15] George Montanez, Saeed Amizadeh, and Nikolay Laptev. “Inertial Hidden Markov Models: Modeling Change in Multivariate Time Series”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 2015.
- [MAP+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [Mar19] Bernard Marr. *Artificial intelligence in practice: how 50 successful companies used AI and machine learning to solve problems*. John Wiley & Sons, 2019.
- [MB18] D. Masti and A. Bemporad. “Learning Nonlinear State-Space Models Using Deep Autoencoders”. In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 3862–3867.
- [Mea86] Nigel Meade. “Long Range Forecasting: From Crystal Ball to Computer (2nd Edition)”. In: *Journal of the Operational Research Society* 37.5 (May 1986), pp. 533–535. ISSN: 0160-5682, 1476-9360. DOI: 10.1057/jors.1986.91.
- [MH00] Spyros Makridakis and Michèle Hibon. “The M3-Competition: Results, Conclusions and Implications”. In: *International Journal of Forecasting* 16.4 (Oct. 2000), pp. 451–476. ISSN: 01692070. DOI: 10.1016/S0169-2070(00)00057-1.
- [MH16] Mesfin M. Mekonnen and Arjen Y. Hoekstra. “Sustainability: Four billion people facing severe water scarcity”. In: *Science Advances* 2.2 (Feb. 2016). ISSN: 23752548. DOI: 10.1126/sciadv.1500323.
- [MNG+14] Elena Mocanu, Phuong H. Nguyen, Madeleine Gibescu, and Wil L. Kling. “Comparison of Machine Learning Methods for Estimating Energy Consumption in Buildings”. In: *2014 International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*. Durham, United Kingdom: IEEE, July 2014, pp. 1–6. ISBN: 978-1-4799-3561-1. DOI: 10.1109/PMAPS.2014.6960635.
- [MPB+20] Stefano Massaroli, Michael Poli, Michelangelo Bin, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. *Stable Neural Flows*. 2020. arXiv: 2003.08063 [cs.LG].
- [MPP+21] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. *Dissecting Neural ODEs*. 2021. arXiv: 2002.08071 [cs.LG].
- [MW01] J. E. Marsden and M. West. “Discrete Mechanics and Variational Integrators”. In: *Acta Numerica* 10 (May 2001), pp. 357–514. ISSN: 0962-4929, 1474-0508. DOI: 10.1017/S096249290100006X.

Bibliography

- [NBD+20] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. *On Second Order Behaviour in Augmented Neural ODEs*. 2020. arXiv: 2006.07220 [cs.LG].
- [NGM+16] Anne H. H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Michael Z. Sheng. “IoT Middleware: A Survey on Issues and Enabling Technologies”. In: *IEEE Internet of Things Journal* (2016), pp. 1–1. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2615180.
- [OGJ+16a] Olalekan Ogunmolu, Xuejun Gu, Steve Jiang, and Nicholas Gans. “Non-linear Systems Identification Using Deep Dynamic Neural Networks”. In: (Oct. 2016). arXiv: 1610.01439 [cs].
- [OGJ+16b] Olalekan P. Ogunmolu, Xuejun Gu, Steve B. Jiang, and Nicholas R. Gans. “Nonlinear Systems Identification Using Deep Dynamic Neural Networks”. In: *CoRR* abs/1610.01439 (2016). arXiv: 1610.01439.
- [OKH+20] Katharina Ott, Prateek Katiyar, Philipp Hennig, and Michael Tiemann. “When Are Neural ODE Solutions Proper ODEs?” In: (July 2020). arXiv: 2007.15386 [cs, stat].
- [OMA21] OMA SpecWorks. <https://omaspecworks.org>. Accessed 2022.10.04. 2021.
- [OVV20] Viktor Oganessian, Alexandra Volokhova, and Dmitry Vetrov. “Stochasticity in Neural ODEs: An Empirical Study”. In: (June 2020). arXiv: 2002.09779 [cs, stat].
- [Pal22] Pallets. *Deploying to Production*. <https://flask.palletsprojects.com/en/2.2.x/deploying/>. Accessed 2022.10.07. 2022.
- [PMP+20] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. *Graph Neural Ordinary Differential Equations*. 2020. arXiv: 1911.07532 [cs.LG].
- [PVG+11] 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”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [QBT19] Meng Qu, Yoshua Bengio, and Jian Tang. “Gmnn: Graph Markov Neural Networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5241–5250.
- [QGM+20] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. *SNODE: Spectral Discretization of Neural ODEs for System Identification*. 2020. arXiv: 1906.07038 [cs.NE].

- [QWX19] Tong Qin, Kailiang Wu, and Dongbin Xiu. “Data Driven Governing Equations Approximation Using Deep Neural Networks”. In: *Journal of Computational Physics* 395 (Oct. 2019), pp. 620–635. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.06.042.
- [Ray19] Susmita Ray. “A Quick Review of Machine Learning Algorithms”. In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. Faridabad, India: IEEE, Feb. 2019, pp. 35–39. ISBN: 978-1-72810-211-5. DOI: 10.1109/COMITCon.2019.8862451.
- [RBD+20] Ribana Roscher, Bastian Bohn, Marco F. Duarte, and Jochen Garcke. “Explainable Machine Learning for Scientific Insights and Discoveries”. In: *IEEE Access* 8 (2020), pp. 42200–42216. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2976199.
- [RCF+20] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. “Temporal Graph Networks for Deep Learning on Dynamic Graphs”. In: *CoRR* abs/2006.10637 (2020). arXiv: 2006.10637.
- [RDH+17] Caleb Robinson, Bistra Dilkina, Jeffrey Hubbs, Wenwen Zhang, Subhrajit Guhathakurta, Marilyn A. Brown, and Ram M. Pendyala. “Machine Learning Approaches for Estimating Commercial Building Energy Consumption”. In: *Applied Energy* 208 (Dec. 2017), pp. 889–904. ISSN: 03062619. DOI: 10.1016/j.apenergy.2017.09.060.
- [RDK+19] David Rolnick et al. “Tackling Climate Change with Machine Learning”. In: (Nov. 2019). arXiv: 1906.05433 [cs, stat].
- [RGJ+22] Chris Rackauckas, Maja Gwozdz, Anand Jain, Yingbo Ma, Francesco Martinuzzi, Utkarsh Rajput, Elliot Saba, Viral B. Shah, Ranjan Anantharaman, Alan Edelman, Shashi Gowda, Avik Pal, and Chris Laughman. “Composing Modeling And Simulation With Machine Learning In Julia”. In: *2022 Annual Modeling and Simulation Conference (ANNSIM)*. San Diego, CA, USA: IEEE, July 2022, pp. 1–17. ISBN: 978-1-71385-288-9. DOI: 10.23919/ANNSIM55834.2022.9859453. (Visited on 05/02/2023).
- [RH18] Lars Ruthotto and Eldad Haber. “Deep Neural Networks Motivated by Partial Differential Equations”. In: (Dec. 2018). arXiv: 1804.04272 [cs, math, stat].
- [RH20] Lars Ruthotto and Eldad Haber. “Deep Neural Networks Motivated by Partial Differential Equations”. In: *Journal of Mathematical Imaging and Vision* 62.3 (Apr. 2020), pp. 352–364. ISSN: 0924-9907, 1573-7683. DOI: 10.1007/s10851-019-00903-1.

Bibliography

- [RJB+19] Martin Rätz, Amir Pasha Javadi, Marc Baranski, Konstantin Finkbeiner, and Dirk Müller. “Automated data-driven modeling of building energy systems via machine learning algorithms”. In: *Energy and Buildings* 202 (Nov. 2019), p. 109384. ISSN: 03787788. DOI: 10.1016/j.enbuild.2019.109384.
- [RM16] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. 2016. arXiv: 1505.05770 [stat.ML].
- [RMP+16] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhan Clarke. “Middleware for Internet of Things: A Survey”. In: *IEEE Internet of Things Journal* 3.1 (Feb. 2016), pp. 70–95. ISSN: 2327-4662. DOI: 10.1109/JIOT.2015.2498900.
- [RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: *International Conference on Machine Learning*. PMLR. 2014, pp. 1278–1286.
- [RPK18] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Multistep Neural Networks for Data-Driven Discovery of Nonlinear Dynamical Systems”. In: (Jan. 2018). arXiv: 1801.01236 [nlin, physics:physics, stat].
- [RPK19] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations”. In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. ISSN: 00219991. DOI: 10.1016/j.jcp.2018.10.045.
- [RS20] R. Rai and C. K. Sahu. “Driven by Data or Derived through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques with Cyber-Physical System (CPS) Focus”. In: *IEEE Access* 8 (2020), pp. 71050–71073.
- [RSG+18] Syama S. Rangapuram, Matthias W. Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. “Deep State Space Models for Time Series Forecasting”. In: *Advances in Neural Information Processing Systems* 31. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 7785–7794.
- [RTB12] Saman Razavi, Bryan A. Tolson, and Donald H. Burn. “Review of Surrogate Modeling in Water Resources: REVIEW”. In: *Water Resources Research* 48.7 (July 2012). ISSN: 00431397. DOI: 10.1029/2011WR011527.
- [Rud19] Cynthia Rudin. “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead”. In: *Nature Machine Intelligence* 1.5 (May 2019), pp. 206–215. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0048-x.

- [RYK20] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. “Hidden Fluid Mechanics: Learning Velocity and Pressure Fields from Flow Visualizations”. In: *Science* 367.6481 (Feb. 2020), pp. 1026–1030. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aaw4741.
- [SAH+22] Thomas Schranz, Qamar Alfalouji, Thomas Hirsch, and Gerald Schweiger. “An Open IoT Platform: Lessons Learned from a District Energy System”. In: *2022 Second International Conference on Sustainable Mobility Applications, Renewables and Technology (SMART)*. Cassino, Italy: IEEE, Nov. 2022, pp. 1–9. ISBN: 978-1-66547-146-6. DOI: 10.1109/SMART55236.2022.9990228.
- [SBC+19] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter W. Battaglia. “Hamiltonian Graph Networks with ODE Integrators”. In: *CoRR* abs/1909.12790 (2019). arXiv: 1909.12790.
- [SBS+13] Maxim Vladimirovich Shcherbakov, Adriaan Brebels, Nataliya Lvovna Shcherbakova, Anton Pavlovich Tyukov, Timur Alexandrovich Janovsky, Valeriy Anatol’evich Kamaev, et al. “A survey of forecast error measures”. In: *World applied sciences journal* 24.24 (2013), pp. 171–176.
- [Sch21] Matthew D. Schwartz. “Modern Machine Learning and Particle Physics”. In: (2021). DOI: 10.48550/ARXIV.2103.12226.
- [SCS+20] Thomas Schranz, Katja Corcoran, Thomas Schwengler, Lisa Eckersdorfer, and Gerald Schweiger. “Mobile Application For Active Consumer Participation in Building Energy Systems”. In: *BauSIM 2020 - 8th Conference of IBPSA Germany and Austria, Proceedings*. Ed. by Monsberger, Michael, Hopfe, Christina Johanna, Krüger, Markus, and Passer, Alexander. 2020, pp. 281–287.
- [SDT20] Elliott Skomski, Jan Drgona, and Aaron Tuor. *Physics-Informed Neural State Space Models via Learning and Evolution*. 2020. arXiv: 2011.13497 [cs.NE].
- [SEL+21] Thomas Schranz, Johannes Exenberger, Christian Møldrup Legaard, Ján Drgoňa, and Gerald Schweiger. “Energy Prediction under Changed Demand Conditions: Robust Machine Learning Models and Input Feature Combinations”. In: *17th International Conference of the International Building Performance Simulation Association (Building Simulation 2021)*. 2021.
- [SEM+21] Gerald Schweiger, Johannes Exenberger, Avichal Malhotra, Thomas Schranz, Theresa Boiger, C Van Treeck, and James O’Donnell. “Data shortage for urban energy simulations? An empirical survey on data availability and enrichment methods using machine learning?” In: *28th International Workshop on Intelligent Computing in Engineering*. Universitätsverlag der TU Berlin. 2021, pp. 301–309. DOI: 10.14279/depositonce-12021.

Bibliography

- [SGE+19] G. Schweiger, C. Gomes, G. Engel, I. Hafner, J. Schoeggel, A. Posch, and T. Nouidui. “An Empirical Survey on Co-Simulation: Promising Standards, Challenges and Research Needs”. In: *Simulation Modelling Practice and Theory* 95 (Sept. 2019), pp. 148–163. ISSN: 1569190X. DOI: 10.1016/j.simpat.2019.05.001.
- [SGP+20] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. “Learning to Simulate Complex Physics with Graph Networks”. In: *CoRR* abs/2002.09405 (2020). arXiv: 2002.09405.
- [SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.
- [SHF20a] Ying Sun, Fariborz Haghighat, and Benjamin C. M. Fung. “A review of the-state-of-the-art in data-driven approaches for building energy prediction”. en. In: *Energy and Buildings* 221 (2020), p. 110022. ISSN: 0378-7788. (Visited on 01/04/2021).
- [SHF20b] Ying Sun, Fariborz Haghighat, and Benjamin C.M. Fung. “A Review of The-State-of-the-Art in Data-Driven Approaches for Building Energy Prediction”. In: *Energy and Buildings* 221 (Aug. 2020), p. 110022. ISSN: 03787788. DOI: 10.1016/j.enbuild.2020.110022.
- [SHS+18] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter W. Battaglia. “Graph Networks as Learnable Physics Engines for Inference and Control”. In: *CoRR* abs/1806.01242 (2018). arXiv: 1806.01242.
- [SKS+17] Kristof T. Schütt, Pieter-Jan Kindermans, Huziel E. Saucedo, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. *SchNet: A Continuous-Filter Convolutional Neural Network for Modeling Quantum Interactions*. 2017. arXiv: 1706.08566 [stat.ML].
- [SL19] Johan Schoukens and Lennart Ljung. “Nonlinear System Identification: A User-Oriented Roadmap”. In: *CoRR* abs/1902.00683 (2019). arXiv: 1902.00683.
- [SMT+21] Thomas Schranz, Christian Møldrup Legaard, Daniella Tola, and Gerald Schweiger. “Portable Runtime Environments for Python-based FMUs: Adding Docker Support to UniFMU”. In: *14th Modelica Conference 2021*. Sept. 2021, pp. 419–424. DOI: 10.3384/ecp21181419.
- [SN17] M. Schoukens and J.P. Noël. “Three Benchmarks Addressing Open Challenges in Nonlinear System Identification**We Thank Torbjorn Wigren and Per Mattsson (Uppsala University, Sweden) for Their Help in Realizing the Cascaded Tanks Benchmark. This Work Was Funded by the Fund for Scientific Research (FWO), the Methusalem Grant of the Flemish

- Government (METH-1), the IAP VII/19 DYSCO Program, and the ERC Advanced Grant SNLSID under Contract 320378. The Author J.P. Noel Is a Postdoctoral Researcher of the Fonds de La Recherche Scientifique - FNRS Which Is Gratefully Acknowledged.” In: *IFAC-PapersOnLine* 50.1 (2017), pp. 446–451. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2017.08.071.
- [SRG+18] Saleh Seyedzadeh, Farzad Pour Rahimian, Ivan Glesk, and Marc Roper. “Machine Learning for Estimation of Building Energy Consumption and Performance: A Review”. In: *Visualization in Engineering* 6.1 (Dec. 2018), p. 5. ISSN: 2213-7459. DOI: 10.1186/s40327-018-0064-7.
- [SS18] Pramila P. Shinde and Seema Shah. “A Review of Machine Learning and Deep Learning Applications”. In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBE)*. Pune, India: IEEE, Aug. 2018, pp. 1–6. ISBN: 978-1-5386-5257-2. DOI: 10.1109/ICCUBE.2018.8697857.
- [SSB+19] Priyadarshi R Shukla, J Skeg, E Calvo Buendia, Valérie Masson-Delmotte, H-O Pörtner, DC Roberts, Panmao Zhai, Raphael Slade, Sarah Connors, S van Diemen, et al. *Climate Change and Land: an IPCC special report on climate change, desertification, land degradation, sustainable land management, food security, and greenhouse gas fluxes in terrestrial ecosystems*. Tech. rep. 2019.
- [SSP+20] Thomas Schranz, Gerald Schweiger, Siegfried Pabst, and Franz Wotawa. “Machine Learning for Water Supply Supervision”. In: *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices*. Ed. by Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 238–249. ISBN: 978-3-030-55789-8. DOI: 10.1007/978-3-030-55789-8_21.
- [Sto16] Peter Stott. “How Climate Change Affects Extreme Weather Events”. In: *Science* 352.6293 (June 2016), pp. 1517–1518. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aaf7271.
- [SVW+21] Elliott Skomski, Soumya Vasisht, Colby Wight, Aaron Tuor, Jan Drgona, and Draguna Vrabie. *Constrained Block Nonlinear Neural Dynamical Models*. 2021. arXiv: 2101.01864 [math.DS].
- [SWL+16] Heung-Il Suk, Chong-Yaw Wee, Seong-Whan Lee, and Dinggang Shen. “State-Space Model with Deep Learning for Functional Dynamics Estimation in Resting-State fMRI”. In: *NeuroImage* 129 (2016), pp. 292–307. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2016.01.005.
- [TCD+21] Thomas Schranz, Christian Møldrup Legaard, Daniella Tola, and Gerald Schweiger. “Portable Runtime Environments for Python-based FMUs: Adding Docker Support to UniFMU”. In: *14th Modelica Conference 2021*. Sept. 2021, pp. 419–424. DOI: 10.3384/ecp21181419.

Bibliography

- [tea20] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: <https://doi.org/10.5281/zenodo.3509134>.
- [Tec21] Graz University of Technology. *Buildings and Technical Support*. <https://www.tugraz.at/en/tu-graz/organisational-structure/service-departments-and-staff-units/buildings-and-technical-support>. Accessed 2021.11.06. 2021.
- [TFL+21] Ellika Taveres-Cachat, Fabio Favoino, Roel Loonen, and Francesco Goia. “Ten Questions Concerning Co-Simulation for Performance Prediction of Advanced Building Envelopes”. In: *Building and Environment* 191 (Mar. 2021), p. 107570. ISSN: 03601323. DOI: 10.1016/j.buildenv.2020.107570.
- [TGR+19a] Fernando Terroso-Saenz, Aurora González-Vidal, Alfonso P. Ramallo-González, and Antonio F. Skarmeta. “An Open IoT Platform for the Management and Analysis of Energy Data”. In: *Future Generation Computer Systems* 92 (Mar. 2019), pp. 1066–1079. ISSN: 0167739X. DOI: 10.1016/j.future.2017.08.046.
- [TGR+19b] Fernando Terroso-Saenz, Aurora González-Vidal, Alfonso P. Ramallo-González, and Antonio F. Skarmeta. “An open IoT platform for the management and analysis of energy data”. In: *Future Generation Computer Systems* 92 (Mar. 2019), pp. 1066–1079. DOI: 10.1016/j.future.2017.08.046. URL: <https://doi.org/10.1016/j.future.2017.08.046>.
- [TRJ+20] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. “Hamiltonian Generative Networks”. In: (Feb. 2020). arXiv: 1909.13789 [cs, stat].
- [UM18] Oliver T. Unke and Markus Meuwly. “A Reactive, Scalable, and Transferable Model for Molecular Energies from a Neural Network Approach Based on Local Information”. In: *The Journal of Chemical Physics* 148.24 (2018), p. 241708. DOI: 10.1063/1.5017898. eprint: <https://doi.org/10.1063/1.5017898>.
- [UM19] Oliver T. Unke and Markus Meuwly. “PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges”. In: *Journal of Chemical Theory and Computation* 15.6 (2019), pp. 3678–3693. DOI: 10.1021/acs.jctc.9b00181. eprint: <https://doi.org/10.1021/acs.jctc.9b00181>.
- [Uni20] United Nations Environment Programme. *2020 Global Status Report for Buildings and Construction: Towards a Zero-emission, Efficient and Resilient Buildings and Construction Sector*. Tech. rep. 2020. URL: <http://globalabc.org/>.

- [VGO+20] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.
- [VGS+00] Charles J. Vörösmarty, Pamela Green, Joseph Salisbury, and Richard B. Lammers. “Global water resources: Vulnerability from climate change and population growth”. In: *Science* 289.5477 (July 2000), pp. 284–288. ISSN: 00368075. DOI: 10.1126/science.289.5477.284.
- [vMB+21] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Michal Walczak, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. “Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2021.3079836.
- [vMS+20] Laura von Rueden, Sebastian Mayer, Rafet Sifa, Christian Bauckhage, and Jochen Garcke. “Combining Machine Learning and Simulation to a Hybrid Modelling Approach: Current and Future Directions”. In: *Advances in Intelligent Data Analysis XVIII*. Ed. by Michael R. Berthold, Ad Feelders, and Georg Kreml. Vol. 12080. Cham: Springer International Publishing, 2020, pp. 548–560. ISBN: 978-3-030-44583-6. DOI: 10.1007/978-3-030-44584-3_43.
- [WE19] Paul Westermann and Ralph Evins. “Surrogate Modelling for Sustainable Building Design – A Review”. In: *Energy and Buildings* 198 (Sept. 2019), pp. 170–186. ISSN: 03787788. DOI: 10.1016/j.enbuild.2019.05.057.
- [Wet11] Michael Wetter. “Co-Simulation of Building Energy and Control Systems with the Building Controls Virtual Test Bed”. In: *Journal of Building Performance Simulation* 4.3 (Sept. 2011), pp. 185–203. ISSN: 1940-1493, 1940-1507. DOI: 10.1080/19401493.2010.518631.
- [WFA+22] Sandra Wilfling, Basak Falay, Qamar Alfalouji, Johannes Exenberger, Thomas Schranz, Mina Basirat, and Gerald Schweiger. “Smart Energy Systems Modeling - Component Exchange during Simulation”. English. In: *Proceedings from the 2nd International Sustainable Energies Conference 2022*. Apr. 2022.
- [WH91] G. Wanner and E. Hairer. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer S. Vol. 1. Springer-Verlag, 1991.
- [WM05] Cj Willmott and K Matsuura. “Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in Assessing Average Model Performance”. In: *Climate Research* 30 (2005), pp. 79–82. ISSN: 0936-577X, 1616-1572. DOI: 10.3354/cr030079.

Bibliography

- [WME+13] Edmund Widl, Wolfgang Muller, Atiyah Elsheikh, Matthias Hortenhuber, and Peter Palensky. “The FMI Library: A High-Level Utility Package for FMI for Model Exchange”. In: *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. Berkeley, CA, USA: IEEE, May 2013, pp. 1–6. ISBN: 978-1-4799-1307-7. DOI: 10.1109/MSCPES.2013.6623316.
- [WMP+18] Hao Wu, Andreas Mardt, Luca Pasquali, and Frank Noe. “Deep Generative Markov State Models”. In: *arXiv preprint arXiv:1805.07601* (2018). arXiv: 1805.07601.
- [Wor19] World Economic Forum. *The Global Risks Report 2019 14th Edition Insight Report*. 2019. ISBN: 9781944835156. URL: <http://wef.ch/risks2019>.
- [WOW+19] Jiang Wang, Simon Olsson, Christoph Wehmeyer, Adrià Pérez, Nicholas E. Charron, Gianni de Fabritiis, Frank Noé, and Cecilia Clementi. “Machine Learning of Coarse-Grained Molecular Dynamics Force Fields”. In: *ACS Central Science* 5.5 (2019), pp. 755–767. DOI: 10.1021/acscentsci.8b00913. eprint: <https://doi.org/10.1021/acscentsci.8b00913>.
- [WPC+19] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. “A Comprehensive Survey on Graph Neural Networks”. In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596.
- [Wri21] Sewall Wright. “Correlation and causation”. In: (1921).
- [WTP20] Sifan Wang, Yujun Teng, and Paris Perdikaris. “Understanding and Mitigating Gradient Pathologies in Physics-Informed Neural Networks”. In: (Jan. 2020). arXiv: 2001.04536 [cs, math, stat].
- [WXP+19] Yixuan Wei, Liang Xia, Song Pan, Jinshun Wu, Xingxing Zhang, Mengjie Han, Weiya Zhang, Jingchao Xie, and Qingping Li. “Prediction of Occupancy Level and Energy Consumption in Office Building Using Blind System Identification and Neural Networks”. In: *Applied Energy* 240 (Apr. 2019), pp. 276–294. ISSN: 03062619. DOI: 10.1016/j.apenergy.2019.02.056.
- [WYP20] Sifan Wang, Xinling Yu, and Paris Perdikaris. “When and Why PINNs Fail to Train: A Neural Tangent Kernel Perspective”. In: (July 2020). arXiv: 2007.14527 [cs, math, stat].
- [WZS+18] Yixuan Wei, Xingxing Zhang, Yong Shi, Liang Xia, Song Pan, Jinshun Wu, Mengjie Han, and Xiaoyun Zhao. “A review of data-driven approaches for prediction and classification of building energy consumption”. In: *Renewable and Sustainable Energy Reviews* 82 (2018), pp. 1027–1047. ISSN: 1364-0321. (Visited on 01/04/2021).

- [WZW+17] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. “Visual Interaction Networks: Learning a Physics Simulator from Video”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.
- [XCL+21] Winnie Xu, Ricky T. Q. Chen, Xuechen Li, and David Duvenaud. “Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations”. In: (Aug. 2021). arXiv: 2102.06559 [cs, stat].
- [XWV+05] M Xu, P Watanachaturaporn, P Varshney, and M Arora. “Decision Tree Regression for Soft Classification of Remote Sensing Data”. In: *Remote Sensing of Environment* 97.3 (Aug. 2005), pp. 322–336. ISSN: 00344257. DOI: 10.1016/j.rse.2005.05.008.
- [ZCH+20] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. “Graph Neural Networks: A Review of Methods and Applications”. In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2021.01.001.
- [ZCZ18] Ziwei Zhang, Peng Cui, and Wenwu Zhu. “Deep Learning on Graphs: A Survey”. In: *CoRR* abs/1812.04202 (2018). arXiv: 1812.04202.
- [ZDC19] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. “Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control”. In: *CoRR* abs/1909.12077 (2019). arXiv: 1909.12077.
- [ZHW+18a] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. “Deep Potential Molecular Dynamics: A Scalable Model with the Accuracy of Quantum Mechanics”. In: *Physical Review Letters* 120.14 (Apr. 2018), p. 143001. DOI: 10.1103/PhysRevLett.120.143001.
- [ZHW+18b] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and Weinan E. *End-to-End Symmetry Preserving Inter-Atomic Potential Energy Model for Finite and Extended Systems*. 2018. arXiv: 1805.09003 [physics.comp-ph].
- [ZM12] Hai-xiang Zhao and Frédéric Magoulès. “A Review on the Prediction of Building Energy Consumption”. In: *Renewable and Sustainable Energy Reviews* 16.6 (Aug. 2012), pp. 3586–3592. ISSN: 13640321. DOI: 10.1016/j.rser.2012.02.049.