Journal of Mathematics in Industry
a SpringerOpen Journal

SURVEY Open Access

Hybrid modeling design patterns

Maja Rudolph[1,2*], Stefan Kurz[3,4,5*] and Barbara Rakitsch[3*]

*Correspondence:
maja.rudolph@us.bosch.com;
stefan.kurz2@de.bosch.com;
barbara.rakitsch@de.bosch.com
[1]Bosch Center for AI, Pittsburgh, PA, USA
[3]Bosch Center for AI, Renningen, Germany
Full list of author information is available at the end of the article

Abstract

Design patterns provide a systematic way to convey solutions to recurring modeling challenges. This paper introduces design patterns for hybrid modeling, an approach that combines modeling based on first principles with data-driven modeling techniques. While both approaches have complementary advantages there are often multiple ways to combine them into a hybrid model, and the appropriate solution will depend on the problem at hand. In this paper, we provide four base patterns that can serve as blueprints for combining data-driven components with domain knowledge into a hybrid approach. In addition, we also present two composition patterns that govern the combination of the base patterns into more complex hybrid models. Each design pattern is illustrated by typical use cases from application areas such as climate modeling, engineering, and physics.

Keywords: Hybrid modeling; Physics-inspired AI; Design patterns

1 Introduction

Models play a crucial role in the scientific process by providing a representation of complex systems, processes, and phenomena. Models help scientists to make predictions, test hypotheses, and gain a deeper understanding of the behavior of these systems [1, 2]. By using mathematical models, such as physical, statistical, or simulation models, scientists can study the relationships between variables, estimate uncertainties, and explore scenarios without having to perform expensive or dangerous experiments [3, Ch. 1]. In this way, models serve as a powerful tool for advancing our knowledge and understanding of the world, and for solving real-world problems in fields such as medicine, engineering, and environmental science.

Traditionally, models are derived from first principles and encode domain knowledge such as physical laws or physical constraints. Such models emerge from the scientific process through a combination of observation, experimentation, and theoretical analysis. After careful observation of natural phenomena, scientists form hypotheses and theories to explain the observed behavior. These theories are then tested through experiments and compared with existing knowledge and models. If a theory withstands experimental scrutiny and provides accurate predictions, it may become accepted as a law or constraint. Models based on first principles are data-efficient, causal, lead to explainable predictions, are often more reliable than data-driven models since the underlying theory has been val-

Springer

idated, and predictions will generalize to other deployment regimes as long as the underlying assumptions of the model still hold.

Data-driven models, on the other hand, are a type of modeling approach that relies on large data sets to identify patterns and correlations in the data that can be used to make predictions or classifications [4, 5]. These models are often used in fields where the underlying physical processes are too complex to model by first-principles. Data-driven models are typically developed using machine learning techniques such as neural networks [6]. These models can be trained on large data sets of labeled and in some cases unlabeled data and can then be used to make predictions or classifications on new data. Data-driven models have shown promise in a wide range of applications, including image and speech recognition, natural language processing, and predictive modeling in finance and healthcare.

Hybrid models combine the strengths of both data-driven and first-principle based models, and can be useful in situations where neither approach alone is sufficient [7–10]. For example, mechanistic models are based on first principles and describe a hypothesized causal process between variables [11]. While they can provide a deep understanding of the underlying physics or biology of a system, they may not always capture all of the relevant details or interactions, leading to inaccuracies. On the other hand, data-driven models can accurately capture complex relationships in large data sets, but may not be able to explain the underlying mechanisms or provide insight into how the system behaves under new conditions. Hybrid models can combine the strengths of both approaches, allowing for more accurate and interpretable predictions even in complex systems with incomplete understanding of the underlying mechanisms.

Hybrid modeling is challenging because it requires expertise in both first-principle-based modeling and data-driven modeling, as well as knowledge of how to integrate the two approaches effectively. It can be difficult to determine the appropriate level of complexity for each component of the hybrid model and to ensure that the different components are compatible with each other. In particular, hybrid modeling requires careful consideration of the trade-offs between accuracy, complexity, interpretability, and scalability, which can be difficult to optimize.

Validating and verifying a hybrid model presents another challenge. Its data-driven and physics-based components may contribute different sources of uncertainty and error which need to be handled with care. For these reasons, designing and implementing a hybrid model requires careful consideration of the strengths and weaknesses of each modeling approach and a thorough understanding of the system being modeled.

The applications of hybrid modeling are incredibly diverse, spanning a wide range of fields and industries. From molecular modeling in drug discovery [12], to simulation tasks in climate [13] and earth science [14] and engineering, to modeling sensor data, hybrid modeling is used in many domains to address unique and complex challenges.

This diversity of applications means that there is a need for solutions that can be applied more broadly, rather than being specific to one particular domain. Developing such approaches requires a focus on abstraction and generalization, so that solutions can be formulated at a higher level of abstraction that can be applied across multiple domains. While literature surveys of hybrid modeling have introduced taxonomies of modeling approaches [8, 9], the contribution of this paper is to present different design patterns for composing data-driven and first-principle based models. The design patterns address re-

curring modeling challenges and distill useful solution approaches that generalize across applications.

Formalizing solutions to recurring modeling challenges into hybrid modeling design patterns provides several benefits. First, it allows for the sharing of knowledge and expertise across application domains, which can lead to faster progress and innovation. Second, it facilitates the development of standardized tools and techniques for hybrid modeling, which can improve the efficiency and reliability of the modeling process. Third, it can help identify common challenges and limitations in hybrid modeling, which can guide future research directions and advance the field as a whole. Overall, the use of hybrid modeling design patterns can improve the accessibility, efficiency, and effectiveness of hybrid modeling across a wide range of applications.

2 Background

In this background section, we introduce modeling and then review both the first-principles-based as well as the data-driven perspective on modeling.

2.1 Computational models

The goal of hybrid modeling is to build a computational model for a system of interest. A computational model is a set of computations that are applied to an input to produce an output. The model of a system can be used to make predictions about how the system would react to certain inputs or to study how the system behaves under certain conditions. Alternatively, the model can be used to simulate the system. Models typically *approximate* the behavior of the underlying system, which might be too complex to model more accurately.

An computational model is of the form

$$y = u(x). \tag{1}$$

The inputs x are manipulated by a function u to produce the outputs y. The functional form of u will depend on the model type. We distinguish between two different model types: The first type is models based on first principles, for example from physics. These are sometimes also called scientific models, and we often call them physical models. The second type of model is data-driven. Here one uses data to find a model within a class of functions that best explains the data. This function is then used as a model, e.g. to make predictions.

2.2 Modeling from first principles

When modeling from first principles, the choice of u is derived using scientific reasoning. There is a justification for both the functional form of u and for the choice of its parameters. For this reason, these models are often called models based on first-principles, mechanistic models, physics-based models or science-based models.

For example, laws of physics, such as Newton's laws of motion and the law of conservation of energy, emerged from centuries of observation and experimentation in the field of mechanics. These laws provide a mathematical framework for understanding and predicting the behavior of physical systems, and have been tested and confirmed through numerous experiments. Similarly, in chemistry, conservation laws, such as the law of conser-

vation of mass, emerged from the study of chemical reactions and provide a fundamental understanding of the behavior of chemical systems.

From a mathematical point of view, scientific models frequently take the form of *algebraic models*, *ordinary differential equations* (ODEs), *partial differential equations* (PDEs), or a combination of those.

2.2.1 Algebraic models

An algebraic mathematical model is a type of mathematical model that uses algebraic equations or functions to represent a real-world situation or system. In an algebraic model, the relationships between the variables are often represented using equations that involve elementary mathematical operations and functions.

One example is the equation for the trajectory of a stone that is vertically thrown in the air, where air resistance is neglected. The height $u(t)$ over ground as a function of time $t \geq 0$ is

$$u(t) = -0.5gt^2 + v_0 t + h_0, \tag{2}$$

where h_0 is the initial height, v_0 the initial velocity and g the gravitational constant.

From a computational perspective, this model could be utilized to compute – for a given instance t_1 – the height at this instance, $h_1 = u(t_1)$.

2.2.2 Ordinary differential equations (*ODEs*)

A more involved model class are differential equations. An ODE is a type of differential equation that involves only one independent variable, usually time t, and its derivatives.

ODE models are particularly useful for systems that involve dynamic behavior, where the behavior of the system changes over time in response to internal or external factors. In an ODE model, the behavior of a system is represented using one or more ODEs that describe the rates of change of the system's variables. The ODEs can be used to predict how the system will evolve over time, based on its initial conditions and the values of its parameters.

Solving an ODE involves finding a mathematical expression that describes the behavior of the system as a function of the independent variable, usually as a function of time. This can be done using various analytical or numerical methods, depending on the complexity of the system and the accuracy of the desired solution. A closed form solution of an ODE yields an algebraic model. For example, the algebraic model (2) is a solution to the ODE

$$\frac{\mathrm{d}^2 u(t)}{\mathrm{d}u^2} = -g,$$

subject to given initial conditions. This is just Newton's law, the first-principle based model that underlies the mechanistic model (2).

Once a solution has been obtained, it can be used to predict the behavior of the system under different conditions or to design interventions to achieve a desired outcome.

In the following, we will consider three additional ODE models that will serve as recurring examples throughout the remainder of the paper.

1. Let us start with the ODE of an *harmonic oscillator*

$$\frac{\mathrm{d}^2 u(t)}{\mathrm{d}t^2} = -u(t), \tag{3}$$

where $u(t)$ yields the normalized displacement at normalized time t. The normalization is with respect to some reference displacement s_0 and the oscillatory period T, respectively. For a spring-mass system with mass m and spring constant k the oscillatory period is $T = \sqrt{m/k}$. The model gets more interesting if a nonlinear damping term is added,

$$\frac{\mathrm{d}^2 u(t)}{\mathrm{d}t^2} = -u(t) + \mu \frac{\mathrm{d}u(t)}{\mathrm{d}t}\left(1 - u(t)^2\right), \tag{4}$$

where the positive real parameter μ determines the amount of nonlinear damping. Equation (4) is the *Van der Pol equation* [15, Sect. 5.7], which exhibits a number of interesting nonlinear phenomena, such as relaxation oscillations [16].

2. The *Lotka-Volterra* equations [17, Sect. 4.1] are used to model the population dynamics of two interacting species of a predator and its prey. The population density of prey is $u(t)$ and the population density of predators is $w(t)$. The population dynamics is modeled by the nonlinear *system of ODEs*

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = \alpha u(t) - \beta u(t) w(t), \qquad \frac{\mathrm{d}w(t)}{\mathrm{d}t} = \delta u(t) w(t) - \gamma w(t), \tag{5}$$

with positive real parameters α, β, γ, and δ determining the self and mutual interactions of the two species.

3. The simplest standard model for a dynamical system with several degrees of freedom is a *system of ODEs*, of the form

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = f\left(u(t), t; \theta\right), \tag{6}$$

where $u(t) \in \mathbb{R}^n$ describes the state of the system at time t, a point in an n-dimensional state space. Herein, $\theta \in \mathbb{R}^p$ is a p-dimensional parameter vector that admits calibrating the model. Given an initial condition $u(t_0)$ at time t_0, the dynamics of the system can be obtained by integrating the ODE system. At time $t_1 > t_0$ we obtain

$$u(t_1) = u(t_0) + \int_{t_0}^{t_1} f\left(u(t), t; \theta\right) \mathrm{d}t. \tag{7}$$

This representation clearly demonstrates that the dynamics of the system is entirely encoded in the function f, which assigns to each state $u(t)$ and time t the rate of change of this state. The structure of the function f is often dictated to us from physics, and the values of the parameters can be obtained from domain knowledge.

Moreover, given an actual numerical implementation of the function f there are several numerical methods, such as Runge-Kutta methods [3, Ch. 4 & 6], to integrate ODE systems. Only together with an integration method will an ODE system yield a computational model (Eq. (1)) for predicting future states.

2.2.3 Partial differential equations (*PDEs*)

A PDE is an equation for a function which depends on more than one independent variable. The equation involves the independent variables, the function, and partial derivatives

of the function, with respect to the independent variables. PDEs are ubiquitous in mathematical physics and foundational in several fields, such as acoustics, elasticity, electrodynamics, fluid dynamics, thermodynamics, general relativity, and quantum mechanics. The independent variables are often *space-time coordinates*, like (x, y, z, t).

As a simple example, we consider a scalar function u, which depends on the spatial coordinates (x, y, z), and the PDE

$$\frac{\partial^2 u(x,y,z)}{\partial x^2} + \frac{\partial^2 u(x,y,z)}{\partial y^2} + \frac{\partial^2 u(x,y,z)}{\partial z^2} = 0. \tag{8}$$

This is the *Laplace equation* in three dimensions. For example, if u denotes the scalar electric potential, (8) is the governing equation in electrostatics, for domains that are free of electrical charges.

To obtain a Computation model (Eq. (1)) for predicting the state of the system over time the PDE will need to be solved either analytically or numerically. Here the finite element method (FEM) is a popular choice [18], but many other methods exist [19].

2.3 Data-driven modeling

An alternative path for developing a model is data-centric. Given data in form of observations, a model is developed to be consistent with the observations, for example, reproducing the data as accurately as possible. There are many different data-driven approaches. Unlike the scientific models, which are chosen based on deductive reasoning, data-driven models are chosen based on their statistical and computational properties and their match to the requirements of the modeling problem at hand.

2.3.1 Data-driven calibration

Data-driven calibration is a methodological approach that leverages observed data in order to optimize the parameters of a given model. Consider, for example, the Lotka-Volterra equations, Eq. (5). In the context of data-driven calibration, the goal is to optimize the parameters α, β, γ, and δ based on observed data, to accurately capture the dynamics of the predator-prey system.

Traditionally, these parameters might be adjusted by specialists through a process of trial-and-error until the desired behavior is achieved. However, more systematic and efficient approaches to parameter identification are available [20]. Data-driven calibration can employ optimization algorithms, often utilizing a specific loss function (e.g., the mean-squared error) to guide the optimization process. For straightforward scenarios, standard least squares approaches can be effective [21], while for complex or non-differentiable problems, derivative-free optimization methods such as genetic algorithms [22], particle swarm optimization [23], and Bayesian optimization [24] offer valuable alternatives. Moreover, data-driven calibration is not limited to refining existing models; it can also facilitate the identification of physical systems from scratch [25, 26].

When considering uncertainty in the data, more sophisticated techniques, termed as Bayesian calibration or simulation-based inference, come into play [27, 28]. These methods do not merely estimate point values for the parameters but learn their posterior distribution, accounting for both aleatoric (inherent randomness) and epistemic (model uncertainty) factors. Furthermore, there are specialized methods designed for ordinary differential equations (ODEs), which improve algorithmic efficiency by utilizing their mathematical structure [29, 30].

While calibration focuses on refining model parameters to align predictions with observed data, standard machine learning techniques as we will discuss next aim to learn patterns directly from data without providing any physical interpretation.

2.3.2 Machine learning

Machine learning presents an approach for learning model parameters from data [4, 5]. While non-parametric approaches exist, a machine learning model often consists of a parameterized function $u(\cdot;\theta)$ with parameters θ, that can predict a response y from inputs x. Different parameter settings correspond to different functional relationships between the predictions $\hat{y} = u(x;\theta)$ and the inputs. The quality of a prediction, i.e. how closely a prediction $\hat{y}$ resembles a desired output y, can be measured in a *loss function* $l(x,y,\theta)$. In the supervised learning setting [5, Ch. 1.3], given a data set $\mathcal{D}$ of examples of x and y pairs, the optimal parameter setting is found by minimizing the loss, averaged over the training examples,

$$\theta^* = \min_\theta \frac{1}{|\mathcal{D}|} \sum_{x,y\in\mathcal{D}} l(x,y,\theta). \tag{9}$$

Machine learning approaches, are also applicable in the unsupervised setting [5, Ch. 1.3] where the training data only contains input samples x, but no labels. Common unsupervised modeling tasks include clustering, where the target label y would be the cluster assignment of an input, or anomaly detection, where the unknown label represents the likelihood that the input sample is an anomaly. For an overview of common machine learning tasks see Ch. 5.1.1 of [6].

2.3.2.1 Probabilistic modeling Probabilistic modeling [4, 5] refers to a class of machine learning methods where data points are treated as observations of random variables. Modeling consists of making assumptions about the underlying distributions from which these data points are drawn. The primary aim is to infer the parameters that characterize these distributions from the available data. Once the model is learned, it can be used to predict future observations, evaluate the likelihood of observed data, or provide uncertainty estimates regarding the outcomes.

In probabilistic modeling, the uncertainty inherent in predictions is embraced, allowing for more robust decision-making in many scenarios. There are numerous techniques and models in this category, including Bayesian networks [4, Ch. 8.1.], Gaussian processes [31], Markov and Hidden Markov Models [5, Ch. 17], and Markov random fields [5, Ch. 19], among others. Each of these models has its own strengths and applications, depending on the nature of the data and the problem at hand. One model class is particularly useful in some hybrid modeling scenarios – Gaussian processes. For this reason, they are introduced next.

2.3.2.2 Gaussian processes Gaussian processes (GPs) define a distribution over functions. They provide a principled, non-parametric methodology to infer underlying patterns in data [31]. A Gaussian process is defined by its mean function $m(x)$ and its covariance or kernel function $k(x,x')$. At a high level, the mean function describes the expected value of the process, and the kernel function dictates how data points influence each other based on their separation in the input space.

Formally, a Gaussian process can be represented as:

$$u(x) \sim \mathrm{GP}\big(m(x), k\big(x, x'\big)\big), \tag{10}$$

where $u(x)$ is the output of the GP for input x, $m(x)$ is the mean function, and $k(x, x')$ is the kernel function.

Since GPs provide a distribution over functions, they can capture an infinite number of possible explanations for the observed data. Any finite set of these observations can be viewed as being drawn from some multivariate Gaussian distribution defined by the mean and kernel functions. This is particularly powerful as it not only provides a prediction for unseen data but also an associated uncertainty, which can be crucial for decision-making in uncertain environments.

Kernel functions play an integral role in shaping the GP, with the choice of kernel determining the nature of functions the GP can represent. For instance, the Radial Basis Function (RBF) kernel assumes that points closer in input space are more correlated, leading to smooth function approximations. On the other hand, periodic kernels can capture cyclical patterns in the data.

Training a GP typically involves maximizing the likelihood of the observed data under the GP prior, leading to the optimization of kernel hyperparameters. Once trained, predictions with GPs involve conditioning the GP on the observed data to infer values (and uncertainties) at unseen input points.

However, one should note that while GPs offer many advantages, including providing uncertainty estimates and flexibility in modeling, they can become computationally expensive with large data sets. But recent advancements and approximations, like inducing points or sparse GPs [32–34], allow for more scalable implementations. If GPs are combined with universal kernels, such as the RBF kernel, their data hunger rises very quickly with the number of input features, an effect also known as the "curse of dimensionality". Here, it often helps to build customized kernels that take properties of the data into account, e.g. convolutional kernels for images [35] or kernels tailored to linear ODE and PDE systems [36, 37].

Altogether, Gaussian Processes are a versatile tool for machine learning and allow hybrid modeling at scale [28, 38, 39].

2.3.2.3 Neural networks Neural networks [6] are computational models consisting of interconnected nodes, or "neurons" (this terminology is borrowed from how the brain processes information), organized into layers: input, hidden, and output layers. The connections between neurons has an associated weight, which is adjusted during training to minimize the difference between the predicted and actual output. Each layer of a neural network can be represented as $\sigma(Wx + b)$, where W is a matrix of weights, x is the input vector from the previous layer, b is the bias vector, and σ represents an activation function, such as the sigmoid or ReLU (Rectified Linear Unit) [40], which is applied element-wise.

The power of neural networks lies in their capacity to approximate complex, non-linear functions. By stacking multiple layers and using non-linear activation functions, neural networks can capture intricate patterns and relationships in data. The training process involves iteratively adjusting the weights using optimization algorithms like gradient descent to reduce the error between the network's predictions and the ground truth.

Deep learning, a sub-field of machine learning, refers to neural networks with many layers, enabling the capture of even more complex representations. For instance, convolutional neural networks (CNNs) [6, Ch. 9] are adept at processing image data, while recurrent neural networks (RNNs) [6, Ch. 10] excel in handling sequential data.

However, while neural networks have achieved remarkable success in various applications, they come with challenges. For example, neural networks require an amount of data that is appropriate for the size of the network to avoid a phenomenon called overfitting. When the network becomes large and has many parameters but is trained on too little data, it can use its modeling capacity to model irrelevant details including noise which leads to overfitting meaning that the predictions will be close to perfect on the training data but will not work well for new test cases. Since model behavior is determined by the training data, out-of-sample and out-of-distribution generalization cannot be assumed. In addition, the "black-box" nature of neural networks usually limits the interpretability of the model and its predictions. Finally, hyperparameter tuning is another area of concern, requiring extensive experimentation to find the optimal settings for parameters such as the learning rate, batch size, and network depth, which can be both time-consuming and resource-intensive.

2.3.2.4 Regularization of machine learning methods Regularization techniques serve as foundational tools in machine learning, designed to prevent models from overfitting to their training data. By introducing a penalty to the model's complexity, regularization ensures that models remain generalizable to unseen data [4]. L_1 (Lasso) and L_2 (Ridge) regularization, which penalize the magnitude of model parameters, can be viewed as implicit modeling methods. They don't dictate the model's structure directly but influence it by penalizing certain parameter configurations. In neural networks, techniques like dropout, which randomly deactivates certain neurons during training, aid in enhancing generalization. Other methods such as early stopping and batch normalization, which normalizes neuron activations, further contribute to model robustness. While regularization provides a shield against overfitting, it introduces the challenge of selecting the right regularization strength, necessitating meticulous tuning and validation.

2.4 Explicit versus implicit models

In Sect. 2.1 we have introduced computational models, and so far avoided the distinction between explicit models, which directly provide computational representations like Eq. (1), and implicit models, which on their own are not enough to obtain a computational model. While an explicit model prescribes a direct mapping from input x to output y implicit models often require a solver or an optimization procedure to result in a computational model akin to Eq. (1). Regularization is a fitting example of this distinction. While it introduces constraints or penalties to the learning process, it doesn't directly specify the functional form of the model. Instead, the model emerges as a result of an optimization process that balances fitting the data with the imposed regularization constraints.

Similarly, differential equations provide the dynamics or laws governing a system but don't directly offer a computational model for predicting states. Only when combined with a solver, often numerical, do they yield a method to predict the state at subsequent time points. Partial differential equations (PDEs), such as Maxwell's equations, also epitomize this concept. While they describe the fundamental relationships between electric

and magnetic fields, a computational model that predicts field values at specific spatial and temporal points necessitates the application of a solver. The allure of implicit models lies in their ability to capture complex behaviors and constraints. However, they also demand a deeper understanding and careful selection of solvers or optimization techniques to ensure accurate and meaningful predictions.

2.5 Model composition

A computational model, as defined in Sect. 2.1 can itself be a composition of multiple sub-models. The generic function u that we have used so far can be composed of other functions representing the sub-models in various ways. The sub-models can be implicit or explicit and can be data-driven or first-principles based. The contribution of this paper is to present different design patterns for composing data-driven and first-principle based models.

2.5.1 Model composition in machine learning

An example of model composition is deep kernel learning [41]. In deep kernel learning, the kernel function of a GP is parameterized using a deep neural network. This means that instead of using a traditional kernel function like the RBF or Matérn kernel, the kernel is defined by the outputs of a neural network. Formally, given two input vectors x and x', the kernel function can be represented as $k_{\theta_k}(f_{\theta_f}(x), f_{\theta_f}(x'))$, where f_{θ_f} is the neural network with parameters θ_f, and k_{θ_k} is a base kernel with parameters θ_k.

This composition allows the model to learn intricate patterns and relationships in the data that might not be captured by a standard GP kernel. By mapping the input data into a new representation space using the neural network, the kernel can operate on features that are potentially more informative and better suited to the problem at hand.

Another illustrative example of model composition is the concept of model stacking or stacked generalization [42]. Here, individual models, often referred to as base learners, make predictions which are then used as input features for another model, typically called the meta-learner or the stacking model. The meta-learner then makes the final prediction. This composition technique aims to combine the strengths of multiple models, thereby improving generalization performance.

A different perspective on model composition can be found in ensemble methods like bagging [43] and boosting [44]. In bagging, multiple models are trained on different subsets of the data and then averaged (for regression) or voted upon (for classification) to make predictions. Boosting, on the other hand, iteratively trains models by giving more weight to instances that previous models got wrong, aiming to correct mistakes made by earlier learners.

2.5.2 Model composition of models based on first principles

Another example of model composition can be found in classical electrodynamics. An electromagnetic field is defined as a four-tuple of space- and time-dependent vector fields $(\vec{E}, \vec{D}, \vec{H}, \vec{B})$, the *electric field* $\vec{E}$, the *electric displacement* $\vec{D}$, the *magnetic field* $\vec{H}$, and the *magnetic flux density* $\vec{B}$. Electromagnetic fields are governed by Maxwell's equations, a set of four PDEs. Two of the equations are dynamic equations, since they contain time

derivatives. We collect them in a sub-model U_1,

$$\frac{\partial}{\partial t}\begin{pmatrix}\vec{D}\\ \vec{B}\end{pmatrix} = \begin{pmatrix}0 & +\operatorname{curl}\\ -\operatorname{curl} & 0\end{pmatrix}\begin{pmatrix}\vec{E}\\ \vec{H}\end{pmatrix} - \begin{pmatrix}\vec{j}\\ 0\end{pmatrix}, \tag{11}$$

with the *electric current density* $\vec{j}$. The first equation in (11) is Ampère's law, the second Faraday's law, respectively. The remaining two equations have the form of PDE constraints. We collect them in the sub-model U_2,

$$\begin{pmatrix}0\\ 0\end{pmatrix} = \begin{pmatrix}\operatorname{div} & 0\\ 0 & \operatorname{div}\end{pmatrix}\begin{pmatrix}\vec{D}\\ \vec{B}\end{pmatrix} - \begin{pmatrix}\rho\\ 0\end{pmatrix}, \tag{12}$$

with the *electric charge density* ρ. These are the electric and magnetic Gauss' laws, respectively. Maxwell's equations (U_1, U_2) need to be complemented by constitutive relations that encode the material properties. For simple media at rest, the additional sub-model U_3 takes the algebraic form

$$\begin{pmatrix}\vec{D}\\ \vec{B}\end{pmatrix} = \begin{pmatrix}\boldsymbol{\varepsilon} & 0\\ 0 & \boldsymbol{\mu}\end{pmatrix}\begin{pmatrix}\vec{E}\\ \vec{H}\end{pmatrix}, \tag{13}$$

with the dielectric tensor $\boldsymbol{\varepsilon}$ and the permeability tensor $\boldsymbol{\mu}$. All three sub-models can be written in implicit form $U_i(\vec{E}, \vec{D}, \vec{H}, \vec{B}) = 0$, $i = 1, 2, 3$, and aggregate to the composed model $U = (U_1, U_2, U_3)$, which yields a predictive model of electrodynamics.

3 Hybrid modeling design patterns

Hybrid modeling is diverse with applications ranging from molecular modeling in drug discovery [45], over various simulation tasks in climate science [46] or various engineering disciplines [47], to modeling sensor data for virtual sensing. Solutions for individual use cases are usually application-specific. New hybrid modeling challenges often seem so unique that interdisciplinary teams come together to develop a custom solution from scratch. While this leads to progress in individual disciplines, solutions are often not accessible to other application domains.

To make progress in hybrid modeling research, it is necessary to abstract recurring modeling challenges and to distill useful solution approaches that generalize across applications. The goal of this paper is to introduce hybrid modeling *design patterns* that formalize these solution approaches at an abstraction level beyond individual applications. We adopt the following definition of design pattern.

Definition 1 A hybrid modeling design pattern is a reusable blue-print for a building block of a general solution to recurring hybrid modeling challenges.

Per our definition, a design pattern should address *recurring* challenges beyond individual application domains. For this reason, the solution approach encoded in the design pattern should be *general*, meaning that application-specific aspects are abstracted away.

Further, the hybrid modeling design patterns are modular and solving a modeling challenge will typically involve the composition of multiple design patterns. Finally, a design pattern is a blue-print rather than an implementation; blue-prints are *reusable* and useful for developing a solution and guiding its implementation.

In this section, we discuss the motivation behind working at this level of abstraction and list properties of useful design patterns. We then introduce the block diagram notation we propose to communicate the design patterns. Finally, we provide some guidance on how the design patterns can be used for new hybrid modeling use cases as well as meta-level research.

3.1 The block diagram notation for hybrid modeling design patterns

We propose a simple block diagram notation for working with the hybrid modeling design patterns. The general question in recurring hybrid modeling challenges is typically how to best combine the available domain knowledge with the available data. The data is processed by a data-driven model, which we denote by $\mathcal{D}$, while the chosen first-principles-based model is denoted by $\mathcal{P}$. Both models $\mathcal{D}$ and $\mathcal{P}$ are computational blocks, which receive inputs and perform computations to produce an output. For example, a data-driven model component will receive observations as an input which it will process to either produce a prediction, a lower dimensional representation of the input, or another quantity that is needed for the modeling challenge at hand. The inputs to $\mathcal{P}$ will depend on the type of domain knowledge available. In the case of a differential equation for example, the inputs might consist of the initial conditions and the time interval over which the dynamics are to be integrated. The desired output could be the simulated dynamics, or the final state.

In the block diagram notation, a computational block (typically $\mathcal{P}$ or $\mathcal{D}$) is represented by a square. Directed arrows indicate the flow of information. For example, a directed arrow between two blocks indicates that the output (i.e. the result of the computation) of the first block, is used as one of the inputs to the second block. A computational block can have multiple incoming arrows, meaning that its inputs come from various sources, and it can have multiple outgoing arrows, meaning that its computational results are further processed in different ways.

In summary, a block diagram for describing a design pattern consists of rectangular boxes representing computational blocks and of directed arrows, which indicate the flow of inputs and outputs between the boxes. Actual examples of design patterns will be presented in Sect. 4.

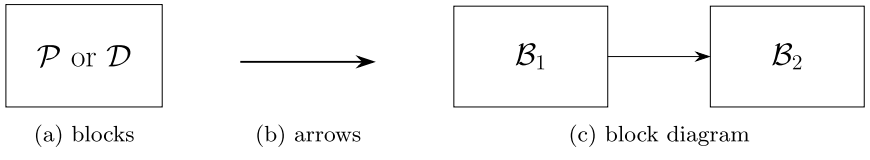


Figure 1 *A block diagram for a hybrid modeling design pattern* consists of computational blocks (Fig. 1a), indicating model components that involve computation, and arrows (Fig. 1b) indicating the flow of data and intermediate computational results. For example, the arrow in Fig. 1c indicates that the result of the computational block $\mathcal{B}_1$ is fed as an input into the computational block $\mathcal{B}_2$

3.2 Properties of useful design patterns

Before diving into the specific design patterns introduced in Sect. 4 and utilizing the block diagram notation to generate patterns that satisfy Definition 1, it is crucial to discuss the properties that make a design pattern useful. Some of these properties are essential and have already been explicitly stated in our definition of hybrid modeling design patterns.

Design pattern versus architecture We prefer the term "design pattern" over "architecture" because, in a specific model architecture, several design patterns might be combined or nested. Additionally, we emphasize that the design patterns were collected by analyzing actual applications. Since there is no comprehensive theory of hybrid modeling from which these patterns could be derived, our collection is not exhaustive and is intended to grow as new design patterns are developed or gain importance.

Abstract and general An essential step in creating design patterns is abstracting useful concepts that are applicable across various applications and formulating them in a way that makes them easily applicable in a general reusable context. A good design pattern is not a finished design, but rather a blueprint that can be adapted to specific problems.

Design patterns should be abstract and general rather than application-specific, allowing them to be applied across a wide range of problems. This flexibility enables researchers and practitioners to adapt and customize the design pattern for their specific needs, promoting innovation and problem-solving in diverse fields.

Broad applicability A useful design pattern should have the potential to address various challenges and applications, enabling researchers and practitioners to benefit from its adoption. By offering solutions that can be adapted to different contexts, a design pattern with broad applicability can contribute to the development and improvement of numerous models, fostering progress across multiple domains.

Modularity and composability Design patterns should be modular, allowing for easy integration with other patterns, and promoting composability for constructing more complex models. This property enables the combination of multiple design patterns, leading to the creation of more sophisticated and powerful hybrid models that can tackle complex challenges.

Tractability and ease of communication A good design pattern should be tractable, facilitating implementation, and easy to communicate, promoting understanding and collaboration among researchers and practitioners. Clear and understandable design patterns encourage adoption and facilitate the sharing of ideas, contributing to the overall growth and development of hybrid modeling methodologies.

Clear interface between physics-based and data-driven components An effective design pattern should provide a clear interface between the physics-based and data-driven components, enabling seamless integration and interaction between the two modeling paradigms. By defining how these two aspects interact, a design pattern can help create a cohesive and well-structured model that effectively leverages the strengths of both approaches.

4 Examples of design patterns

We now delve into the key design patterns for hybrid modeling. There will be two types of patterns, base patterns and composition patterns. The base patterns establish systematic approaches for combining a first-principles-based model $\mathcal{P}$ with a data-driven model $\mathcal{D}$, capitalizing on the strengths of both modeling techniques. In Sect. 4.1, each of the base design patterns is described in detail, elucidating the principles and methodologies underlying their application. Furthermore, we provide illustrative examples to enhance comprehension and demonstrate the practical utility of these design patterns in various scenarios. In Sect. 4.2, we present patterns for the composition of base patterns. These composition patterns facilitate building more elaborate hybrid modeling solutions for complex modeling tasks.

4.1 Base patterns for hybrid modeling

The base patterns are the basic building blocks for the development of hybrid modeling solutions. Each design pattern takes two computational models, typically a first-principles-based model $\mathcal{P}$ and a data-driven model $\mathcal{D}$ and combines their computation steps into a hybrid model. The order in which the computation is executed, and the flow of inputs and outputs between computational blocks will differ between the design patterns.

In the following sections, we present a total of four base patterns, with the first three having previously been introduced by von Stosch et al. [48] within the context of process systems engineering.

4.1.1 The delta model

The delta model serves as a fundamental design pattern in hybrid modeling, providing an effective method to combine the strengths of both first-principles-based and data-driven models. This design pattern is particularly useful when the first-principles-based model captures the primary underlying physical, chemical, or biological processes but may lack the precision or comprehensiveness required for specific applications. By introducing a data-driven component that accounts for discrepancies or unmodeled phenomena, the delta model can significantly enhance the accuracy and predictive capabilities of the overall hybrid model.

The delta model is formulated by additively combining a first-principles-based model $\mathcal{P}$ with a data-driven model $\mathcal{D}$, resulting in a hybrid model $\mathcal{H}$ as follows:

$$\mathcal{H}(x) = \mathcal{D}(x) + \mathcal{P}(x). \tag{14}$$

The block diagram is given in Fig. 2. In the equation, x represents the input variables, and $\mathcal{H}(x)$, $\mathcal{P}(x)$, and $\mathcal{D}(x)$ are the output predictions for the hybrid, first-principles-based,

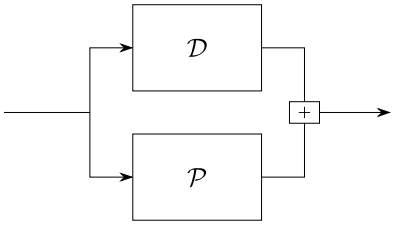


Figure 2 *The block diagram of the delta model* (Sect. 4.1.1). In this design pattern, the outputs of the data-driven computation $\mathcal{D}$ and the first-principles-based computation $\mathcal{P}$ are combined additively in a computational block denoted by "+"

and data-driven models, respectively. The first-principles-based model, $\mathcal{P}(x)$, encapsulates the primary knowledge of the underlying processes, while the data-driven model, $\mathcal{D}(x)$, is trained to capture the discrepancies between $\mathcal{P}(x)$ and the observed data. The data-driven component, therefore, accounts for the unmodeled or inaccurately modeled phenomena, refining the overall predictions made by the hybrid model.

Typical use cases The delta model is applicable in a variety of scenarios, including but not limited to:

- Thompson and Kramer [49] suggest compensating for the inaccuracies of first principle based equations, such as mass and component balances by building a hybrid model which additively combines these simple process models with a neural network. For a survey of more recent approaches we refer the reader to Zendehboudi et al. [50].
- Ground water modeling in geoscience: Xu and Valocchi [51] showcase that various data-driven models are effective at correcting the bias of physics-based ground flow models and can in addition produce well calibrated error bars.
- Computational fluid dynamics: Reynold-averaged Navier Stokes (RANS) equation solvers are an important computational tool for modeling turbulent flows. Unfortunately, RANS predictions are often inaccurate due to large discrepancies in the predicted Reynolds stress. Wang et al. [52] propose to mitigate these discrepancies with a data-driven correction term.
- Dynamics modeling: Levine and Stuart [53] present a unified framework for learning the modeling error in dynamical systems, when $\mathcal{P}$ is described by differential equations.

Example To study the delta model in action, we consider data from an accelerometer. The long-term effects can be described by a harmonic oscillator with non-linear dampling, while the short-term effects lack a physical interpretation. We will study the delta model in comparison to just its physical component $\mathcal{P}$ or the data-driven component $\mathcal{D}$. We assume, that the underlying dynamics of the system resemble the Van der Pol equation (Eq. (4)) and that the short-time behavior can be simulated by a Gaussian process (GP).

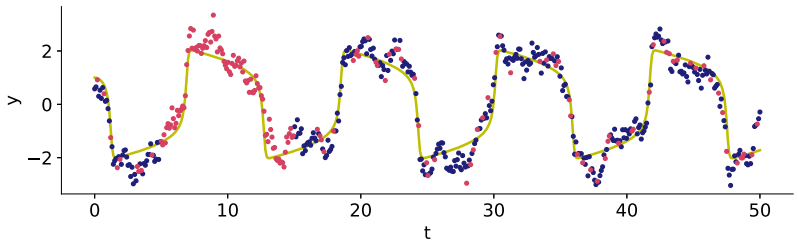
We generate data according to the model

$$y(t) = u_{\text{vdp}}(t) + u_{\text{loc}}(t) + \epsilon, \tag{15}$$

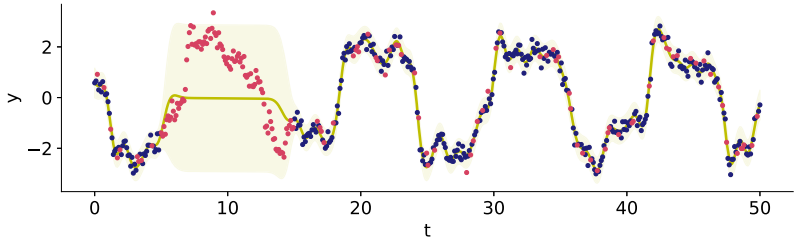
where $u_{\text{vdp}}(t)$ are the predictions obtained from the Van der Pol equation, $u_{\text{loc}}(t) \sim \text{GP}(0, k(t, t'))$ are simulated local effects according to a GP with squared exponential kernel with variance 0.2 and length scale 0.5 and $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ is Gaussian noise with variance $\sigma_n^2 = 0.05$.

To simulate the Van der Pol equation (Eq. (4)), we define the differential $f_{\text{ODE}} : \mathbb{R}^2 \to \mathbb{R}^2 : (s_t, v_t) \to (v_t, -s_t + \mu v_t(1 - s_t^2))$ in the state-space $h_t = (s_t, v_t) = (u_t, \frac{du_t}{d_t})$, where for ease of readability, we denote a function evaluated at time point t with the subindex t, e.g. $u_t \equiv u(t)$. We use a order 5(4) Runge-Kutta method to simulate $\frac{dh_t}{dt} = f_{\text{ODE}}(h_t; \mu)$ over the time interval $[0, 50]$ (at a resolution of 0.1 units) with $\mu = 5$, and initial state $h_0 = (1, 0)$.

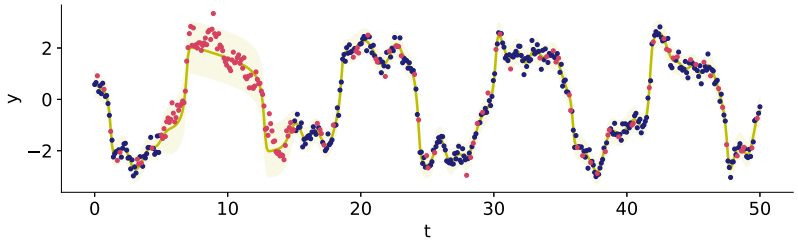
The generated time series data $\mathcal{D} = (t_k, y_k)_{k=1,\ldots,K}$, where y_k is the measured dynamic response at time t_k is depicted in Fig. 3, with training data denoted by blue points and test data denoted by red points. It can be seen that the generated data follows mostly the Van



(a) Van der Pol oscillator



(b) Gaussian Process



(c) Hybrid Model (Delta Pattern)

Figure 3 *Evaluation of different methods on a toy accelerometer set-up.* From top to bottom: Predictions from a (a) Van der Pol oscillator ($\mathcal{P}(t)$), (b) Gaussian Process ($\mathcal{D}(t)$) and (c) hybrid model combining both approaches according to the delta model ($\mathcal{H}(t) = \mathcal{P}(t) + \mathcal{D}(t)$). Training data is shown in blue, test data in red. The predictions are shown in yellow. The yellow shaded areas in Figure (b) and (c) depict the 95% confidence interval of the predictions. We can observe that the Van der Pol oscillator cannot capture the local effects of the data, while the Gaussian Process falls short when training data is scarce. The hybrid model combines the best of both worlds and performs well under all data scenarios

der Pol equation, which covers the majority of the underlying physical processes, but does not fully account for certain localized phenomena or short-term dynamics. To make the modeling task more challenging, we further assume that the measurement system had a black-out between 5 and 15 time units during which no training data is available.

The results in the figure provide a qualitative comparison of a pure first principles-based modeling approach based on Eq. (4), fitting a data-based approach (Eq. (10)), and a hybrid model using the delta approach.

Figure 3a shows the dynamic response according to the Van der Pol equation. While this model accurately captures the long-term behavior of the system, it falls short in capturing the finer details and short-term effects.

The GP predictions are shown in Fig. 3b. When abundant training data is available, the Gaussian Process performs well. However, if training data is scarce (between 5 and 15 time

units), the predictions fall back to the prior (which is zero) and are accompanied by high uncertainties.

Finally, we combine the Van der Pol oscillator with the Gaussian Process. The data-driven model learns the discrepancies between the first-principles-based model's predictions and the observed data, effectively accounting for unmodeled or inaccurately modeled phenomena. Results are depicted in Fig. 3c demonstrating that the hybrid model combines the best of both worlds: when training data is available, the Gaussian Process improves the predictions compared to the physics-based model significantly, capturing effects not considered in the Van der Pol equation. When training data is limited, the physics-based model takes over, as the Gaussian Process predictions revert to the prior.

Employing the delta model combines the first-principles-based and data-driven components, resulting in an improved hybrid model. Our results confirm that this model provides more accurate and reliable predictions by accounting for both the strengths and the limitations of the individual models in different data scenarios.

Discussion The delta model offers several compelling advantages that underscore its utility in hybrid modeling. One of its primary strengths is the facilitation of fast prototyping. With the availability of a first-principles-based model $\mathcal{P}$, researchers and practitioners can swiftly initiate their modeling efforts. As more data becomes available or as the need for enhanced precision arises, the data-driven component $\mathcal{D}$ can be incrementally introduced, refining the model without necessitating a complete overhaul.

Moreover, the delta model inherently promotes higher accuracy and robustness. While the physical model $\mathcal{P}$ provides a foundational understanding, it might occasionally fall short due to assumption mismatches or its inability to encapsulate the stochasticity inherent in many real-world processes. For instance, $\mathcal{P}$ might be predicated on idealized assumptions, such as negligible noise levels or presumed linearity, which might not hold true in practical scenarios. The data-driven component $\mathcal{D}$ serves as a corrective mechanism in such instances, adeptly learning to account for complex non-linearities, stochastic effects, and other intricate real-world phenomena that the physical model might overlook.

Another salient advantage of the delta model is its data efficiency. Learning the deviations or discrepancies from an existing model $\mathcal{P}$ is often more data-efficient than attempting to learn the entire function from scratch solely through $\mathcal{D}$. This efficiency is particularly pronounced when training data is sparse. By incorporating the physical model, the delta model introduces a beneficial inductive bias, ensuring that even in low-data regimes, plausible estimates can be generated.

Lastly, the delta model's design inherently supports specialization. In many scenarios, it might be infeasible to obtain training data that spans the entirety of the input domain, perhaps due to safety concerns, prohibitive measurement costs, or other constraints. The delta model elegantly addresses this challenge. For test points that lie outside the domain covered by the training data, the physics-based model $\mathcal{P}$ takes precedence, leveraging its capability to extrapolate reliably. Conversely, for inputs that are well-represented in the training data, the data-driven model $\mathcal{D}$ offers its specialized insights, ensuring predictions that are both accurate and nuanced.

The advantages described above, make the delta model a popular design pattern for hybrid modeling. However, it also has its limitations. Due to the additive nature of the pattern, it has limited modeling flexibility. Specifically, it does not explicitly model higher-order interactions between the physics-based model and the data-driven component.

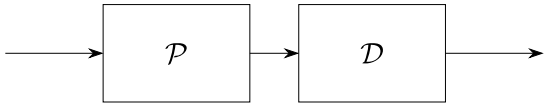


Figure 4 *In physics-based preprocessing* (Sect. 4.1.2), the inputs to the data-driven model $\mathcal{D}$ are first transformed based on first-principles in the computational block $\mathcal{P}$

4.1.2 Physics-based preprocessing

Physics-based preprocessing is another crucial design pattern in hybrid modeling that leverages domain knowledge to enhance the performance of data-driven models. By incorporating transformations derived from physical laws or other domain-specific knowledge, this design pattern preprocesses the input data before feeding it into a data-driven model. The preprocessing step can introduce useful inductive biases, reduce the dimensionality of the data, and improve the overall efficiency and interpretability of the resulting model.

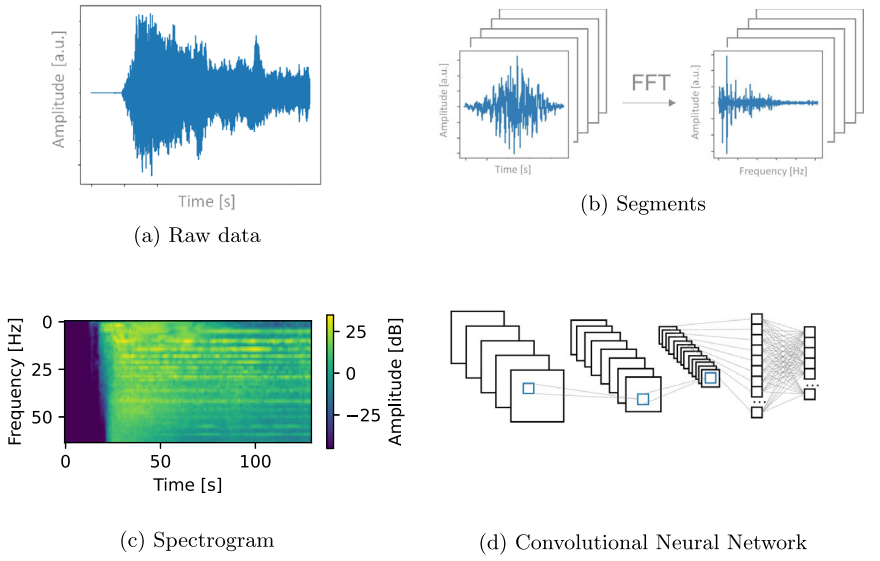
In the physics-based preprocessing design pattern, a transformation model $\mathcal{P}$ is applied to the input variables x before they are fed into a data-driven model $\mathcal{D}$. The transformation function incorporates domain knowledge, such as physical laws or constraints, to preprocess the data. The output prediction of the hybrid model $\mathcal{H}(x)$ can be expressed as:

$$\mathcal{H}(x) = \mathcal{D}\big(\mathcal{P}(x)\big). \tag{16}$$

Here, $\mathcal{P}(x)$ represents the preprocessed input variables, and $\mathcal{H}(x) = \mathcal{D}(\mathcal{P}(x))$ are the output predictions for the hybrid and data-driven models, respectively. The transformation function, $\mathcal{P}(x)$, is designed based on domain knowledge to enhance the data's representation or to simplify the data-driven model's task, leading to improved performance and interpretability. The block diagram for physics-based preprocessing is in Fig. 4.

Typical use cases Physics-based preprocessing is applicable in various scenarios, including:

- Time-series processing with spectrograms: Time-series data is often preprocessed using short-time Fourier transform (STFT) turning the 1-D time domain signal into a 2-D time-frequency representation. Deep learning based methods are more effective in the time-frequency domain for many different applications such as time-series anomaly detection [54], sound classification [55], heart disease diagnosis on electrocardiograms [56] and object classification on radar sensors [57].
- Fault-detection in mechanical engineering: Rolling-element bearings are an integral component of many machines and bearing fault detection is an important task in mechanical engineering [58]. There is a long history of analyzing vibration patterns and acoustic signals for bearing fault detection. For example, peaks in certain spectra are known to be predictive of imminent failure. Sadoughi and Hu [59] exploit this know-how for physics-based preprocessing of vibration and acoustic data which is then fed into a convolutional neural network (CNN) for bearing fault detection and localization.
- Demand forecasting: Accurate electricity demand forecasting is an important factor for efficient planning in industry, healthcare, and urban planning. Bedi and Toshniwal [60] combine empirical mode decomposition (EMD) with deep learning. In EMD, the



(a) Raw data (b) Segments

(c) Spectrogram (d) Convolutional Neural Network

Figure 5 *Audio classification with spectrograms as an example of physics-based preprocessing.* The raw audio data is segmented into overlapping windows which are mapped to Mel spectograms. This representation of the data in the time-frequency domain is then processed in a data-driven manner

electricity load signals are first decomposed into signals with different time scales, chosen based on domain-knowledge, as well as a residual term. Each of the signal components is then used to train a separate long short-term memory (LSTM) network [61]. These LSTM networks can then be combined to forecast electricity demand.

Example Consider the example of sound classification which is used in many different application fields such as music categorization based on genres, user identification based on voice or bird classification based on audio recordings.

The audio data (see Fig. 5a) undergoes an initial transformation into a spectrogram using physics-based preprocessing denoted as $\mathcal{P}(x)$. This involves segmenting the audio into overlapping windows of a fixed size (refer to Fig. 5b). For each window, a Fourier transform is applied, resulting in a 2-D representation in the time-frequency domain. Subsequently, each snapshot can be plotted as a Mel spectrogram [62], where time is represented on the x-axis, frequency on the y-axis, and the amplitude is depicted using colors (see Fig. 5c).

By obtaining an image representation of the data, we can leverage standard image classification models, denoted as $\mathcal{D}(\mathcal{P}(x))$, such as convolutional neural networks (see Fig. 5d). These architectures are designed to respect image structures, incorporating features like translation equivariance and locality. This design choice not only reduces memory requirements but also enhances the model's ability to generalize effectively.

Discussion Physics-based preprocessing in hybrid modeling can improve data efficiency. Using the transformation model $\mathcal{P}$ can allow the model to compute features directly, reducing the learning burden on the data-driven model $\mathcal{D}$. Especially when $\mathcal{P}$ is a type of dimensionality reduction, the lower-dimensional presentation has often a lower complexity since noise is removed or redudant information is discarded. This makes it simpler for the learning algorithm to extract meaningful patterns leading to a better trade-off between performance and training dataset size.

Note however, that in cases where $\mathcal{P}$ does not capture all relevant raw feature information, a purely data-driven model might perform better in data-rich scenarios. This is because $\mathcal{D}$ can identify features that outperform human-designed ones, as seen in deep learning methods applied to speech recognition and computer vision.

Similarly, the design pattern also offers resource efficiency. Using pre-computed features in $\mathcal{P}$ can simplify the data-based model $\mathcal{D}$, potentially removing the need for complex structures like deep neural networks. With features from $\mathcal{P}$, simpler algorithms might be adequate for $\mathcal{D}$.

Finally, the pattern can increase robustness by avoiding irrelevant feature learning in $\mathcal{D}$, that could lead to overfitting or offer an opportunity for adversarial attacks, and it can increase the explainability of the model, by providing a physical interpretation of the features.

4.1.3 Feature learning

The feature learning design pattern combines data-driven feature learning with downstream physics-based processing. This design pattern comes into play when the first principle based model $\mathcal{P}$, for example a controller or a PDE, has some input features that are difficult to measure directly or are difficult to compute precisely from first principles.

In the feature learning design pattern, a data-driven model $\mathcal{D}$ is employed to estimate unmeasurable input variables v based on measurable input variables x, $v = \mathcal{D}(x)$. These estimated variables are then used as an input for a first-principles-based model $\mathcal{P}$ that performs downstream physics-based computations. The output prediction of the hybrid model $\mathcal{H}$ can be expressed as:

$$\mathcal{H}(x) = \mathcal{P}\big(x, \mathcal{D}(x)\big) \tag{17}$$

Here, x represents the measurable input variables, and $v = \mathcal{D}(x)$ are the estimated unmeasurable input variables produced by the data-driven model. $\mathcal{H}(x)$ and $\mathcal{P}(x, \mathcal{D}(x))$ denote the output predictions for the hybrid and first-principles-based models, respectively. The data-driven model, $\mathcal{D}(x)$, is trained to estimate the unmeasurable input variables v using available data, which is then utilized by the first-principles-based model $\mathcal{P}(x, \mathcal{D}(x))$ for its computations. The block diagram for feature learning is given in Fig. 6. In some applications, $\mathcal{D}(x)$ will be pre-trained and then combined with $\mathcal{P}(x, \mathcal{D}(x))$ for hybrid predictions. In other applications, the feature extractor is learned by directly predicting the outputs of the combined hybrid model $\mathcal{H}(x) = \mathcal{P}(x, \mathcal{D}(x))$. This is called end-to-end training.

When $\mathcal{P}$ is a physical model, the learned input variables will often have a physical interpretation. The feature learning design pattern is closely related to the design pattern of physical constraints, which will be discussed in Sect. 4.1.4. Since $\mathcal{P}$ is used to process

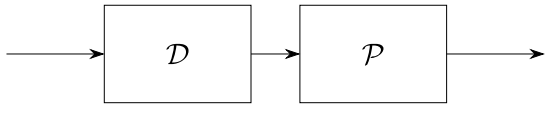


Figure 6 The block diagram of feature learning (Sect. 4.1.3). In this design pattern, some of the features of $\mathcal{P}$ are computed in a data-driven manner by $\mathcal{D}$

the predictions of $\mathcal{D}$ we can see $\mathcal{P}$ as transforming the outputs of $\mathcal{D}$ in a meaningful way, e.g. to fulfill physical constraints.

One nuance to consider for the feature learning design pattern is whether $\mathcal{P}$ is only used during training, e.g. to provide a loss or regularization term to guide the data-driven model to make physically plausible predictions, or whether $\mathcal{P}$ is also used to make predictions.

Typical use cases The feature learning design pattern can be applied in various scenarios, including:

- Electromagnetic field simulations: The optimization of photonic devices requires calculating electromagnetic fields. Chen et al. [63] propose a hybrid approach, where a deep learning model predicts the magnetic near-field distribution. A discrete version of Ampère's law is then used to calculate the electric from the predicted magnetic near field. Eventually, the far field of the outgoing plane wave is computed from the electric near field, by using a near-to-far-field transformation.
- Solving PDEs: Deep learning methods for approximating PDE solutions also exemplify the feature learning design pattern. In these approaches deep learning techniques are employed to learn the differential operators and nonlinear responses of the underlying (parametric) PDE [64–70]. This results in models that are capable of capturing complex dynamics while adhering to the physical principles governing the system.
- Virtual sensors: Some first-principle-based systems, for example, controllers, require input modalities that are impractical or impossible to measure. For example, a controller for electrical machine torque might require an estimate of rotor temperature [71]. Virtual sensors are data-driven replacements that predict the input modalities that cannot be measured directly but are required for downstream physics-based computations [72].

Discussion The feature learning design pattern offers several distinct advantages in hybrid modeling. Firstly, it addresses the challenge of unmeasurable or imprecisely computed input features. By employing a data-driven model $\mathcal{D}$ to estimate these features, the pattern effectively bridges the gap between available data and the requirements of a first-principles-based model $\mathcal{P}$. This not only enhances the accuracy of the hybrid model but also broadens its applicability to scenarios where direct measurements or computations are infeasible.

This enables virtual sensing, where a predictive model replaces an expensive sensor or enables applications where a required input cannot be measured. In control engineering, this concept is widespread and known as state observer or state estimate.

One limitation of this design pattern is that end-to-end optimization usually requires $\mathcal{P}$ to be differentiable. Only then can $\mathcal{D}$ and $\mathcal{P}$ be optimized jointly with gradient-based methods. Applying feature learning to non-differentiable $\mathcal{P}$ requires iterative optimization schemes or simulation-based inference.

When $\mathcal{P}$ represents a physical model, the learned input variable often carries a meaningful physical interpretation, adding a layer of interpretability to the hybrid model. Furthermore, the integration of $\mathcal{P}$ ensures that the outputs of $\mathcal{D}$ are transformed in a manner that aligns with physical constraints or other domain-specific knowledge (this design pattern is described next). This not only enhances the reliability of the model but also ensures that its predictions adhere to known principles, such as the softmax function ensuring outputs that can be interpreted as probabilities. Lastly, the versatility of the pattern allows for

$\mathcal{P}$ to be employed both during training, as a guiding mechanism, and during prediction, ensuring that the model remains grounded in first principles throughout its life cycle.

4.1.4 Physical constraints

Physical constraints is a hybrid modeling design pattern that incorporates domain knowledge, such as conservation laws, priors, invariances, or statistical independence, to inform the architecture of a data-driven model. The constraints can either affect the structure of the model, the parameters of the model, or its computational results, including both intermediate or final outputs.

In the design pattern of physical constraints, domain knowledge can be tightly interwoven with the structure or parametrization of a data-driven model $\mathcal{D}$. The resulting hybrid model $\mathcal{H}$ is formed by incorporating these constraints into the data-driven model, which in its most general form we denote by

$$\mathcal{H}(x) = \mathcal{D}_{\mathcal{P}}(x). \tag{18}$$

We choose the notation $\mathcal{D}_{\mathcal{P}}$ to indicate that the data-driven model $\mathcal{D}$ is informed by physical constraints $\mathcal{P}$. The design pattern of physical constraints allows the data-driven model to adhere to the underlying physical principles while still leveraging the benefits of data-driven modeling techniques.

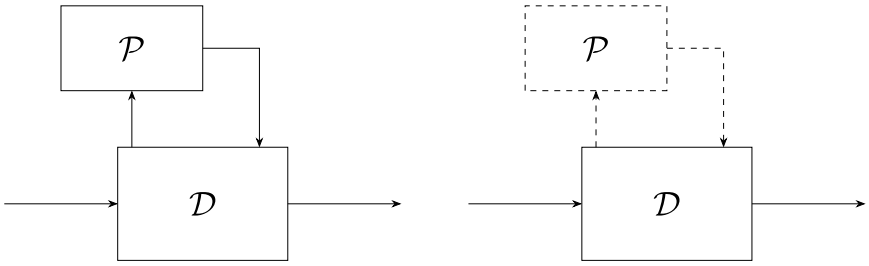
In most of the examples we consider below, the physical constraints are incorporated into model *predictions* by first doing the data-driven computations (e.g. feature extraction with the forward pass of a neural network) and then executing some computational steps derived from first-principles. In this case, the hybrid model can be written as in Eq. (17). A discussion of how physical constraints relate to feature learning can be found in the end of this Section. There are many flavors for building hybrid models where a data-driven block $\mathcal{D}$ is followed by computation $\mathcal{P}$ derived from first-principles. We roughly distinguish three directions: Hard constraints (e.g., [13]), soft constraints (e.g., [73]), and feature learning which has already been described. In hybrid models with hard constraints, the constraints are implemented in a way such that the predictions of the hybrid model cannot possibly violate the constraints. In contrast, soft constraints, which are often implemented in terms of physics-informed losses for training only approximately guide the predictions to lie within the desired ranges. Feature learning is closely related to the design pattern of hard constraints but has a different motivation. It comes into play, when a model $\mathcal{P}$ is missing some input dimensions that cannot be measured and have to be estimated with a data-driven model instead.

4.1.4.1 Hard constraints

The block diagram for hard constraints is depicted in Fig. 7a.

Typical use cases Hard physical constraints can be applied in various scenarios, such as:

- Multi-class classification: In multi-class classification, a neural network or another data-driven model $\mathcal{D}$ is tasked to produce probabilities over the possible class labels. To ensure that the outputs are in the right range (probabilities are between 0 and 1) and are properly normalized, the last layer is fed through a softmax activation function [74]. This constraint cannot be violated and ensures that the outputs can be interpreted as probabilities. In this example, the constraints affect the output of the



(a) Hard constraints (Sec. 4.1.4.1). (b) Soft constraints (Sec. 4.1.4.2).

Figure 7 In the physical constraints design pattern, the computation in the data-driven block $\mathcal{D}$ is informed by the domain knowledge in $\mathcal{P}$. The constraints can affect the architecture of $\mathcal{D}$, its parameters, or computational results both at intermediate levels and at the output. We distinguish between hard constraints (Fig. 7a) which take effect both during the model fitting stage and inference time and soft constraints (Fig. 7b) which are typically only applied at training time. After training soft constraints are implicitly encoded in $\mathcal{D}$, but no longer used explicitly. For this reason, we denote the soft constraints in a dashed manner

model and are part of the model architecture meaning that they take effect both during training and at test time. Also, the softmax implements a *hard constraint*; since it is part of the model architecture final predictions cannot violate the desired constraint.

- Classical mechanics: Hamiltonian neural networks [75, 76] and Lagrangian neural networks [77] are another excellent example of this design pattern. In these networks, the model architecture is structured to ensure that the dynamics adhere to conservation laws, such as energy conservation, leading to more accurate and physically meaningful predictions. When modeling the motion of a pendulum, for example, Greydanus et al. [75] use a neural network to directly predict the Hamiltonian of the system. Classical mechanics then determines how to predict the system dynamics, based on the predicted Hamiltonian. Thanks to the Hamiltonian formulation, the structure of the model guarantees that the predicted dynamics conserves energy.
- Neural network-based PDE solvers can be modified to achieve exact satisfaction of boundary conditions, by introduction of length factors [78] or geometry aware trial functions [79].
- Climate modeling: Beucler et al. [13] propose two ways to incorporate linear conservation laws into a neural network for emulating a physical model: By constraining the loss function, or by constraining the architecture itself. Incorporating physical constraints through a loss function is different than modifying model structure: The loss will only guide model outputs to be physically plausible during training. At test time, regularization terms are dropped and while the model might have learned to obey the physical constraints, there are no guarantees that the outputs will be correct. Incorporating physics-based loss terms is therefore an example of *soft constraints,* which are discussed next.

4.1.4.2 Soft constraints: surrogates and physics-informed losses We have discussed hard constraints, where physical principles are encoded directly into the model structure. An alternative approach for incorporating physical constraints is based on *soft constraints*. Here a data-driven model is guided during training to mimic physically plausible behaviour. At

inference time, the constraints are usually no longer used explicitly, which is why we use dotted lines to denote soft constraints in Fig. 7b. A soft constraint is typically achieved by training a surrogate model, i.e. defining a set of training inputs $\mathcal{X}$ and using training pairs $\{x, \mathcal{P}(x) | x \in \mathcal{X}\}$ for training a data-driven model, usually a neural network, to emulate the desired behavior. After training, we will have achieved $\mathcal{D}(x) \approx \mathcal{P}(x)$ for all $x \in \mathcal{X}$.[1] A related approach for incorporating soft physical constraints is based on physics-based losses. Here the loss function used to train $\mathcal{D}$ will have some term, also called regularization terms, that will encourage $\mathcal{D}$ to make physically plausible predictions. These regularization terms can either affect intermediate computation or the final output of the model. In the latter case, the relationship to surrogate modeling becomes clear, as the regularization term will encourage $\mathcal{D}(x) \approx \mathcal{P}(x)$ for all $x \in \mathcal{X}$. For the design pattern of soft physical constraints, the influence of the physics based model is only explicit during the training phase of model development. At deployment time, the model structure is indistinguishable from a purely data-driven approach. The physical constraints are "implicitly" encoded in the parameters of the model.

Typical use cases Soft physical constraints can be applied in various scenarios, such as:

- In [73], the authors want to train neural networks to help find solutions of PDEs. For this, they suggest collecting data, where PDEs are solved using the finite element method (FEM). Using this FEM data, the authors train surrogate models that can predict solutions directly. Physical constraints, such as knowledge about the form of the PDE or its boundary values, are incorporated during training via regularization terms. Since high-fidelity solutions are more accurate but more costly to obtain, the authors propose a *multi-fidelity* approach. They train a cheaper low-fidelity surrogate model and a more expensive high-fidelity surrogate model, as well as a *difference*-NN that can be thought of as a correction term for obtaining a high-fidelity solution from the lower-fidelity one. In this manner, the authors also exploit the delta-model design pattern, in addition to physical constraints.
- Solving PDEs: Deep learning methods for approximating PDE solutions [64, 65] also exemplify the physical constraints design pattern. In these approaches, the model is structured as a PDE, with deep learning techniques employed to learn the differential operators and nonlinear responses of the underlying PDE. This results in models that are capable of capturing complex dynamics while adhering to the physical principles governing the system. Physics-Informed Neural Networks (PINNs) [80] demonstrate another application of the physical constraints design pattern. In PINNs, the state of the PDE is parameterized by a neural network, while the structure of the differential operator depends on the specific application, giving rise to the resulting hybrid model. The constraint is included in the loss function. A specialized case of this design pattern is developed by De Bézenac et al. [81] for advection-diffusion PDEs, which are used for sea surface temperature prediction. A similar approach can be found in Chen et al. [63], which was also discussed in the context of the feature learning design pattern (Sect. 4.1.3). A neural network infers the magnetic near-field distribution from the structure of a photonic device. The proposed loss function for training the

[1] Note that in some use-cases consistency is favored over accuracy and instead, training achieves $\mathcal{D}(x) = \mathcal{P}(y)$ for some $x, y \in \mathcal{X}$.

network contains two additive terms: the usual data-driven loss term and an additional Maxwell loss term, in the spirit of the PINN approach. The Maxwell loss measures the failure of the magnetic field to comply with the vector wave equation. Both loss terms can be balanced by a hyperparameter. The method works most effectively in a regime where more weight is given to data loss. The Maxwell loss can be seen as a regularization, "*to push the outputted data to be more wavelike*".

- Object detection and tracking: Consider the task of learning to detect and track objects in a video. A deep learning approach would typically require labeled examples of input output pairs, such that a neural network (for video typically a CNN) can be trained to predict the outputs given the inputs. Stewart and Ermon [82] show that the labeled examples can be replaced by domain knowledge such as physical laws. Instead of using loss functions such as predictive accuracy, they translate physical laws into penalty and regularization terms, yielding loss functions that do not require labels.

Example The design pattern of physical constraints can be used for simulating the electrodynamics of an unknown material. The laws of electrodynamics combine the three sub-models (11)–(13). While Maxwell's equations (11)–(12), i.e., sub-models U_1, U_2, are accepted as first principles, the constitutive relations (13), i.e., sub-model U_3, is heuristic. Typically an overly simplistic (e.g., polynomial) model is fitted to measurements of material properties. The resulting modeling error compounds when all sub-models are put together.

In [10, 83, 84] an alternative approach for magnetostatic problems is presented, where the sub-model U_3 is discarded altogether. Instead, the authors develop a *hybrid solver* that acts directly on the material data to find the best fitting model within all models that are consistent with Maxwell's equations (U_1 and U_2 in (11)–(12)). This line of research goes back to the seminal paper [85]. In the magnetostatic case, Maxwell's equations reduce to the PDE constraints

$$\operatorname{curl}\vec{H} - \vec{j} = 0, \qquad \operatorname{div}\vec{B} = 0. \tag{19}$$

We denote by $\mathcal{P}$ the space of physics-conforming magnetostatic fields. These are vector fields $z = (\vec{H}, \vec{B})$ that exhibit sufficient regularity and are constrained by (19).

The measurement data consist of data points $z_i^* = (\vec{H}_i^*, \vec{B}_i^*)$, $i = 1, \dots, N$, that are collected in a set $\widetilde{\mathcal{D}}$. These data are lifted to the space $\mathcal{D}$ of piece-wise constant vector fields $z^* = (\vec{H}, \vec{B})$ with respect to a computational grid, such that $(\vec{H}(x), \vec{B}(x)) \in \widetilde{\mathcal{D}}$ almost everywhere. Obviously, the data-induced space $\mathcal{D}$ characterizes the magnetic material properties only imperfectly, since it is based on a finite number of measurement points and a spatial discretization by the underlying grid.

The solution is formally given by $\mathcal{S} = \mathcal{P} \cap \mathcal{D}$. These are fields that fulfill Maxwell's equations, while being compatible with the measurement data. However, for a finite number of data points, this set is very likely to be empty. Even for an infinite data set, the noise that is always inherent to measurements may lead to an empty set. Therefore, we define the solution by the relaxed condition

$$\mathcal{S} = \operatorname*{arg\,min}_{z \in \mathcal{P}} \left(\min_{z^* \in \mathcal{D}} \| z - z^* \| \right), \tag{20}$$

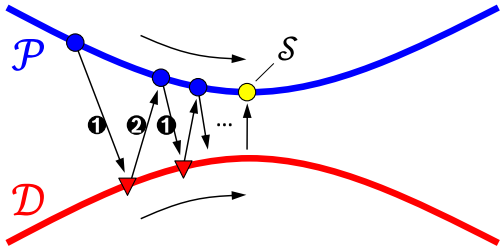


Figure 8 Iterative hybrid solver. The fixed point iteration alternates between discrete optimization problems (1) with solutions in the data-induced space $\mathcal{D}$ (red triangles), and variational problems (2) with solutions in the physics-conforming space $\mathcal{P}$ (blue circles), the latter being accomplished by a modified finite element solver. [Adapted from [10], Fig. 3.] The algorithm is an instance of the hard constraints design pattern, see paragraph 4.1.4.1, in particular Fig. 7a

where $\|\cdot\|$ is a suitable norm which serves as loss function. We accept a solution z that conforms to Maxwell's equations, while minimizing the loss function, hence being "closest" to the available measurement data.

The hybrid solver is organized as a fixed point iteration, see Fig. 8. Under convexity assumptions this algorithm converges to the solution of (20). Furthermore, it can be shown that the conventional solution is recovered with measurement data sets of increasing size.

Note that even the conventional approach could be interpreted in terms of design patterns. If a spline curve is learned from measured material data and then used in a finite element solver, this could be understood as feature learning, in the sense of Sect. 4.1.3. A more sophisticated model, e.g., explicitly accounting for the Rayleigh region (low field magnetic behavior of ferromagnetic materials), could be seen as a hierarchical setup because physics knowledge is leveraged already in the learning process.

Discussion The physical constraints design pattern provides an intuitive interface for incorporating desired behavior grounded in first-principles into a data-driven model. Especially when using hard constraints at the output level, one is guaranteed that model outputs lie within a plausible range. Depending on how they are implemented, hard constraints can introduce non-differentiable nonlinearities which can make gradient-based optimization challenging. In those cases, soft constraints might produce a favorable optimization landscape. However, while soft constraints are usually easier to work with during the modeling stage, they provide no guarantee that the desired constraint is implemented exactly. In addition, it is not always straightforward to fit a hybrid model that incorporates multiple physical constraints.

An important advantage of the physical constraints design pattern is the potential for increased data efficiency. By integrating physical constraints, the complexity (e.g. dimensionality) of the problem can be reduced, potentially diminishing the volume of required training data. This pre-structuring of the search space accelerates the training of data-based models. Moreover, when $\mathcal{P}$ provides a training signal, such as a physically informed self-supervised loss, it can obviate the need for the often expensive labeling process, and instead the training of the data-driven component can benefit from available unlabeled data.

The design pattern of physical constraints results in hybrid models that benefit from prior knowledge. Priors related to geometry, shapes, invariances, and equivariances, as

seen in geometric deep learning [86, 87], enable the selection of optimal models, bolstering their accuracy and robustness. Furthermore, the explainability of the model is heightened. By grounding the model in physical principles, its topologies become more interpretable, facilitating a clearer understanding of its data-driven components and their interactions with the physical constraints.

The relationship between physical constraints and feature learning There are use cases that fit both the physical constraints and the feature learning design pattern, so we describe their relationship here. Unlike hard constraints, soft constraints are only used during the training phase. At deployment time, there is no more computation derived from first principles; instead, the data-driven model has learned to emulate the desired behavior. In contrast, a hard constraint is not removed at deployment time. In [63], there are hard and soft constraints: a neural network, i.e. a data-driven model is used to predict the magnetic near field distribution. A soft constraint based on Maxwell's equations, ensures that the predictions adhere with the laws of physics. These predictions are then processed by a computational block $\mathcal{P}$ that implements a discrete version of Ampère's law, followed by a near-to-far field transformation. $\mathcal{P}$ can be interpreted as imposing a hard constraint since it is guaranteed to produce a prediction of the electric field that is consistent with the magnetic field prediction of $\mathcal{D}$. The constraint is used both during training and at test time. In this example, the soft constraint is on an intermediate output of the model, while the hard constraint affects the final output of the model. In general, constraints can either affect intermediate of final computation, or parameter values of the model, or the structure of the model. Note that a hybrid modeling solution, where a computational block $\mathcal{D}$ is followed by a hard constraint, i.e. a constraint that is not removed after training and that affects the final computational output, is consistent with Eq. (17) and therefore also fits the feature learning design pattern. In fact, [63] was presented as an example of the feature learning design pattern in Sect. 4.1.3 for that reason.

It is quite common for hybrid modeling solutions to combine multiple design patterns. In the next section, we describe design patterns for pattern composition.

4.2 Composition patterns for hybrid modeling

Next, we describe composition patterns. They provide patterns for composing the base patterns from Sect. 4.1 into more elaborate hybrid modeling solutions.

4.2.1 Recurrent composition

An important design pattern, especially when dealing with sequential data, is recurrent composition. The recurrence design pattern encompasses a wide range of models involving an internal state that is updated sequentially. This pattern is observed in recurrent neural networks and numerical integration schemes for differential equations. The main

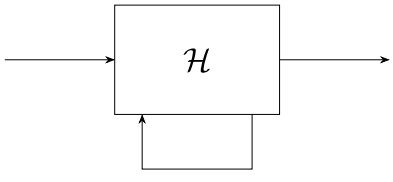


Figure 9 The design pattern of recurrent composition (Sect. 4.2.1) has a computational block that is repeatedly applied to sequential inputs. Typically, it has an internal state that is updated sequentially with each execution of the computational block

principle is to compute the dynamics of a system through a recursive update rule as depicted in the block diagram in Fig. 9. The computational block $\mathcal{H}$ for the update rule can either be data-driven, or based on first principles, or consist of a hybrid computational block that relies on one or more of the design patterns presented above.

The recurrence design pattern features an internal state s which is updated sequentially over time. The state at time t is computed from a previous state:

$$s_t = \mathcal{H}(s_{t-1}, \ldots). \tag{21}$$

The function $\mathcal{H}(\cdot)$ can have additional inputs, such as observations from a sequence $x_1, x_2, \ldots, x_T$, the time t, and the time difference Δ_t between s_{t-1} and s_t. In control or signal processing applications, there might also be a control input. Whether $\mathcal{H}$ is data-driven, physics-based, or hybrid, depends on the use-case. Some typical use cases are described next.

Typical use cases

- Recurrent neural networks in deep learning: Recurrent neural networks (RNNs) are powerful sequence models. When trained on sequences of observations $x_1, x_2, \ldots, x_T$, they have the capacity to leverage s_t as a hidden state to summarize all the relevant information in the sequence up until time t. At each time step the hidden state is updated based on the current observation and the previous hidden state $s_t = \mathcal{H}(s_{t-1}, x_t)$. To obtain a prediction, the hidden state can then be mapped to the desired output. For a vanilla RNN, $\mathcal{H}(s_{t-1}, x_t)$ will be an affine transformation followed by a non-linearity, but other choices exist, such as gated recurrent units (GRUs) [88] and LSTMs [61]. For most RNNs, $\mathcal{H}$ is data-driven, meaning that the parameters are learned by fitting to training data [6].
- Numerical integration: A dynamical system is often described by an ODE as in Eq. (6). Some ordinary differential equations (ODEs) allow recovering the system state using analytic solutions but in many interesting cases numerical integration schemes have to be employed to compute the state of the system as a function of time. In a numerical integration scheme, the system state is approximated by s_t, which can also be thought of as the intermediate integration results at time t. Typically, there is a recursive update rule where s_t is computed based on a previous state s_{t-1} as well as the step size and the vector field f. In the backward Euler method for example $s_t = \mathcal{H}(s_{t-1}, \Delta_t, s_t, t) = s_{t-1} + \Delta_t f(s_t, t; \theta)$, with f and θ as defined in Eq. (6).
- Neural ODEs: Neural ODEs [89] are a model class at the intersection of deep learning and differential equations. The vector field f in Eq. (6) is parameterized by a neural network. The result is a flexible dynamics model whose parameters are fitted in a data-driven way. Neural ODEs rely heavily on numerical integration: The system has to be integrated to form a prediction, and back-propagation through the ODE solver can be handled efficiently by numerically integrating an auxiliary (adjoint state) ODE backward in time [90].
- State estimation: State estimation is a crucial process in control theory and signal processing that aims to accurately determine the state of a dynamic system based on noisy and potentially incomplete measurements over time [91]. The relationship between the inputs and the outputs of the dynamical system is often described by

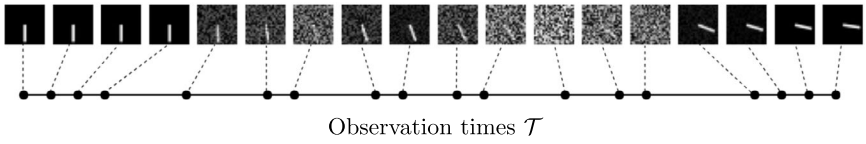


Figure 10 The CRU [94] can help infer the pendulum angle from images observed at irregular time intervals

ODEs. In addition to predicting the system state by (numerical) integration of the dynamics, state estimation also entails accounting for the influence of control inputs, and for measurement noise, thereby systematically improving the accuracy of the system state's prediction. One notable example of an algorithm used for state estimation is the Kálmán filter [92], which provides the optimal solution to estimate the state of a linear dynamic system perturbed by Gaussian noise. For state estimation in non-linear systems, variations such as the Extended Kálmán Filter (EKF) or Unscented Kálmán Filter (UKF) are often used [93].

Example Modern recurrent neural networks typically assume regular time intervals between observations. A notable exception is the continuous recurrent unit (CRU) which can be used to model irregularly sampled time series [94]. It assumes a hidden state that evolves according to a linear stochastic differential equation (SDE). To model a sequence, each measurement is first mapped into a latent space by a neural network. The transformed observation is then treated as an observation of the latent state, which can now be inferred via state estimation, specifically the continuous-discrete formulation of the Kálmán filter [95].

The recursive update of the CRU is a hybrid block, combining a data-driven block $\mathcal{D}$, which consists of a neural network and is applied to each measurement x_t, and a state estimation block $\mathcal{P}$ consisting of the updates of the continuous-discrete Kálmán Filter,

$$s_t = \mathcal{H}(s_{t-1}, x_t, \Delta_t) = \mathcal{P}\big(s_{t-1}, \mathcal{D}(x_t), \Delta_t\big). \tag{22}$$

As an illustrative example, consider the problem of predicting the angle of a pendulum from noisy images taken at irregular time intervals (Fig. 10). Since some of the images are very noisy, angle prediction will benefit from a model that takes temporal structure into account, such as the CRU. While the pendulum dynamics are relatively simple and can be described by a second-order ODE, inferring them from high-dimensional inputs such as images is non-trivial. The CRU can accurately predict the angle, optimally accounting for different sources of noise.

Discussion The concept of recurrence is useful in hybrid modeling and machine learning for several reasons. First, recurrent models can learn to recognize patterns across time. For example, they can learn to predict the next word in a sentence based on the context provided by the preceding words. This is possible, because the model has a way of remembering the previous context, enabling it to learn how the current state is influenced by the previous states.

Another advantage of this design pattern is parameter sharing. Recurrent models apply the same set of weights to the inputs at each time step. This means that they are making

the assumption that the same patterns that are useful to process at one point in time will be useful to process at other points in time. This significantly reduces the number of parameters in the model, which can help to avoid overfitting and make the model easier to train. The main limitations of this design pattern are of computational nature. Dynamical systems, especially when they are stiff, are difficult to optimize numerically. Similarly, recurrent architectures in machine learning are sometimes difficult to optimize. Numerical instabilities can lead to exploding or vanishing gradients.

Finally, recurrence provides a natural modeling paradigm to deal with input and output sequences of variable length. For example, you can use an RNN to process a sentence of any length and produce a sentiment score. Traditional methods like feed-forward neural networks cannot handle this variability as they require fixed-size input vectors.

4.2.2 Hierarchical pattern composition

The pattern of pattern composition emphasizes the flexibility and composability of hybrid modeling design patterns. In this pattern, the concept is that hybrid models themselves can serve as building blocks for constructing more complex hybrid models. To represent this idea, we introduce the following notation:

Let $\mathcal{H}(\mathcal{P}, \mathcal{D})$ denote a hybrid model that combines a physics-based model $\mathcal{P}$ and a data-driven model $\mathcal{D}$. The pattern of pattern composition suggests that $\mathcal{P}$ and $\mathcal{D}$ themselves can be hybrid models. We can represent this idea by considering two hybrid models, $\mathcal{H}_1$ and $\mathcal{H}_2$, such that:

$$\mathcal{H}(\mathcal{P}, \mathcal{D}), \quad \text{where } \mathcal{P} = \mathcal{H}_1(\mathcal{P}_1, \mathcal{D}_1) \text{ and } \mathcal{D} = \mathcal{H}_2(\mathcal{P}_2, \mathcal{D}_2). \tag{23}$$

This notation conveys that $\mathcal{H}_1$ and $\mathcal{H}_2$, each being a combination of physics-based and data-driven models, are now being combined to form a new, more complex hybrid model $\mathcal{H}$. This pattern highlights the recursive nature of hybrid modeling, where models can be built upon one another in a hierarchical manner, leading to increasingly sophisticated representations of the underlying system.

By applying the pattern of pattern composition, practitioners can create multi-layered hybrid models that address various aspects of the problem at hand, and tackle more complex challenges by leveraging the strengths of multiple modeling paradigms. This approach also allows researchers to explore novel combinations of the design patterns introduced in this paper, potentially leading to new insights and advances in the field of hybrid modeling.

Typical use cases

- Lake Temperature Modeling: Daw et al. [96] present a hybrid modeling solution for lake temperature modeling. The goal is to predict temperature from physical quantities that are known to drive lake temperature. The authors assume access to observations and a physics-based simulation of lake temperature $\mathcal{P}_1$, which might be inaccurate due to inadequate calibration or missing physics. The physics-based pre-processing design pattern is used to first augment the input variables with the potentially inaccurate but still useful predictions of $\mathcal{P}_1$. The original observed features x are concatenated with these physically preprocessed predictions to $[x, \mathcal{P}_1(x)]$, which is then fed into a data-driven model that is further subjected to the design pattern of

physical constraints. An additional loss term $\mathcal{P}_2$ assures that the predictions fulfill plausible density-depth and density-temperature relations. The combined hybrid model can be written as $\mathcal{H}(x) = \mathcal{D}_{\mathcal{P}_2}([x, \mathcal{P}_1(x)])$.

- ODEs with missing physics: Another example of hierarchical pattern composition is a hybrid neural ODE [97] where the vector field f of the ODE in Eq. (6) is parameterized by multiple terms which are added according to the delta model design pattern. This can be beneficial when part of the dynamics are explicitly known, while other missing parts are modeled in a data-driven way, typically with a neural network. Extensions to stochastic dynamical systems also exist [98].
- Dynamics modeling with unknown unknowns: Long et al. [99] propose a hybrid model for dynamics modeling with many unknowns. For example, in a fluid dynamics application, it is known that the dynamics are governed by Navier-Stokes equations, but they cannot be solved without knowledge of the geometry of the system or access to physical parameters such as viscosity, material density, or external forces. In such a setting the authors suggest employing a learnable PDE solver $\mathcal{H}_1$ based on cellular neural networks. This learnable PDE solver can be seen as a hybrid approach: it is a data-driven approach where missing physical parameters are learned from data, but its structure is derived from first-principles and adheres to the underlying PDE. To deal with missing inputs, e.g. with unobserved external perturbations to the inputs, the authors further employ the feature learning design pattern. A data driven model $\mathcal{D}$, specifically a convolutional LSTM, predicts the missing inputs, which are then fed into $\mathcal{H}_1$, resulting in the composed hybrid model $\mathcal{H}_2(x) = \mathcal{H}_1(\mathcal{D}(x))$.

Example Many time-series algorithms face challenges when attempting to simultaneously capture short- and long-term effects. Data-driven models (denoted as $\mathcal{D}$) often excel at providing detailed short-term predictions. However, even small errors in their short-term forecasts can accumulate over time, leading to deteriorated long-term performance. In contrast, models capable of reliable long-term predictions can often be developed by leveraging physics-based simulations (referred to as $\mathcal{P}$).

The work of [100] addresses this challenge by decomposing predictions into two components: one that accurately predicts long-term behavior and another one that excels at short-term prediction. The long-term predictions are generated by the physics-based model $\mathcal{P}$, while the short-term predictions are generated by the data-driven model $\mathcal{D}$. To ensure that each model operates within its domain of competence, the authors introduce two hard constraints: They apply a low-pass filter (F_{low}) to the predictions of the physics-based model $\mathcal{P}$ and a high-pass filter (F_{high}) to the predictions of the data-driven model $\mathcal{D}$. Finally, the two prediction components are combined using the delta pattern resulting in a complementary filtering approach depicted in Fig. 11:

$$\mathcal{H}(x) = F_{\text{low}}\big(\mathcal{P}(x)\big) + F_{\text{high}}\big(\mathcal{D}(x)\big). \tag{24}$$

The fusion of high and low-frequency information from different signals is a well-established technique in control engineering and signal processing applications. An illustrative example can be found in robotics, specifically in tilt estimation [101]. In this context, accelerometer and gyroscope measurements are often recorded simultaneously. The gyroscope delivers precise short-term position estimates, but due to integration

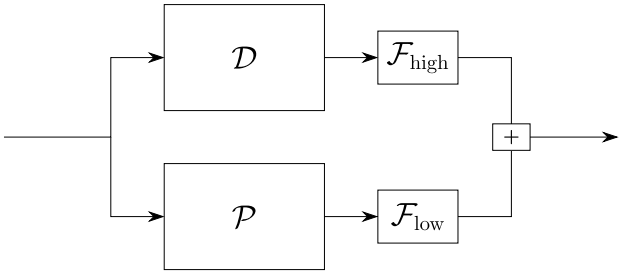


Figure 11 The block diagram of the hybrid modeling example presented in Eq. (24) which is taken from Ensinger et al. [100]. This composed design pattern combines the delta model (Sect. 4.1.1) with the physical constraints design pattern (Sect. 4.1.4). The predictions of the data-driven model are fed through a high-pass filter, while the physics-based predictions are processed by a low-pass filter. The overarching delta model combines the predictions additively. (Note: Since the constraints in the example are at the output level, they are visualized in a block diagram notation of the feature learning design pattern. See the last section of Sect. 4.1.4.2 for a discussion of the relationship between physical constraints and feature learning)

at each time step, accumulating errors introduce drift in the long-term. In contrast, accelerometer-based position estimates are more stable over the long-term but exhibit substantial noise, making them less reliable for short-term predictions. As a consequence, the position estimate can be significantly improved by combining both signals after applying a high-pass filter to the gyroscope measurements and a low-pass filter to the accelerometer measurements.

Discussion Only through composition do the design patterns reach their full potential. While here we have provided three examples, for how design patterns can be composed, the possibilities are endless. While each of the design patterns has their own set of advantages, through composition we can build hybrid models that combine many of these advantages into a single modeling solution.

5 Conclusion

In conclusion, this paper has presented a systematic exploration of various design patterns for hybrid modeling, showcasing the potential of combining the strengths of both data-driven and mechanistic models to address complex problems in diverse domains. These design patterns provide a unified framework for understanding and organizing the myriad approaches used in hybrid modeling, and they facilitate the sharing of knowledge and expertise across application domains.

The identification and formalization of these design patterns serve as a valuable resource for researchers and practitioners in the field, allowing them to better understand the underlying principles, common challenges, and potential solutions for hybrid modeling. By providing a higher level of abstraction, these design patterns enable the development of more generalizable and standardized tools and techniques, leading to improved efficiency and reliability of the modeling process.

Furthermore, the use of design patterns can help to identify common limitations and areas for improvement in hybrid modeling, thus guiding future research directions and fostering innovation. As the field of hybrid modeling continues to evolve, we anticipate that the exploration and refinement of these design patterns will play a crucial role in shaping

the development of new models, methods, and applications, ultimately contributing to the advancement of our understanding and the solution of real-world problems.

In summary, the design patterns presented in this paper offer a valuable framework for organizing and advancing the field of hybrid modeling. By embracing the principles of abstraction and generalization, researchers and practitioners can better address the unique challenges and complexities of their domains, while also contributing to the broader knowledge and understanding of hybrid modeling as a whole.

Funding
All authors are employed by Robert Bosch GmbH or its subsidiaries.

Abbreviations
NN, neural network; LSTM, long short-term memory; ROM, reduced-order model; CNN, convolutional neural network; HYM, hybrid modeling; PINN, physics-informed neural network; RNN, recurrent neural network; GRU, gated recurrent unit; CRU, continuous recurrent unit; ODE, ordinary differential equation; PDE, partial differential equation; SDE, stochastic differential equation.

Data availability
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
All authors are employed by Robert Bosch GmbH or its subsidiaries.

Author contributions
MR, SK, and BR wrote the paper. All authors read and approved the final manuscript.

Author details
[1]Bosch Center for AI, Pittsburgh, PA, USA. [2]University of Wisconsin – Madison, Madison, WI, USA. [3]Bosch Center for AI, Renningen, Germany. [4]University of Jyväskylä, Jyväskylä, Finland. [5]ETH Zurich, Zurich, Switzerland.

Received: 22 December 2023 Accepted: 22 February 2024 Published online: 19 March 2024

References

1. Eck C, Garcke H, Knabner P. Mathematical modeling. Berlin: Springer; 2017.
2. Gershenfeld NA. The nature of mathematical modeling. Cambridge: Cambridge University Press; 1999.
3. Deuflhard P, Bornemann F. Scientific computing with ordinary differential equations. vol. 42. Berlin: Springer; 2012.
4. Bishop CM, Nasrabadi NM. Pattern recognition and machine learning. vol. 4. Berlin: Springer; 2006.
5. Murphy KP. Machine learning: a probabilistic perspective. Cambridge: MIT Press; 2012.
6. Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge: MIT Press; 2016.
7. Karpatne A, Atluri G, Faghmous JH, Steinbach M, Banerjee A, Ganguly A, Shekhar S, Samatova N, Kumar V. Theory-guided data science: a new paradigm for scientific discovery from data. IEEE Trans Knowl Data Eng. 2017;29(10):2318–31.
8. Rueden L, Mayer S, Beckh K, Georgiev B, Giesselbach S, Heese R, Kirsch B, Pfrommer J, Pick A, Ramamurthy R et al. Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems. IEEE Trans Knowl Data Eng. 2021;35(1):614–33.
9. Willard J, Jia X, Xu S, Steinbach M, Kumar V. Integrating scientific knowledge with machine learning for engineering and environmental systems. ACM Comput Surv. 2022;55(4):1–37.
10. Kurz S, De Gersem H, Galetzka A, Klaedtke A, Liebsch M, Loukrezis D, Russenschuck S, Schmidt M. Hybrid modeling: towards the next level of scientific computing in engineering. J Math Ind. 2022;12(1):8.
11. Hilborn R, Mangel M. The ecological detective: confronting models with data (MPB-28). Princeton: Princeton University Press; 2013.
12. Stokes JM, Yang K, Swanson K, Jin W, Cubillos-Ruiz A, Donghia NM, MacNair CR, French S, Carfrae LA, Bloom-Ackermann Z et al. A deep learning approach to antibiotic discovery. Cell. 2020;180(4):688–702.
13. Beucler T, Rasp S, Pritchard M, Gentine P. Achieving conservation of energy in neural network emulators for climate modeling. arXiv preprint. 2019. arXiv:1906.06622.
14. Reichstein M, Camps-Valls G, Stevens B, Jung M, Denzler J, Carvalhais N. Deep learning and process understanding for data-driven Earth system science. Nature. 2019;566(7743):195–204.
15. Kaplan D, Glass L. Understanding nonlinear dynamics. Berlin: Springer; 2012.

16. Grasman J. Asymptotic methods for relaxation oscillations and applications. Applied mathematical sciences. 1987.
17. Brauer F, Castillo-Chavez C, Castillo-Chavez C. Mathematical models in population biology and epidemiology. vol. 2. Berlin: Springer; 2012.
18. Braess D. Finite elements: theory, fast solvers, and applications in solid mechanics. Cambridge: Cambridge University Press; 2007.
19. Logan JD. Applied partial differential equations. Berlin: Springer; 2014.
20. Hillar C, Sommer F. Comment on the article "distilling free-form natural laws from experimental data". arXiv preprint. 2012. arXiv:1210.7273.
21. Nash JC, Walker-Smith M. Nonlinear parameter estimation. An integrated system on BASIC. NY, Basel. 1987;493.
22. Alonge F, D'Ippolito F, Ferrante G, Raimondi F. Parameter identification of induction motor model using genetic algorithms. IEE Proc, Control Theory Appl. 1998;145(6):587–93.
23. Schwaab M, Biscaia EC Jr, Monteiro JL, Pinto JC. Nonlinear parameter estimation through particle swarm optimization. Chem Eng Sci. 2008;63(6):1542–52.
24. Perdikaris P, Karniadakis GE. Model inversion via multi-fidelity Bayesian optimization: a new paradigm for parameter estimation in haemodynamics, and beyond. J R Soc Interface. 2016;13(118):20151107.
25. Schmidt M, Lipson H. Distilling free-form natural laws from experimental data. Science. 2009;324(5923):81–5.
26. Bongard J, Lipson H. Automated reverse engineering of nonlinear dynamical systems. Proc Natl Acad Sci. 2007;104(24):9943–8.
27. Cranmer K, Brehmer J, Louppe G. The frontier of simulation-based inference. Proc Natl Acad Sci. 2020;117(48):30055–62.
28. Kennedy MC, O'Hagan A. Bayesian calibration of computer models. J R Stat Soc, Ser B, Stat Methodol. 2001;63(3):425–64.
29. Calderhead B, Girolami M, Lawrence N. Accelerating bayesian inference over nonlinear differential equations with Gaussian processes. Adv Neural Inf Process Syst. 2008;21.
30. Kersting H, Krämer N, Schiegg M, Daniel C, Tiemann M, Hennig P. Differentiable likelihoods for fast inversion of 'likelihood-free' dynamical systems. In: Hal III D, Singh A, editors. Proceedings of the 37th international conference on machine learning. Proceedings of machine learning research. vol. 119. PMLR; 2020. p. 5198–208. https://proceedings.mlr.press/v119/kersting20a.html.
31. Williams CK, Rasmussen CE. Gaussian processes for machine learning. vol. 2. Cambridge: MIT press; 2006.
32. Quinonero-Candela J, Rasmussen CE. A unifying view of sparse approximate Gaussian process regression. J Mach Learn Res. 2005;6:1939–59.
33. Snelson E, Ghahramani Z. Sparse gaussian processes using pseudo-inputs. Adv Neural Inf Process Syst. 2005;18.
34. Titsias M. Variational learning of inducing variables in sparse Gaussian processes. In: Artificial intelligence and statistics. PMLR; 2009. p. 567–74.
35. Wilk M, Rasmussen CE, Hensman J. Convolutional Gaussian processes. Adv Neural Inf Process Syst. 2017;30.
36. Alvarez M, Luengo D, Lawrence ND. Latent force models. In: Artificial intelligence and statistics. PMLR; 2009. p. 9–16.
37. Harkonen M, Lange-Hegermann M, Raita B. Gaussian process priors for systems of linear partial differential equations with constant coefficients. In: International conference on machine learning. PMLR; 2023. p. 12587–615.
38. Rabenstein G, Demir O, Trachte A, Graichen K. Data-driven feed-forward control of hydraulic cylinders using Gaussian process regression for excavator assistance functions. In: 2022 IEEE conference on control technology and applications (CCTA). New York: IEEE; 2022. p. 962–9.
39. Yıldız Ç, Kandemir M, Rakitsch B. Learning interacting dynamical systems with latent Gaussian process odes. Adv Neural Inf Process Syst. 2022;35:9188–200.
40. Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y. What is the best multi-stage architecture for object recognition? In: 2009 IEEE 12th international conference on computer vision. New York: IEEE; 2009. p. 2146–53.
41. Wilson AG, Hu Z, Salakhutdinov R, Xing EP. Deep kernel learning. In: Artificial intelligence and statistics. PMLR; 2016. p. 370–8.
42. Wolpert DH. Stacked generalization. Neural Netw. 1992;5(2):241–59.
43. Breiman L. Bagging predictors. Mach Learn. 1996;24:123–40.
44. Schapire RE. The strength of weak learnability. Mach Learn. 1990;5:197–227.
45. Wong F, Zheng EJ, Valeri JA, Donghia NM, Anahtar MN, Omori S, Li A, Cubillos-Ruiz A, Krishnan A, Jin W et al. Discovery of a structural class of antibiotics with explainable deep learning. Nature. 2024;626:177–85.
46. Slater L, Arnal L, Boucher M-A, Chang AY-Y, Moulds S, Murphy C, Nearing G, Shalev G, Shen C, Speight L et al. Hybrid forecasting: blending climate predictions with AI models. 2023.
47. Venkatasubramanian V. The promise of artificial intelligence in chemical engineering: is it here, finally? AIChE J. 2019;65(2):466–78.
48. Stosch M, Oliveira R, Peres J, Azevedo SF. Hybrid semi-parametric modeling in process systems engineering: past, present and future. Comput Chem Eng. 2014;60:86–101.
49. Thompson ML, Kramer MA. Modeling chemical processes using prior knowledge and neural networks. AIChE J. 1994;40(8):1328–40.
50. Zendehboudi S, Rezaei N, Lohi A. Applications of hybrid models in chemical, petroleum, and energy systems: a systematic review. Appl Energy. 2018;228:2539–66.
51. Xu T, Valocchi AJ. Data-driven methods to improve baseflow prediction of a regional groundwater model. Comput Geosci. 2015;85:124–36.
52. Wang J-X, Wu J-L, Xiao H. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. Phys Rev Fluids. 2017;2(3):034603.
53. Levine M, Stuart A. A framework for machine learning of model error in dynamical systems. Commun Am Math Soc. 2022;2(07):283–344.
54. Qiu C, Pfrommer T, Kloft M, Mandt S, Rudolph M. Neural transformation learning for deep anomaly detection beyond images. In: International conference on machine learning. PMLR; 2021. p. 8703–14.
55. Hershey S, Chaudhuri S, Ellis DP, Gemmeke JF, Jansen A, Moore RC, Plakal M, Platt D, Saurous RA, Seybold B et al. CNN architectures for large-scale audio classification. In: 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP). New York: IEEE; 2017. p. 131–5.

56. Huang J, Chen B, Yao B, He W. ECG arrhythmia classification using STFT-based spectrogram and convolutional neural network. IEEE Access. 2019;7:92871–80.
57. Patel K, Rambach K, Visentin T, Rusev D, Pfeiffer M, Yang B. Deep learning-based object classification on automotive radar spectra. In: 2019 IEEE radar conference (RadarConf). New York: IEEE; 2019. p. 1–6.
58. Hamrock BJ, Anderson WJ. Rolling-element bearings. Technical report. 1983.
59. Sadoughi M, Hu C. Physics-based convolutional neural network for fault diagnosis of rolling element bearings. IEEE Sens J. 2019;19(11):4181–92.
60. Bedi J, Toshniwal D. Empirical mode decomposition based deep learning for electricity demand forecasting. IEEE Access. 2018;6:49144–56.
61. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
62. Rabiner LR, Schafer RW. Introduction to digital speech processing. Hanover: Now Publishers; 2007.
63. Chen M, Lupoiu R, Mao C, Huang D-H, Jiang J, Lalanne P, Fan JA. Wavey-net: physics-augmented deep learning for high-speed electromagnetic simulation and optimization. CoRR. 2022.
64. Long Z, Lu Y, Ma X, Dong B. PDE-Net: learning PDEs from data. In: International conference on machine learning. PMLR; 2018. p. 3208–16.
65. Long Z, Lu Y, Dong B. PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network. J Comput Phys. 2019;399:108925.
66. Boullé N, Townsend A. A mathematical guide to operator learning. arXiv preprint. 2023. arXiv:2312.14688.
67. Kovachki N, Li Z, Liu B, Azizzadenesheli K, Bhattacharya K, Stuart A, Anandkumar A. Neural operator: learning maps between function spaces with applications to PDEs. J Mach Learn Res. 2023;24(89):1–97.
68. Li Z, Kovachki NB, Choy C, Li B, Kossaifi J, Otta SP, Nabian MA, Stadler M, Hundt C, Azizzadenesheli K, et al. Geometry-informed neural operator for large-scale 3D PDEs. arXiv preprint. 2023. arXiv:2309.00583.
69. Parekh V, Flore D, Schöps S. Performance analysis of electrical machines using a hybrid data-and physics-driven model. IEEE Trans Energy Convers. 2022;38(1):530–9.
70. Raonić B, Molinaro R, Rohner T, Mishra S, Bezenac E. Convolutional neural operators. arXiv preprint. 2023. arXiv:2302.01178.
71. Ganchev M, Kral C, Oberguggenberger H, Wolbank T. Sensorless rotor temperature estimation of permanent magnet synchronous motor. In: IECON 2011 – 37th annual conference of the IEEE industrial electronics society. New York: IEEE; 2011. p. 2018–23.
72. Liu L, Kuo SM, Zhou M. Virtual sensing techniques and their applications. In: 2009 international conference on networking, sensing and control. New York: IEEE; 2009. p. 31–6.
73. Liu D, Wang Y. Multi-fidelity physics-constrained neural network and its application in materials modeling. J Mech Des. 2019;141(12):121403.
74. Bridle JS. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: Neurocomputing: algorithms, architectures and applications. Berlin: Springer; 1990. p. 227–36.
75. Greydanus S, Dzamba M, Yosinski J. Hamiltonian neural networks. Adv Neural Inf Process Syst. 2019;32.
76. Toth P, Rezende DJ, Jaegle A, Racanière S, Botev A, Higgins I. Hamiltonian generative networks. arXiv preprint. 2019. arXiv:1909.13789.
77. Cranmer M, Greydanus S, Hoyer S, Battaglia P, Spergel D, Ho S. Lagrangian neural networks. arXiv preprint. 2020. arXiv:2003.04630.
78. McFall KS, Mahan JR. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. IEEE Trans Neural Netw. 2009;20(8):1221–33.
79. Sukumar N, Srivastava A. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. Comput Methods Appl Mech Eng. 2022;389:114333.
80. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys. 2019;378:686–707.
81. De Bézenac E, Pajot A, Gallinari P. Deep learning for physical processes: incorporating prior scientific knowledge. J Stat Mech Theory Exp. 2019;2019(12):124009.
82. Stewart R, Ermon S. Label-free supervision of neural networks with physics and domain knowledge. In: Thirty-first AAAI conference on artificial intelligence. 2017.
83. De Gersem H, Galetzka A, Ion IG, Loukrezis D, Römer U. Magnetic field simulation with data-driven material modeling. IEEE Trans Magn. 2020;56(8):1–6.
84. Galetzka A, Loukrezis D, De Gersem H. Data-driven solvers for strongly nonlinear material response. Int J Numer Methods Eng. 2021;122(6):1538–62.
85. Kirchdoerfer T, Ortiz M. Data-driven computational mechanics. Comput Methods Appl Mech Eng. 2016;304:81–101.
86. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P. Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag. 2017;34(4):18–42.
87. Bronstein MM, Bruna J, Cohen T, Veličković P. Geometric deep learning: grids, groups, graphs, geodesics, and gauges. arXiv preprint. 2021. arXiv:2104.13478.
88. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint. 2014. arXiv:1412.3555.
89. Chen RT, Rubanova Y, Bettencourt J, Duvenaud DK. Neural ordinary differential equations. Adv Neural Inf Process Syst. 2018;31.
90. LeCun Y, Touresky D, Hinton G, Sejnowski T. A theoretical framework for back-propagation. In: Proceedings of the 1988 connectionist models summer school. vol. 1. San Mateo, CA, USA. 1988. p. 21–8.
91. Särkkä S, Svensson L. Bayesian filtering and smoothing. vol. 17. Cambridge: Cambridge University Press; 2023.
92. Kálmán RE. A new approach to linear filtering and prediction problems. 1960.
93. Julier SJ, Uhlmann JK. Unscented filtering and nonlinear estimation. Proc IEEE. 2004;92(3):401–22.
94. Schirmer M, Eltayeb M, Lessmann S, Rudolph M. Modeling irregular time series with continuous recurrent units. In: International conference on machine learning. PMLR; 2022. p. 19388–405.
95. Jazwinski AH. Stochastic processes and filtering theory. Courier Corporation; 2007.

96. Daw A, Karpatne A, Watkins W, Read J, Kumar V. Physics-guided neural networks (PGNN): an application in lake temperature modeling. arXiv preprint. 2017. arXiv:1710.11431.
97. Yin Y, Le Guen V, Dona J, Bézenac E, Ayed I, Thome N, Gallinari P. Augmenting physical models with deep networks for complex dynamics forecasting. J Stat Mech Theory Exp. 2021;2021(12):124012.
98. Haußmann M, Gerwinn S, Look A, Rakitsch B, Kandemir M. Learning partially known stochastic dynamics with empirical PAC Bayes. In: International conference on artificial intelligence and statistics. PMLR; 2021. p. 478–86.
99. Long Y, She X, Mukhopadhyay S. Hybridnet: integrating model-based and data-driven learning to predict evolution of dynamical systems. In: Conference on robot learning. PMLR; 2018. p. 551–60.
100. Ensinger K, Ziesche S, Rakitsch B, Tiemann M, Trimpe S. Combining slow and fast: complementary filtering for dynamics learning. arXiv preprint. 2023. arXiv:2302.13754.
101. Trimpe S, D'Andrea R. Accelerometer-based tilt estimation of a rigid body with only rotational degrees of freedom. In: 2010 IEEE international conference on robotics and automation. New York: IEEE; 2010. p. 2630–6.

Publisher's Note