Received: 21 October 2020 | Revised: 20 January 2021 | Accepted: 25 January 2021 | Published on: 4 March 2021

DOI: 10.1002/gamm.202100002

ORIGINAL PAPER

A perspective on machine learning methods in turbulence modeling

Andrea Beck[1] | **Marius Kurz**[2]

[1]Laboratory of Fluid Dynamics and Technical Flows, University of Magdeburg "Otto von Guericke", Magdeburg, Germany

[2]Institute of Aerodynamics and Gas Dynamics, University of Stuttgart, Stuttgart, Germany

Correspondence
Andrea Beck, Laboratory of Fluid Dynamics and Technical Flows, University of Magdeburg "Otto von Guericke", Universitätsplatz 2, 39106 Magdeburg, Germany. Email: mail@beck.aero

Abstract

This work presents a review of the current state of research in data-driven turbulence closure modeling. It offers a perspective on the challenges and open issues but also on the advantages and promises of machine learning (ML) methods applied to parameter estimation, model identification, closure term reconstruction, and beyond, mostly from the perspective of large Eddy simulation and related techniques. We stress that consistency of the training data, the model, the underlying physics, and the discretization is a key issue that needs to be considered for a successful ML-augmented modeling strategy. In order to make the discussion useful for non-experts in either field, we introduce both the modeling problem in turbulence as well as the prominent ML paradigms and methods in a concise and self-consistent manner. In this study, we present a survey of the current data-driven model concepts and methods, highlight important developments, and put them into the context of the discussed challenges.

KEYWORDS

closure models, LES, machine learning, RANS, turbulence simulation

1 | INTRODUCTION

With the recently exploding interest in machine learning (ML), artificial intelligence (AI), Big Data, and generally data-driven methods for a wide range of applications, it is not surprising that these approaches have also found their way into the field of fluid mechanics [9], for example, for optimization and control [75], flow field reconstruction from experimental data [38], or shock capturing for numerical methods [6,62]. An area with a particularly high number of recently published research that uses ML methods is turbulence, its simulation, and its modeling. The reasons for this marriage between these two fields are numerous, and some will be explored in this article. The most natural one might be—mostly from the perspective of computational fluid dynamics (CFD), but also from the experimentalist's view—is that, when simulating or measuring turbulence, one naturally encounters large amounts of high-dimensional data from which to extract rather low-dimensional information, for example, the drag coefficient of an immersed body. This generation of knowledge (or at least models and ideas about knowledge) from a large body of data is at the core of both understanding turbulence and ML—and of course many other fields. At least from the authors' perspective, there often seems to exist a mismatch between the tremendous amount of data from simulations or measurements that we have on turbulence, and the tangible results we can actually deduce from that. Here is where ML and Big Data might help, for at least in theory, they can make more effective use of the data we have.

In this article, we intend to give an overview of recent research efforts into turbulence modeling with ML methods or, more precisely, data-driven closure models and methods. It is not meant to replace the already existing valuable review articles on this matter, for example, [17,55,58], but to provide a complementary perspective with a particular focus on large Eddy simulation (LES) methods, although many of the challenges and chances of ML in this field also directly transfer to the Reynolds averaged Navier-Stokes (RANS) system. We have attempted to present both the simulative side and the ML side in a concise and self-contained manner with a focus on the basic principles, so that the text may be useful to nonspecialists as well, and provide a first starting point from which to venture further. The focus of this review is thus not so much on specific examples of how ML methods can augment turbulence modeling (a very daunting task given the current surge in publications), but on the challenges, chances, and capabilities of these methods.

Before discussing the combination of ML and turbulence models, we start in Section 1.1 with a brief introduction to turbulence and the two prevalent simulation strategies, namely, RANS and LES. The discussion is focused on the formal aspects of defining the governing equations and the associated solution and on the factors that influence them. This is somewhat of a shift from the usual introduction to LES, but in the light of data-driven modeling, *data-consistency* (during training) and *model-data-consistency* (during inference) are important aspects. Highlighting the different factors that influence the LES solution and its modeling is not only helpful here but also motivates the use of ML methods in this complex, high-dimensional, and nonlinear situation. With this introduction in place, we will then give a perspective on the challenges and possibilities of ML and turbulence models in Section 1.2. Here, we discuss what types of challenges arise in turbulence modeling, which are mostly specific to the ML approach, but also give reasons to be optimistic that data-driven turbulence simulations can augment models and simulation approaches derived from first principles. Before discussing the recent literature and presenting concrete examples, we first review the most basic ML paradigms and methods in Section 2. Here, as in the rest of the manuscript, we mostly focus on *supervised* learning approaches. Also, since neural networks are the most commonly used method in this context, the discussion will also lean towards them and related methods. In Section3, we present a possible hierarchy of modeling approaches, from data-based parameter estimation to the full replacement of the original partial differential equation (PDE). We focus on the general idea behind the methods and on their relationship to the challenges and chances identified in Section 1.2. Due to the dynamic research landscape and the high frequency of new publications and ideas, no converged state of accepted best practices has emerged yet, so our goal here is not to rank the proposed methods but to equip the reader with the conceptual knowledge of the current state of the art. Section 4 concludes with some comments and ideas for future developments.

1.1 | Turbulence and turbulence modeling

Turbulence is a state of fluid motion that is prevalent in nature and in many technical applications. The blood stream in the bodies of mammals can become turbulent, and the resulting increased stresses can lead to the onset of serious medical conditions. Transition and turbulence govern the flow through turbo-machines and significantly influence their efficiency and reliability. Finally, turbulent drag and heat transfer not only strongly influence the fuel consumption of subsonic passenger aircrafts but also pose a great safety challenge for supersonic vehicles. This list is, of course, far from exhaustive but serves to highlight the importance of understanding, analyzing, and predicting turbulent flows.

As opposed to laminar flow, turbulence is characterized by a broad range of interacting temporal and spatial scales. This is an expression and a result of the nonlinearity of the governing equations, which leads to rich dynamics and a high sensitivity to initial and boundary conditions. Turbulence shares this feature with other chaotic systems, and they all have to cope with a number of challenges in experimental investigations, mathematical analysis and also numerical modeling. Turbulence is typically characterized by the Reynolds (Re) number, a dimensionless ratio of the actions of inertia and the counteracting viscous effects. The viscous 1D Burgers' equation for a variable $u(x,t)$ representing a velocity serves as an incomplete but useful model for the turbulent cascade and highlights the influence of the Re number:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial u^2}{\partial x} - \frac{1}{Re}\frac{\partial^2 u}{\partial x^2} = 0. \tag{1}$$

Assuming an initial solution at time $t=0$ with amplitude $\hat{u}$ and a single wavenumber k as $u(x,0) = \hat{u}\ sin(kx)$, the initial time derivative becomes

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = -\frac{1}{2}\hat{u}^2\ \sin(2kx) - \frac{\hat{u}}{Re}k^2 \sin(kx). \tag{2}$$

Equation (2) already reveals a number of interesting insights. First of all, it is the nonlinear convective term that drives the scale cascade by producing higher-frequency contributions ($2kx$), while the viscous term damps the initial waveform (kx). The magnitude of this damping is a function of k^2/Re, that is, regardless of Re, this term will dominate in the limit $k \to \infty$[1], a fact that motivates the choice of dissipative closure models. For small k and/or large Re, however, the evolution of Equation (1) is dominated by the inviscid dynamics. Note that the Burgers' equation should only serve as a limited example for the incompressible NSEs, in which the quadratic nonlinearity in the convective fluxes is the most important term for the turbulent dynamics.

For the full Navier-Stokes system, a famous result obtained by balancing production and dissipation derived by Kolmogorov [31] states that the ratio of the smallest to the largest occurring scale is given by $\frac{\eta}{l_0} \propto Re^{-3/4}$. Taking this estimate to three spatial dimensions and accounting for the corresponding smallest temporal scales lead to the estimate for a lower bound on the number of degrees of freedom $\mathcal{N}$ required for a direct numerical simulation (DNS) of free turbulence [64]:

$$\mathcal{N} \propto n_{ppw}^4 Re^3, \tag{3}$$

where n_{ppw} is a measure of the accuracy of the discretization scheme. More important for our discussion here, however, is the scaling of the Reynolds number, which can make a full resolution of all occurring scales unbearably expensive. Thus, instead of solving the full governing equations, in the vast majority of practical applications, a coarse-scale formulation of the NSEs is sought. These governing equations, that is, the LES or RANS equations describing the coarse-scale dynamics and solution, are found as follows: we start from a general form of a conservation law for a vector of conserved quantities $U(x,t)$, for example, the NSEs, in the form

$$R(U) = 0. \tag{4}$$

Here, $R()$ is the temporal and spatial PDE operator. We now introduce a low-pass filter function $\overline{(\cdot)}$ with linear filter kernel and a filter width $\overline{\Delta}$, such that we obtain a separation into coarse and small scales of a quantity ϕ as $\overline{\phi}$ and $\phi' = \phi - \overline{\phi}$, respectively. Applying this operation to Equation (4) yields, under some assumptions, the coarse-scale governing equations for $\bar{U}$:

$$R(\bar{U}) = \underbrace{R(\bar{U}) - \overline{R(U)}}_{=:M \approx \hat{M}(\bar{U},(\cdot))}. \tag{5}$$

Note that Equation (5) incorporates both the RANS and LES methods, the difference being the choice of the filter. In RANS, a temporal averaging filter is chosen, returning the time-averaged or mean solution field. In LES, filtering is usually done in physical space, and a significant part of the temporal and spatial dynamics remains present. In both cases, the nonlinearity of U in R prevents the commutation of filter and PDE operator and introduces the *closure problem*: although the LHS of Equation (5) is now formulated in coarse-scale quantities only, the RHS still contains a term that requires modeling: $\overline{R(U)}$. This model term $\hat{M}$ as an approximation of M must be formulated in coarse-scale quantities $\bar{U}$ (and possibly free parameters) only; otherwise, the advantage of this approach would be lost. If, of course, $\overline{R(U)}$ was known exactly (eg, from a precursor DNS), then $M = \hat{M}$ would recover the exact closure term. This approach is termed "perfect LES" and is used as a consistent framework of model development [35], but as it requires prior full-scale information, it is not generalizable. Note also that, in order to solve for $\bar{U}$ numerically, R must be replaced by its discrete equivalents. This entails a number of new difficulties, which will be discussed later.

Clearly, the introduction of the model $\hat{M}$ infuses uncertainties and errors into the solution process. Still, the rationale and justification for solving the coarse-scale equations instead of the original ones are given by

- Universality of the modeled regime: The filtered-out small scales are generally less affected by the boundary conditions and thus show a much more isotropic behavior than the large scales, which makes them amenable to modeling [4]. In addition, the small scales carry only a fraction of the kinetic energy, whereas the large (resolved) scales are dominating the dynamics of the flow.

[1]The inviscid counterpart to the Navier-Stokes equations (NSEs), the Euler equations, lack the limiting viscosity. There are some suggestions that the Euler system might develop singularities and thus contain different dynamics than the NSEs, but this is not considered here.

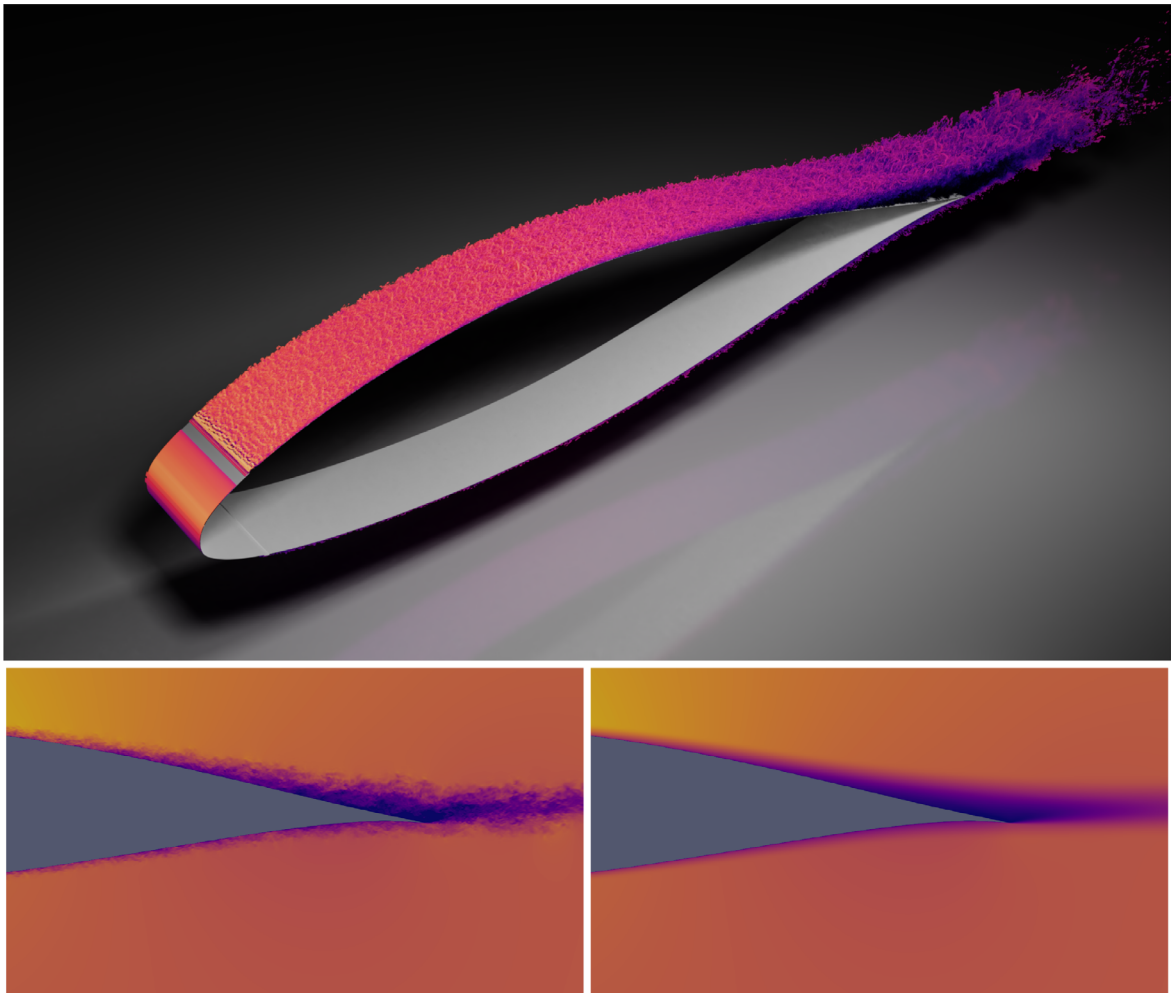

FIGURE 1 Flow over an NACA 64418 airfoil at $Re_c = 10^6$ and Mach number $Ma = 0.2$. Top row: underresolved direct numerical simulation, shown are isocontours of the Q-vortex criterion, colored by velocity magnitude. Lower row: contours of vorticity magnitude, left: flow field from large Eddy simulation, right: time-averaged (Reynolds averaged Navier-Stokes) results

- Structural stability of the coarse-scale solution: While single realizations of a turbulent field are highly sensitive to the initial and boundary conditions and small-scale perturbations, the general structure of the solution is largely invariant to them. It can be shown that the truncated NSEs have attractors that share a number of properties to those of the continuous equations. This is the reason why the average and higher moments of the LES and DNS solutions are often in good agreement [76].
- Computational necessity: Since Equation (5) contains only coarse-scale quantities (under the assumptions that the model regularizes the solution sufficiently and aliasing is accounted for), it is computationally considerably cheaper and avoids the requirement in Equation (3). In fact, the computational costs can be specified a priori (within certain limits given by physical requirements) by choosing the filter width $\overline{\Delta}$.

In order to visualize the model challenge and the RANS and LES solution approaches, Figure 1 shows an example of the two different coarse-scale formulations applied to an airfoil flow at $Re = 10^6$. In the top figure, an underresolved DNS, the multiscale character of the turbulent boundary layer, is clearly visible. In the lower row, an LES (left) and a RANS (right) solution of the flow field in the vicinity of the trailing edge are shown. In both cases, the fine-scale information is lost through the filtering process (all fluctuations are discarded in RANS), and both formulations require an additional model; however, the computational cost is reduced significantly as opposed to the full scale resolution.

While, so far, RANS and LES methods have shared the formalism, significant differences and challenges occur at closer inspection. First, while the filter in RANS is always defined as a temporal average, and often, the steady version of Equation (5) is solved (we exclude the unsteady RANS approaches here), there is a lot more ambiguity in LES as one

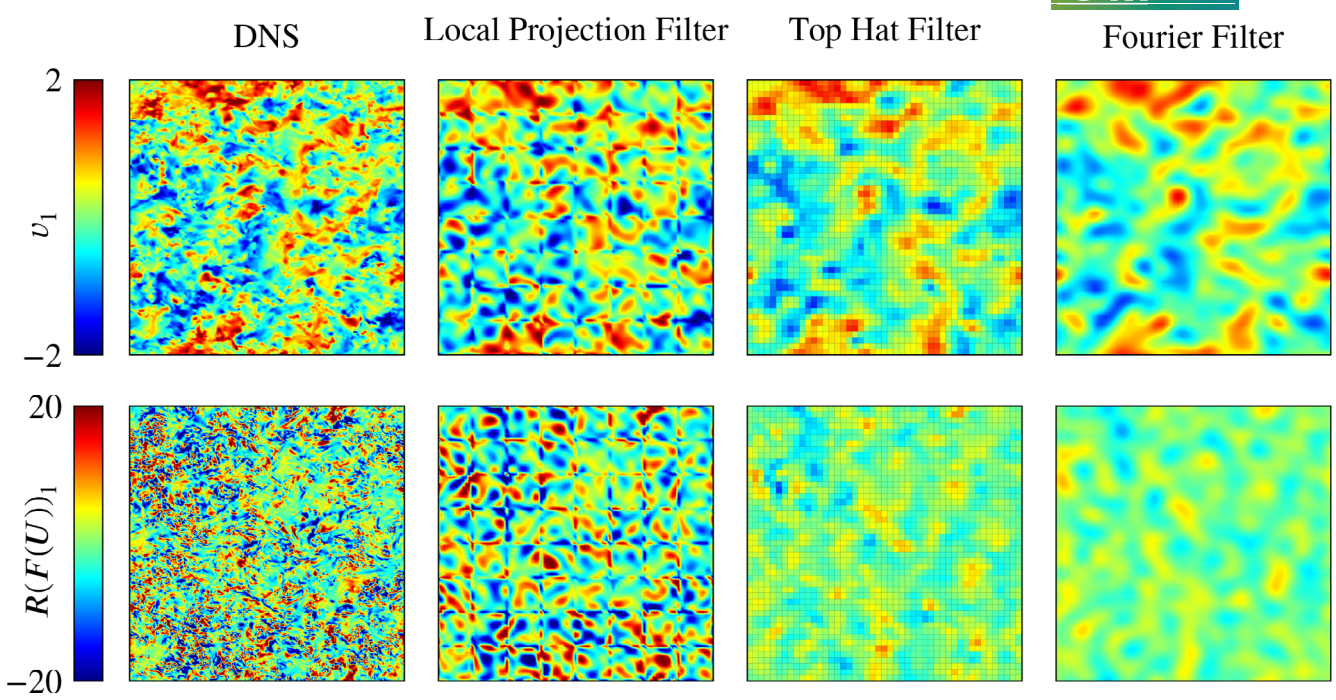

FIGURE 2 Two-dimensional slices of the full and the filtered x-velocity field (*top*) and $\overline{R(U)}$ flux divergence (*bottom*, $R(U)$ for direct numerical simulation [DNS]) from a decaying homogeneous isotropic turbulence flow. Shown is the field solution (from left to right) for the DNS, the local projection filter, the local top hat filter, and the global Fourier filter. Reproduced from [33]

is essentially free to choose any meaningful filter with specific properties. One such example is the option to choose Favre-filtering for the compressible NSE. Another choice concerning the filter function directly are box filters, modal filters, discrete vs continuous filters, or even discretization-induced filters. A subtle but important point for model development is that any choice of filter always brings with it its own associated closure term M, which is clearly a function of the chosen filter [59]. Figure 2 underlines this property by showing results of the simulation of decaying homogeneous isotropic turbulence (DHIT), that is, a canonical turbulent flow in a periodic box. The upper row contains the results for a velocity component $v_1 \in \bar{U}$, filtered from the DNS result (left column) with three different filters (second to fourth column) typical for LES. The bottom row shows the associated contributions to the closure $\overline{R(U)}$ for the corresponding filter. The influence of the filter on the closure terms is clearly visible and stresses the role of the filter choice in LES. Additional complexities in the formal LES equations arise from (a) the nonhomogeneity of the filter (even if the filter kernel itself is shift-invariant, issues with grid stretching or at boundaries can occur) and (b) the discretization operator and its associated errors. The action of the numerical schemes becomes the most important factor in so-called implicitly filtered LES. Here, the filter $(\cdot)$ is replaced by the discretization, which by design reduces the number of degrees of freedom of the full solution U to a discretized version $\tilde{U}$. Note that, while the $(\tilde{\cdot})$ operator is often also called a filter (and the LES formulation is called "implicitly filtered" or "discretization/grid filtered"), its properties differ from $(\bar{\cdot})$ in important aspects: numerical errors occur, which are particularly active on the marginally resolved scales, and the discretization operators themselves can include nonlinearities (eg, for stabilization). This results in two issues worth noting: first, the actual solution $\tilde{U}$ is not known a priori and cannot be determined from DNS data through filtering (as is true for $\bar{U}$). Second, the choice of the discretization determines the closure terms (see also Figure 2) and the discretization errors must become a part of their model. For a recent and more thorough discussion of these intricacies of LES, we refer the reader to [33,49]. Fortunately, the situation is less complex for the RANS method. Here, because only the mean solution is sought, the discretization effects and their interaction with the closure play a much lesser role, which allows the RANS modeling efforts to focus on "physics only," whereas for grid-filtered LES, both numerical and physical aspects must be considered.

Summarizing this discussion, the closure problem of turbulence can be described formally rather easily at first glance, but the resulting terms to be modeled can be influenced by a number of choices beyond the physical effects. In particular, for grid-filtered LES, the numerical scheme, its errors, and its induced scale filter contribute strongly to the exact closure terms. Thus, before attempting a data-based modeling approach, a careful appreciation and definition of the governing equations, their solution, and the closure terms is necessary to ensure data consistency.

Now returning to Equation (5), finding a model $\hat{M} \approx M$ is the next task. Here, in general, two approaches are common:

$$\hat{m_1} := \min_{f(\theta),\theta} \| M - \hat{M}(\theta, f(\theta)) \| \quad \text{or} \quad \hat{m_2} := \min_{f(\theta),\theta} \| J(M) - J(\hat{M}(\theta, f(\theta))) \|. \tag{6}$$

In the first case, a best approximation to M is sought directly as a function of some relationship $f(\theta)$ with parameters θ, where both f and θ can themselves be functions of the coarse scale fields. In the second case, the model is designed to approximate some derived quantity that is dependent on the model, for example, an interscale energy flux or a spectrum. The most common form of models both for LES and RANS are from the $\hat{m_2}$ family and try to match the dissipation rate of the kinetic energy. Two major strategies exist for proposing the function f in Equation (6): it can be posited explicitly, that is, some functional form based on physical or mathematical considerations is assumed (so-called explicit modeling), or the discretization error (possibly specifically modified) can replace f (implicit modeling). In any case, due to the complexities in both the optimization process of Equation (6) and the interdependencies discussed above, the resulting models will be approximate and often only effective and accurate under circumstances which are comparable to the one present during their inception. A salient example of this is the fact that the constant for Smagorinsky's model has to be retuned for different discretizations and resolutions.

This section has served to highlight the challenges involved in model development and might help to explain why a converged state of the art in LES models is still lacking. In the next section, we give some ideas on why ML methods might help improve upon this situation.

1.2 | Turbulence modeling and ML

Before discussing the challenges and opportunities in ML and turbulence modeling, we give a brief and possibly incomplete working definition of ML for the purpose of this paper without going into the details of the different methods (cf. Section 2):

> ML is a subfield of AI. It can be defined as a set of methods and algorithms for estimating the relationship between some inputs and outputs with the help of a number of trainable parameters. The learning of these parameters, that is, their optimization with respect to a given metric, is achieved in an iterative manner by comparing the model predictions against ground truth data or evaluation of the model performance.

From this definition, it becomes instantaneously clear that ML algorithms are *not* the recommended method of choice if the relationship between inputs and outputs is (a) known (eg, given analytically) and easy to evaluate. In this case, ML methods typically suffer from two drawbacks: (1) They are approximations to the analytical function only, even with possibly very high precision and (2) they are not as efficient to evaluate as the analytical function. ML methods are also not recommended if (b) the relationship between inputs and outputs is unknown but simple: for example, if a simple linear regression or quadratic curve fit will give sufficiently accurate results, then advanced ML algorithms will work but are unnecessary.[2] Lastly and trivially, ML methods are not useful if (c) input and output are not correlated. However, this statement must be accepted with some caution; even if there is only a negligible correlation between feature A and target Y as well as feature B and target Y, a feature made up of a nonlinear combination of A and B may result in a high correlation to Y. In fact, this *feature selection capability* is a particular strength of some of the ML methods.

In the case of the turbulence modeling challenge outlined above, neither of these hindrances is true however. There is no closed known analytical formulation (on coarse grid data only) to close the equations; simple models work acceptably well but can be made to fail easily, and we also know that coarse grid quantities correlate with the subgrid terms (in fact, the test filtering in the dynamic procedure of the Smagorinsky's model is predicated on this). Therefore, it is at least reasonable to expect that ML methods are viable candidates to help in turbulence model building.

Thus, a general application of an ML method to turbulence modeling can be formulated by specifying that the functional form f and/or its parameters θ in Equation (6) are to be found by an ML method through training on data. As will also be discussed later, the functions that can be approximated by ML methods are quite rich, and finding an optimal

[2]Linear regression can, of course, be formulated as an ML problem, and a least-squares fit is in fact the smallest building block of a neural network. However, in this context, we consider ML methods to include only more complex approximators like multilayered neural networks or support vector machines (SVMs).

function space that best represents the data is one of their strong points—thus, it is important to stress that ML methods cannot just fit parameters to some given basis but also find the basis itself from data. This is somewhat orthogonal to the previously described modeling concept of explicit and implicit closure modeling in that no a priori functional form of the closure needs to be postulated. The training process of the ML model is then the optimizing w.r.t. a given error norm based on the data available during the (offline) training phase. The application of an optimized model in a predictive manner (often called inference or model deployment) is then just an evaluation on a new data point. If the model is able to generalize well, it will give useful predictions for this unseen data. Here, careful observation of model data consistency is a key factor: the data during training (which implicitly defines the model) and inference must remain consistent.

With this brief conceptual overview of ML and turbulence modeling, we will now discuss some of the challenges and possible drawbacks but also chances of ML methods in turbulence. These lists are far from complete and are meant to raise general issues that may or may not be applicable to the specific case considered. It is expected that the surging research efforts in this field will help to answer these questions in the next years.

1.2.1 | Challenges for ML-augmented turbulence modeling

In this section, we try to summarize the challenges and difficulties ML-augmented turbulence models must tackle. Not all of the issues are particular to ML, but casting them in an ML context is useful for the purpose of this discussion.

The need for data:ML methods are, at their core, optimization algorithms that find an optimal fit of functions to data. What makes them so expressive is that the functions typically are high-dimensional and nonlinear. This makes the optimization within the feature space costly, as the typical algorithms employed here are variants of a gradient-based, iterative progression to the minimum. The rate of convergence is thus limited and a function of the step size (called learning rate in ML), and there is no guarantee a global minimum has been found. Often in practice, several optimizations are run parallel to check if the found minimum is stable w.r.t. the hyperparameters and starting points. Thus, training an ML method requires significant amounts of data or a cheap repetition of the data generation mechanism. As both experiments and DNS, which are the two "data generation" choices for turbulence, are rather costly, this poses a significant challenge.[3] Mining data from existing databases is also not straightforward due to the issues of data consistency and data model consistency. In addition to the availability and suitability of data, the question of cost-effectiveness comes into play: it must be assured that the generation of training data constitutes a responsible use of computing time and experimental resources and that the outcome (eg, an improved turbulence model) justifies the investment.

Consistent data and model: Defining target quantities and input data for an ML method to learn from is not a straightforward task. As discussed above, while the solution and associated filter kernel to a RANS formulation is rather clear, the situation for practical, discretization-filtered LES is a lot more ambiguous. This also translates to any derived LES quantity computed from the solution field or through the application of the implicit filtering. This makes both the target of an ML method as well as its inputs dependent on the chosen LES formulation and discretization. In addition, the ML models must be made robust against training-inference inconsistency: even if great care is taken to apply a model in a setting that corresponds to the one it was trained in, noisy data, the probabilistic parts of the ML method, error accumulation, and the nonlinear nature of the underlying turbulent process can lead to an input distribution that is vastly different from the one experienced in training. Even if the model predictions themselves are stable against disturbed inputs, their overall effect can be troublesome. For example, if the ML model predicts parameters in a closure model, and the predictions themselves are reasonable and within range, the term the parameters are applied to can diverge from the training situation (eg, a velocity gradient from training [DNS data] and become vastly different from the one during prediction [RANS/LES solution]). Another example is the influence of discretization operators discussed above. Thus, in general, for an ML model to be successful, training and inference data must be consistent, the inference problem must be well posed, and the ML models must be made robust against unavoidable uncertainties. This is likely a crucial point in determining whether the advantages of ML-augmented models can be brought to bear.

[3]We note that there are attempts to synthesize turbulent fields through generative models, for example, in [19,48], which potentially ameliorates this situation.

Inclusive optimization: Along the same vein, closure models must be considered on the level of the system of equations to be solved, not on a more fine-grained level. While this is true for any modeling approach besides ML-based attempts, it is worth pointing out here. This means that the overall closure must be optimized (also in the context of the discretization, see above) and that just improving the prediction of certain effects or terms does not necessarily lead to better models. It is instead the interplay of the different terms that must be considered. One example of this is the dynamics in turbulent boundary layers, where models that respect the overall balance of the physical effects have been shown to be superior to approaches that prefer the modeling of single contributions [36].

Physical constraints: Incorporating physical and mathematical prior information and constraints into the models is likely important as it not only helps with data model consistency or even consistency of the model to the governing PDEs but also makes the resulting models more robust and accurate and easier to train. In addition, this information can help to reduce the amount of training data significantly by confining the parameter space to a sensible subspace. There are a number of ways to enforce constraints in the ML process: the most obvious one is through the selection of training samples that share a given property, for example, periodicity or positivity. This informs the ML methods of their existence implicitly, and helps to construct surrogates that also obey them; however, this is not guaranteed during inference or for extrapolated inputs. An additional measure is to explicitly include the required constraint like symmetry or realizability into the loss function and optimize accordingly. This usually leads to increased stability of the model but again does not enforce the fulfillment of the constraint. So, while constraining the input or the model output is the current state of the art, there is research underway to develop ML approximations with guaranteed properties, for example, [66]. However, the relative importance of the respective constraints and the best way to enforce them remains an open challenge—not just for ML-based methods. For more information and a list of invariants and constraints, we refer the reader to [51,73].

Generalizability, interpretability, and convergence: As with all models, some basic questions must be addressed to ML-based formulations, and their properties must be understood. Among these properties, generalizability is certainly one of the most desirable. This not only includes the applicability of a respective model outside of its training regime (ie, at a Reynolds number not trained on) but also a consistent fallback mechanism in cases where the model is likely to fail. Even before that, a means of measuring confidence in the model prediction should accompany any model—in its simplest form, this could be an estimate of the position of the input data in feature space and a comparison against the statistics gathered during training. However, it is still unclear if we can expect more from ML in this context than from classical turbulence models, which all have their points of failure and generalization issues. The simplest case of generalization capability of an ML-augmented model should be at both limits of modeling: it should turn off in laminar flow and at the DNS resolution. A related but subtler property, particularly in conjunction with a discretization, is the question of the convergence of the model prediction with increasing flow resolution. Interpretability is probably the most illusive property on the list, and one may argue that a correct model prediction is more important than an understandable one. However, as it is unlikely that ML-augmented models will be ultimately successful without infusing expert human knowledge, understanding the ML-decision making is a necessary condition. Some general work on explainable ML algorithms can be found, for example, in [16,87].

Algorithmic and hardware considerations: ML methods, in particular neural networks, strive on graphic processing units (GPUs) or even more specialized hardware (eg, tensor processing units, TPUs) and in their native software environment like the Tensorflow [1] or PyTorch [57] suites, with Python being the usual language in which the user interacts with the underlying computational kernels. Legacy LES and RANS codes are, on the other hand, often written in C or Fortran, mostly with a focus on CPU execution, although GPU or hybrid codes are on the rise. In any case, during a practical turbulence simulation enhanced by ML methods, both classes of algorithms have to run concurrently, either on the same hardware or on specialized systems between which communication takes place. How to find optimal hardware and software stacks and how to balance the load between the flow solver and the ML kernels are an open field of research. Even tackling this question is currently difficult, as the state of turbulence modeling and ML (or any combination of traditional CFD and ML) is so fluent. In fact, as discussed in Section 3, the role of ML in this field is yet undefined, and with this comes the uncertainty of "how much" ML will be incorporated into CFD. If only lightweight algorithms with very few parameters run locally, the question of how to incorporate them into a flow solver is dramatically different from the case when the full flow field itself or model terms are to be predicted by the ML methods. In the first limit, ML can be incorporated as an additional feature into the codes; in the second limit, new flow

solvers need to be designed around the ML algorithms themselves. Finding the best compromise here must be the next step after the possibilities and limits of ML for this application have been fully explored.

Efficiency and ease of use: Finally, from a practitioners point of view, the question that will likely determine the fate of ML-based models is: Is it worth it? A general observation is that not always the best model or approach finds widespread acceptance in the community, but rather the ones that are quite simple to understand and implement, robust to handle and computationally cheap. What might also make these models harder to adopt for nonspecialists is the fact that they are not describable in a few lines of algebra or code but instead as a computational graph and its parameters. This is, of course, a technical issue that can be solved, but it makes experimenting with the models much more cumbersome. So, while ML methods have shown to be rather mighty in terms of accuracy, the other aspects are open for debate.

1.2.2 | Opportunities for ML-augmented turbulence modeling

With the challenges identified in the previous section, we now aim to balance the discussion and give a perspective on the motivation and opportunities for ML-augmented turbulence models. In Section 3, we will also present some successful data-based turbulence models and describe their relationship to the challenges and opportunities.

A new paradigm: Although considerable efforts have been invested in the last decades into finding closure models that are universal and accurate, there is still no generally accepted *best* model. This is due to the complexities and interdependencies outlined above. All previous attempts at modeling based on proposing a functional form or a set of equations have shown that some form of data-assistance is necessary to make the model useful, typically through the introduction of parameters or the imposition of constraints (eg, limiters). This reveals that there is still some form of functional relationship or knowledge not incorporated in these models—some external information seems to be missing. Here again, ML methods can help by attacking the problem from an alternate direction: the external data are not an afterthought to fix parameters but an integral part of the model development itself. This is emphatically not meant as a replacement of mathematical theory or physical reasoning in the modeling process, but rather as an inclusion of an additional stream of information.

Feature extraction: While abstractions like the energy cascade, the universality of small scales, scaling behaviors or law-of-the-wall encapsulate the essence of the underlying mechanisms of turbulence and elucidate their effects, they can give a false sense of providing the full picture. Typically, physically motivated closure models attempt to exploit these known (or assumed) correlations. The most prominent example of this are eddy-viscosity based closures: since the effects of small scale fluctuations are diffusive *on average*, a model that shares this behavior *on average* is a good initial guess. Following Boussinesq, the subfilter terms are then modeled as proportional to the mean velocity gradients, and model improvement comes from tweaking the proportionality constant. ML can improve upon this situation not only by finding even better model constants but, more importantly, by identifying new correlations automatically (*feature extraction*) from data, in particular very high-dimensional data. Thus, we can expect not only better tuned existing models but also the discovery of a better functional basis for model building.

Flexibility: ML methods can be designed to deal with nonstationary and nonhomogeneous statistics, that is, they can approximate regime changes in space and time. This is highly interesting for turbulence as this could help develop models capable of predicting intermittency and extreme events more reliably.

Incorporating discretization effects: An underappreciated aspect of LES is that a good closure model is an ever-moving target. This is due to the fact that, with very few exceptions, LES for practical problems is based on an implicitly filtered formulation. Here, the discretization itself acts as a low-pass filter and thus defines its very specific (and typically inhomogeneous and anisotropic) closure terms. Thus, a well-tuned and investigated closure approach for a given discretization can fail significantly in another setting. This strong nonlinear interaction of model and numerical scheme adds additional dimensions to the closure problem—in a sense, this can be seen as the introduction of an additional nonlinearity to the problem. While this makes the theoretical approach to finding models exponentially more difficult, it should—at least conceptually—be naturally included in an ML approach.

Exploiting existing turbulence data: As mentioned above, ML algorithms are not computationally cheap. In fact, one may argue that data-fitting approaches or learning methods are per se data-hungry during training—at least

if the typical incremental learning processes are used. Although generating the training data from scratch through experiments or numerical simulation is possible, it is also prohibitively costly—after all, if the costs of providing enough samples to lead to a converged ML model are orders of magnitude larger than using established no-ML models in the first place, there is no justification for going the ML route. However, generating new data specifically for model training is likely unnecessary in the long run: a large treasure trove of flow data already exists, both from numerical and experimental sources, for example, in [39]. Mining these data, for example, cleaning and processing the data in a consistent manner before injecting them in the training cycle remains a challenge however. Here, the flexibility of ML approaches towards the input features and the inherent capability of nonlinear feature combination can be beneficial in reducing the requirements on the training data.

Arbitrary input features: ML methods are able to incorporate both temporal and spatial data naturally and can also deal with sparse data. Up to now, typical closure models are often exclusively based on exploitation of spatial relationships or spatial data—for example, a velocity gradient at a certain position and given point in time. This comes from the fact that (a) the scale production mechanism that is at the root of the closure problem is formulated as a divergence and describes a spatial transport process and (b) that the vast majority of scale-separation filtering in LES is formulated in physical space as well, with a notable exception being the work by Pruett et al [52]. Incorporating both dimensions into the modeling process is an attractive idea as turbulence is a nonlocal phenomenon.

Summarizing this discussion, ML-based methods are attractive tools that can help to extract more, often hidden, knowledge from data, and thus provide a greater flexibility in modeling. The promise of ML here lies in the incorporation of flexible, high-dimensional data and its natural use of nonlinear modeling assumptions. Still, as in any modeling process, a number of challenges exist that need to be tackled, and no converged state of the art has been reached.

After briefly introducing selected ML algorithms for context in the next section, we will then discuss some successful applications of ML methods to turbulence modeling in Section 3. This should not only represent a slice through the current state of the art but also show the diversity of the ideas and methods currently under investigation.

2 | ML ALGORITHMS

The terms *AI*, *ML*, and *deep learning* (DL) are often used synonymously due to the tremendous success of the latter two in the last decades, making clear distinctions and definitions beyond general statements like the one given in Section 1.2 elusive. In principle, ML is only one specific discipline of many in the broad realm of AI but is currently the most prominent one. A complementary field in AI is, for example, "Good Old Fashioned AI," which encompasses AI systems that can be written down in a symbolic, human-readable way [23]. They therefore build on explicit domain expertise incorporated into the systems. One of the most famous representatives of this field is the *expert systems*. Another famous example for AI without the use of ML is the DeepBlue system, which was able to beat the chess world champion at the time, Garry Kasparov, in 1997 by using AI [10] but without a hint of ML.

In contrast, ML algorithms are (at least to a certain degree) agnostic to the specific task they are used for. Rather, they are able to learn from experience and adapt themselves to (or *learn*) the specific task autonomously based on data. This is, of course, an idealized view since choosing a better suited ML algorithm over another for a specific task based on experience already incorporates some form of expertise. The ML approach obviously has some advantages: foremost, ML algorithms can be employed where no sufficient domain knowledge is available to build expert systems. Additionally, existing algorithms can be used for a variety of tasks that are similar in structure (eg, using a neural network that is trained on detecting visual edges in photographs can very easily be brought to recognize shock waves as sharp edges from flow field data [6]) and only have to be adapted slightly, while traditional AI algorithms developed might be useless for the new task.

The field of ML can be categorized according to several learning paradigms: the first is supervised learning (SL, cf. Section 2.1). The foundation of an SL task is always an assumed correlation between at least two quantities, an input and an output. It is then the ML algorithm's task to approximate this unknown, but probably present, functional connection between these quantities solely from sampled training data. In contrast, unsupervised learning (UL) tasks (cf. Section 2.2) do not need labeled data pairs. This learning paradigm is of more an exploratory nature and confines itself to identifying correlations and patterns inside data, as well as finding efficient low-dimensional representations. The last learning

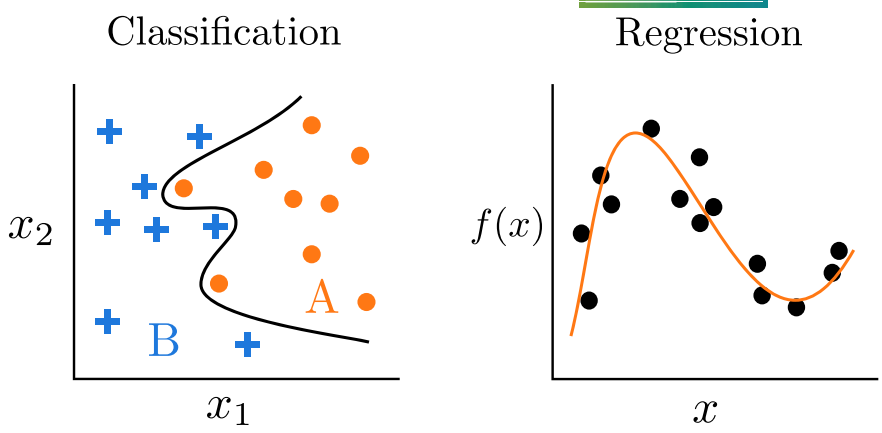

FIGURE 3 The two major paradigms of supervised learning: regression and classification. For a classification task, the machine learning algorithm needs to find a decision boundary, which separates the data into classes. New data points can then be classified by determining on which side they reside. For regression, the algorithm approximates the continuous functional relationship of the data

paradigm is reinforcement learning (RL), which does not rely on a dataset but rather employs algorithms which learn by interacting with an environment (cf. Section 2.3).

We are aware that the borders of these categories are fluent, and the assignment of applications and algorithms to the specific fields is more often than not ambiguous. Moreover, many methods do not fit comfortably into this framework but rather establish a small subcategory on their own. Nonetheless, this segmentation follows the usual classification and gives orientation for more in-depth excursions in ML. In the following discussion, we put a special emphasis on SL with neural networks, since they have become the de facto state of the art for many applications, among them turbulence modeling.

2.1 | Supervised learning

The basis of an SL task is a dataset of samples $\{X_i, Y_i\}$ from some unknown functional input-output relationship $Y = f(X)$. The task of the supervised ML algorithm is to approximate the true function $f(X)$ solely based on this (possibly noisy) dataset, which is commonly called training data. For continuous data, this task is called regression. Once a good approximation to the true function is found, the model can be evaluated for data points, which were originally not part of the training set (the *generalization*). If the data are discrete, that is, the algorithms have to assign the input data to a finite number of classes; this task is referred to as a classification task. The objective of the algorithm is then to find some (nonlinear) decision boundary in the input space, which separates the data points of the different classes from each other. Based on this decision boundary, new unknown data can be classified by determining on which side of the decision boundary they reside. Both approaches of SL are shown in Figure 3.

As discussed above, ML algorithms are agnostic to the underlying learning task and can be applied to a wide variety of problems. To accomplish this, ML models comprise free parameters, which have to be adapted for each learning task. The search of good model parameters for a specific learning task can be formulated as an optimization task with regard to some performance metric called *loss function* $\mathcal{L}(\hat{Y}, Y)$ or $\|Y - \hat{Y}\|$, see Equation (6). A common loss function for a regressor is, for example, the L_2-distance between Y and $\hat{Y}$. The optimization task can therefore be interpreted as finding a set of model parameters θ, for which the loss function reaches a minimum on the given training set $\{X_i, Y_i\}$:

$$\theta_{opt} = \arg\min_{\theta} \mathcal{L}\left(\theta \,|\{X_i, Y_i\}\right). \tag{7}$$

In the context of ML, this optimization process is commonly referred to as *learning* or *training*. A supervised ML algorithm therefore has two distinct phases: in the learning phase, the free model parameters are adapted to minimize the loss function for the given training dataset. In the inference phase, the trained model is evaluated on *unseen* data to obtain predictions for the corresponding unknown output value. In the following, we will shortly introduce some of the most common ML methods for SL: random forests, SVMs, and artificial neural networks (ANNs), with special emphasis on the latter. Regardless of the specific architecture chosen, the SL paradigm of finding an unknown, nonlinear mapping from input-output pairs $\{X_i, Y_i\}$ lends itself naturally to the task of parameter fitting (cf. Section 3.1) or subgrid flux reconstruction (cf. Section 3.2) in model development.

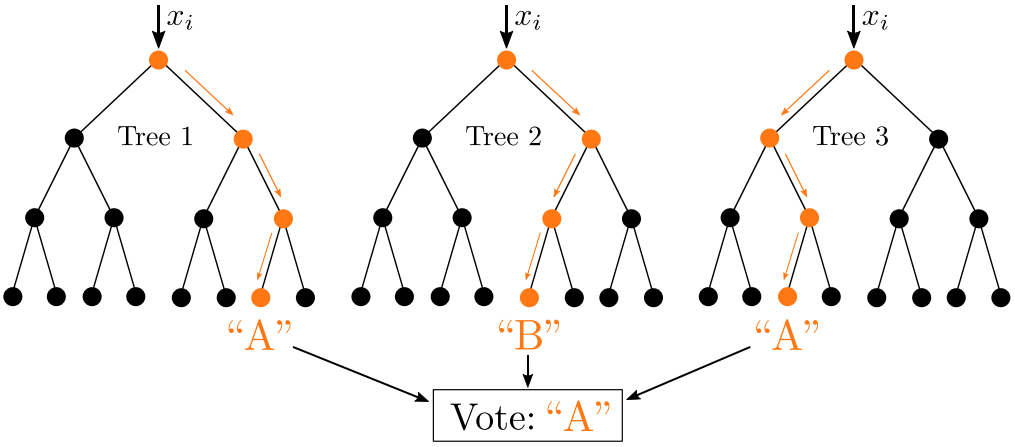

FIGURE 4 A schematic view of the random forest method. The depicted random forest model comprises three independently trained decision trees, which receive the same unseen input data x_i. The inferred class of x_i is then derived from the individual predictions by a majority vote

2.1.1 | Random forests

Decision trees are one of the most intuitive ML algorithms. This ML algorithm classifies data by dividing the training set recursively according to decision rules until each of the resulting subsets contains only data of one distinct class. Training the decision tree refers to finding a set of decision rules which divide the training set most efficiently. New unseen data can then be classified with the same decision rules and the model can propose an assumed class for the respective input data. Since it is not trivial to ensure that the found decision rules rely on intrinsic characteristics of the data and therefore to generalize well to unseen data, decision trees are prone to overfitting and are highly sensitive to changes in the training dataset. Therefore, more elaborate variants of decision trees are proposed in literature, with *random forests* as one of the most famous representatives. The method of random forests was originally proposed by Ho [26], who extended the single decision tree to an ensemble of individual decision trees to improve the generalization properties of the method. The main idea is that each individual decision tree is trained only on a random subset of the training data or receives only a randomly selected subset of the data features as input data. The overall class label for the data is then obtained by a majority vote of the individual predictions, which significantly enhances the generalization abilities of the decision tree method (cf. Figure 4). The idea to combine individual models to ensembles in order to enhance the overall prediction accuracy can also be employed for other algorithms, for example, neural networks. The decision tree and random forest algorithms can be easily extended to regression tasks by replacing the majority vote by averaging the individual predictions. In the field of turbulence simulation, random forests were, for example, applied by Wang et al [82], who trained the model on DNS data in order to correct the Reynolds stresses of RANS simulations.

2.1.2 | Support vector machines

The SVM algorithm strives to find a hyperplane that separates the space spanned by the input features such that only data points of one distinct class reside on each side of the hyperplane. New unseen data can then be classified by determining on which side of the separating hyperplane the respective data point resides. For most training datasets, however, the classes are obviously not linearly separable. Boser et al [8] therefore proposed to employ a nonlinear mapping of the input space in a high-dimensional (and possibly infinite-dimensional) feature space, in which the training data will always be linearly separable by a hyperplane, as indicated in Figure 5. These computations in high-dimensional space are generally expensive. The so-called *kernel trick* can be used to keep the computational cost at reasonable levels. By choosing an appropriate kernel function, computations in the high-dimensional feature space can be carried out implicitly by applying the kernel function, while the coordinates of the data points in the high-dimensional feature space never have to be computed explicitly. SVMs were primarily applied for supervised classification tasks, but can also be transferred to regression [72] and to unsupervised tasks, for example, clustering [7]. One of their most interesting features is the capability to enforce constraints directly through the kernel choice [70]. A more in-depth introduction to SVM can be found in [74]. In [41], SVMs are used together with decision trees and random forests to predict in which flow regions certain assumptions of RANS models become invalid, which can, for example, be used to dynamically adapt turbulence models based on the prevalent flow regime.

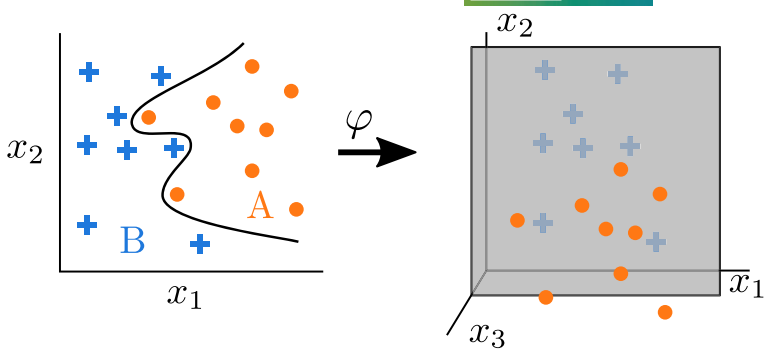

FIGURE 5 General idea of a high-dimensional feature space. Some nonlinear mapping φ is used to transform the two-dimensional input space into a three-dimensional feature space, in which the data becomes linearly separable. The inverse mapping then transforms the linear plane into the nonlinear decision boundary in the two-dimensional input space on the left

2.1.3 | Artificial neural networks

ANNs are the most widespread ML algorithm for many fields and applications, especially due to their generality and ease of training on GPUs. Their name is derived from the somewhat constructed resemblance to structures in mammalian brains [47,65], which was the initial motivation for the specific design of ANN as a combination of individual "neurons," shown in Figure 6. The classic feed-forward network is comprised of several concatenated layers. Each layer in turn consists of a number of neurons. The input layer of the network receives a data vector X, which is then successively passed through the *hidden layers*, which are not directly connected to the input or output, until the data are passed to the output layer. ANN with few hidden layers are referred to as *shallow* ANN, while networks with many hidden layers are called *deep* ANN. A typical design for the neurons is the perceptron as shown in Figure 6, which was originally formulated by Rosenblatt [65] a simple mathematical model for neural activity. Each perceptron receives the outputs of the previous layer X_i as inputs. These inputs are weighted by *weights* ω_i and summed up with an additional bias b. The output (also called activation) of the perceptron is then obtained by applying a nonlinear *activation function*. Commonly used activation functions are sigmoidal functions like the sigmoid ($g(x) = 1/(1+e^{-x})$) or the hyperbolic tangent ($g(x) = \tanh(x)$). More recently, the rectified linear unit (ReLU, $g(x) = \max\{x, 0\}$) became the state of the art for many applications. The computed output of the perceptron is then

$$Y = g\left(\sum_{i=1}^{N} \omega_i X_i + b\right). \tag{8}$$

The output Y is passed as input for the neurons in the succeeding layer. The described architecture of a feed-forward ANN with one or more hidden layers is commonly referred to as *multilayer perceptron* (MLP). The main objective during training is to determine optimal weights to obtain accurate predictions from the network. Thus, a neural network is a nested sequence of linear and nonlinear functions with variable parameters. It has been shown that ANNs are *universal approximators* and can approximate any continuous functional relationship between input and output quantities solely based on data [15,28,42].

A particularly successful branch of ML emerged over the last decade: DL. The method DL exploits that deeper networks can learn more complex relationships since each additional layer provides a (potentially) more abstract and complex representation of the data [37]. Training such deep and parameter-rich networks was made possible by a variety of circumstances. First, the method of backpropagation yields a very efficient way of optimizing the model's weights. The more recent success was also supported by the steadily increasing computational power, particularly by employing highly parallel GPUs for training [69]. The other major contributing factor was the availability of large datasets, which are needed to train deep networks, since they comprise an enormous amount of free parameters, and thus tend to overfit on small datasets [37]. These advancements fueled the research and development in the field of DL and caused it to become the most prominent and successful representative of ML for many applications.

A number of specialized ANN architectures exist, which are designed for a specific range of tasks. For multidimensional data, for example, images, convolutional neural networks (CNNs) are the current state of the art [37,69]. In contrast to the perceptron architecture, CNNs apply filter kernels to the input data in each layer, while the filter kernels themselves are learned during training. This generally results in a more sparse connection of succeeding layers and therefore in less trainable weights than in classic fully connected networks. This approach is especially suited for multidimensional data, which are not point-local, but rather have to be examined in context of their surroundings. Very deep ANN have

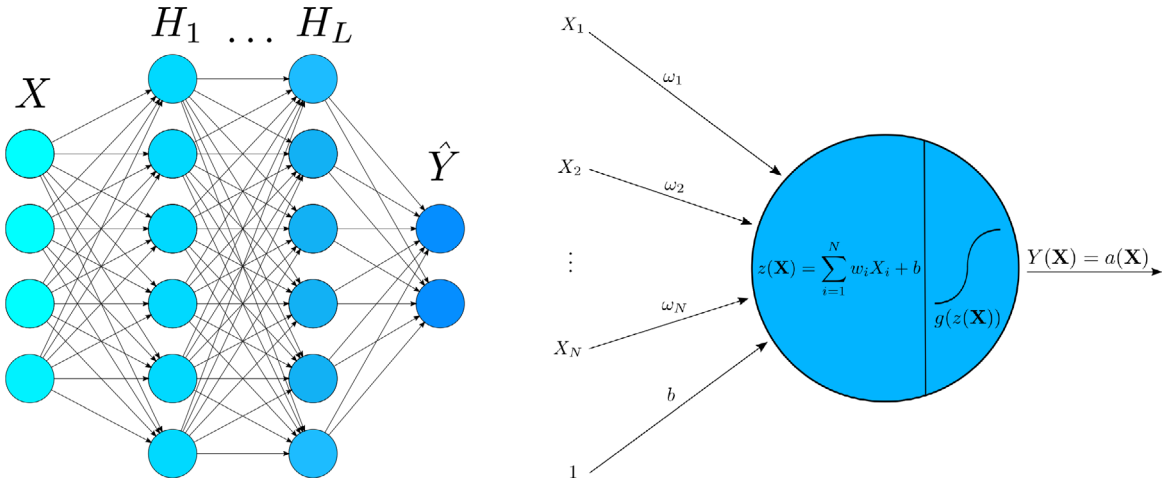

FIGURE 6 Shown on the left is the layout of a general feed-forward artificial neural network with two hidden layers, each comprised of several neurons. The design of a perceptron neuron is shown in more detail on the right

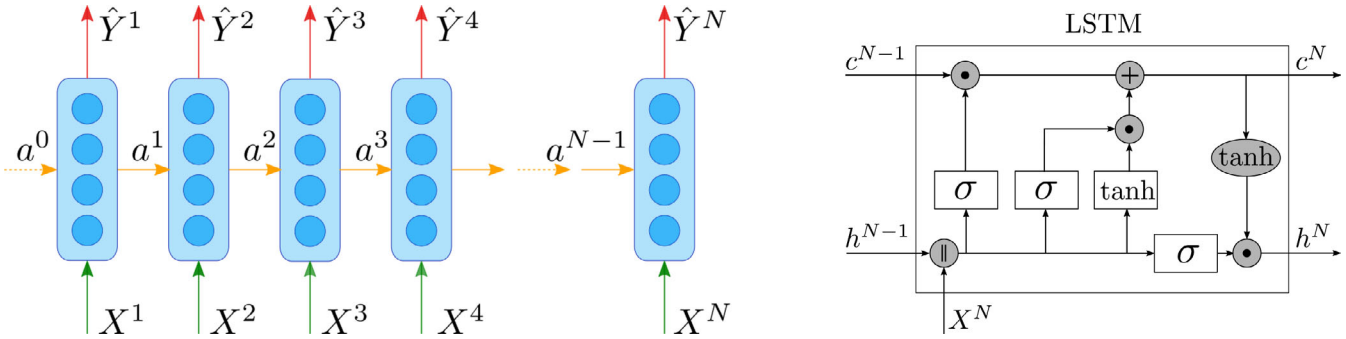

FIGURE 7 Shown on the left is the schematic of the recurrent neural network (RNN) approach when unrolled in time. Along with the current sample of the sequence X^N, the RNN receives information from the last timestep a^{N-1}, which allows it to retain temporal correlations from the sequential data. For the long short term memory (LSTM) cell on the right, the rectangular shapes indicate layer-wise operations according to Equation (8) with either a sigmoid (σ) or a hyperbolic tangent (tanh) as activation function. Round shapes imply element-wise operations, with $\|$ indicating concatenation. For the LSTM, the retained information a^N consists of the cell state c^N and the hidden state h^N, while the latter is also the layer output $\hat{Y}^N = h^N$

shown to be more difficult to train, since they suffer from the vanishing and exploding gradients problem during back-propagation; see, for example, [69]. To alleviate this restriction, He et al [24] proposed the residual neural network, which introduces skip connections, allowing information to shortcut the nonlinear layers. The CNN layers therefore only have to approximate the nonlinear fluctuations, while linear relationships are approximated quickly and stably. This allowed us to design and train very deep CNN successfully, thereby allowing to build more expressive and accurate CNN [24].

For sequential data, that is, where the ordering of the data points is important, recurrent neural networks (RNNs) are the established ANN architecture. A general schematic of the RNN principle is shown in Figure 7. The standard RNN works like the perceptron network shown in Figure 6 when unrolled in time. However, along the current input sample of the sequence X^N, the RNN also receives some information a^{N-1} from the last sample, which enables the RNN to memorize information from previous samples. How the network output Y^N and the memorized state a^N are computed depends on the specific RNN type. For long sequences, the RNN architecture leads to very deep networks in time, which again raises the issue of vanishing and exploding gradients during backpropagation, rendering the standard RNN unsuitable for long sequences [27]. More elaborate variants of RNN resolve this problem by introducing gating mechanisms which control the error transport during backpropagation. Two of the most common variants are the gated recurrent units (GRU) [13] and the long short term memory (LSTM) network [27], which is detailed in Figure 7. Due to their ease of trainability and access to open source frameworks, SL for turbulence modeling is predominantly done with ANN variants, see the examples in Section 3.

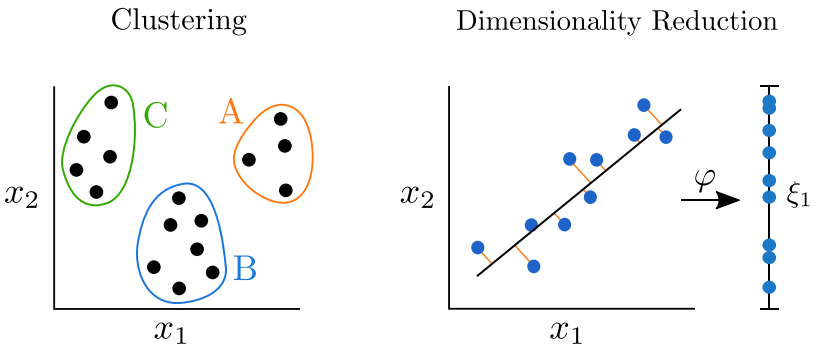

FIGURE 8 Two main applications of unsupervised learning: clustering and dimensionality reduction. For clustering, points with high similarity (ie, near each other) are grouped together in clusters to identify patterns in data. The field of dimensionality reduction tries to transfer high-dimensional data into efficient low-dimensional representations by finding appropriate transformations φ

2.2 | Unsupervised learning

In contrast to the SL paradigm, UL does not need labeled data, that is, it does generally not require any information on the exact solution of the problem. A common area of application for UL is clustering, which is the task of grouping data points based on their similarity, which in consequence allows us to find unknown correlations and patterns in datasets. In the following section, the k-means algorithm will be discussed exemplary as a common and straight-forward clustering algorithm. However, also clustering variants of several SL algorithms like SVM [7] can also be found in literature. Another major task of UL is the field of dimensionality reduction, which targets finding efficient low-dimensional representations of data, while minimizing the resulting information loss—this is, of course, closely related to finding a cluster of data with similar features. Both concepts are illustrated in Figure 8. Other applications of UL like probability density estimation are not reported here, since this exceeds the scope of this work. In the context of turbulence modeling and analysis of coarse-scale evolution, classical methods of dimensionality reduction like proper orthogonal decomposition (POD) or dynamic mode decomposition can be seen as UL algorithms. Their comparison to and augmentation by ML-focused UL algorithms is currently under way, for example, in [20]. Clustering methods applied to the local flow state can also become a helpful initial step in selecting appropriate adaptive closure models.

2.2.1 | k-Means

A popular and fairly simple clustering algorithm is the *k-means* method [43], for which a vast number of variations and extensions are proposed in literature. The first step of the algorithm is to obtain a first guess of the k partition centroids by, for example, choosing k data points at random as centroids or by more elaborate approaches [3]. Second, all data points are assigned to the respective nearest cluster centroid, and thus, each data point is assigned to one of the k clusters. In a third step, the cluster centroids can be updated by computing the center of all points assigned to the respective cluster. The second and third steps are then repeated until the algorithm converges and the clusters cease to change. Thus, the k-means algorithm eventually minimizes the average variance in the clusters. More elaborate variants of the k-means algorithm increase the method's accuracy and allow us to give guarantees and estimates of the method's performance as, for example, k-means++ [3]. An application of k-means clustering in conjunction with an NN for the prediction of an eddy viscosity for the RANS equations can be found in [91].

2.2.2 | Variational autoencoder and generative models

Autoencoders are a variant of ANN which is used for dimensionality reduction in a UL setting [25]. The autoencoder consists of two parts: first an encoder, which transforms the high-dimensional input X into a low-dimensional (*latent*) representation Z. This low-dimensional representation is then transformed back into the initial high-dimensional input space by the decoder, yielding $\hat{X}$. The general layout is depicted in Figure 9. The characteristic hourglass shape forces the autoencoder to find an effective low-dimensional representation of the input. The accuracy of the autoencoder can then be determined by computing the error of the network's input X and output $\hat{X}$ vector. Autoencoders are conceptually linked to the concepts of POD, principal component analysis and singular value decomposition, since all of these concepts describe efficient low-dimensional representations of data. An extension of this method to variational autoencoders

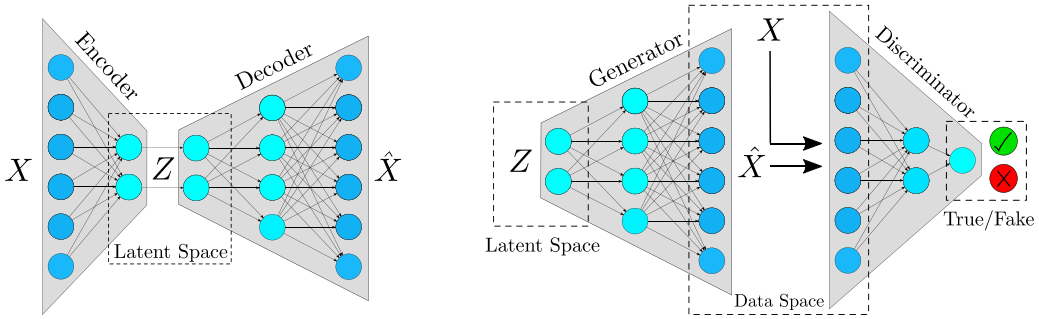

FIGURE 9 *Left:* architecture of an autoencoder. The encoder transforms the input data X into a low-dimensional latent representation Z, while the decoder transforms the data back into the input space $\hat{X} \approx X$. *Right:* general generative adversarial network architecture consisting of a generator and a discriminator. The former draws samples from the latent space and transforms them into approximate data samples $\hat{X}$, while the discriminator is trained to distinguish between the *fake* samples generated by the generator and the *true* data samples X from the dataset

(VAs) not only learns latent states Z, but a distribution given the input data. Thus, in inference mode, they are powerful tools for generative methods [30], with application to the generation of synthetic turbulence in [18,48]. An alternative variant of generative models are so-called generative adversarial networks (GANs [22]), in which two competing ML methods (typically ANNs) improve their prediction in an adversarial feedback loop. The goal of this architecture as shown in Figure 9 is to match the distribution encoded in the training samples and to be able to generate new samples that are indistinguishable (by the learned discriminator) from a true input. Both the VA and GAN can thus be interesting candidates for turbulence modeling as they learn a low-rank representation of high-dimensional data and can reproduce new statistically congruent samples. An application of GANs to subgrid flux modeling is discussed in Section 3.2.

2.3 | Reinforcement learning

In contrast to supervised and UL, the paradigm of RL is not restricted to learn solely from given data points, but to learn from interaction with an environment and therefore to learn from experience. The *agent* is an ML algorithm that performs actions to interact with an environment. These actions change the current state of the environment s_n in a stochastic manner, that is, the state change is generally not deterministic. Alongside the new state s_{n+1}, the agent receives a reward r_{n+1} based on a reward function, which has to be adapted to the specific learning task. The agent strives to adapt its actions a in order to maximize the potential future reward. Training the agent then means to find a decision policy $\pi(a|s)$, which determines the actions the agent should take to maximize its future reward, given the current state. The agent interacts with the environment across multiple *episodes* to gradually improve its policy π. The most straightforward approach is to explore all possible combinations of states s and actions a and memorize the expected reward in a table. For future episodes, the agent can then look up the current state s and perform the action, which promises the highest reward. This method is called Q-learning [83].

Obviously, this approach has limitations for large problems, since the memory cost and the time the agent needs to explore all possible combinations of states and actions are prohibitive for large problems. Instead, the approach of deep RL uses deep ANN to learn the policy for the RL algorithm, which was also used for AlphaGo, the first AI to beat a professional Go player [71]. A recent application of RL for turbulence modeling can be found in [50]. Here, a multi-agent reinforcement method is used to dynamically adapt the coefficients of a turbulence model for the HIT test case.

3 | EXAMPLES OF ML-AUGMENTED TURBULENCE MODELING

In the previous sections, we have focused on the challenges and chances of ML methods in turbulence modeling and given a brief introduction to selected ML algorithms. In the following, we will discuss some successful applications of these ML methods to RANS and LES problems in more detail. The majority of the following examples are from the SL methods discussed in Section 2.1. Among them, ANN and derived methods (cf. Section 2.1.3) dominate the published

literature. We will thus follow this trend here, with exceptions being the discussion on generative models in Section 3.2 and the SL/UL Physics-informed neural network (PINN) approach in Section 3.3. We note that this discussion is far from complete, but we will try to match the presented cases to some of the challenges outlined above and show how these issues are tackled by the respective authors. We will focus here almost exclusively on SL, mainly due to the much larger amount of research available than for other learning methods.

Recalling Section 1.2, the task of turbulence modeling is to find a model $\hat{M}$ for the effect of the subfilter terms on the resolved solution. For the purpose of this overview, one possible hierarchy for ML-augmented turbulence modeling shown in Equation (9) is thus *on which level* the modeling occurs: the first and most common approach is to replace the unclosed stresses by an a priori determined model function g with parameters θ and then fit or find the parameters through optimization approaches. Alternatively, the objective can be formulated in terms of a derived metric, see Equation (6). This approach has the clear advantages that (a) it can be used for model discrimination and that (b) existing turbulence models are naturally incorporated. Furthermore, the effect of the model on the governing equations is typically via a modified viscosity, which is often very easy to incorporate in existing schemes. The main point here is to stress that the structural form of the model is posited a priori. We summarize these approaches as parameter estimation problems. Opposed to this, the second level of turbulence modeling directly predicts the closure terms (either the stresses or the forces), that is, no functional form g is assumed, and it is an outcome of the optimization. The rationale behind this is to derive more universal closure models. The third level discussed here arguably leaves the realm of modeling—instead, it directly approximates the solution $\hat{u}$ of the full (turbulent) equations through ML methods. While these methods give up some of the rigorous results of classical approximation schemes for PDEs, their motivation comes mainly from the fact that they are generally mesh-free and can work with sparse data.

$$\begin{aligned} L1: \quad & \theta_{opt} := \arg\min_{\theta} \| M - g(\theta) \|, \\ L2: \quad & (\theta_{opt}, g_{opt}) := \arg\min_{g(\theta),\theta} \left\| M - \hat{M}(\theta, g(\theta)) \right\|, \\ L3: \quad & (\theta_{opt}, g_{opt}) := \arg\min_{g(\theta),\theta} \| u - \hat{u}(\theta, g(\theta)) \| + \left\| \mathcal{N}(\hat{u}(\theta, g(\theta))) \right\| \end{aligned} \tag{9}$$

3.1 | Parameter estimation

LES or RANS models based on physical or mathematical considerations are typically formulated as (collection of) functions or transport equations of coarse grid data and associated tunable parameters θ. These parameters might be obvious, for example, as a scaling factor of a viscosity, or more hidden in the details of the model and its implementation, for example, the choice of where and what to average over in the dynamic SmagorinskyŠs model. Since—when coming from a turbulence modeling background—parameter estimation tends to be seen as the obvious and most direct approach, a lot of literature focusing on this method with various flavors has been published in the last years. We do not aim to give a concise overview here but highlight some interesting cases belonging to this L1 family.

From the RANS modeling community, a number of publications on fitting the Reynolds stress tensor (or the discrepancy to existing models) based, for example, on a decomposition into its eigenvectors have been published [68,77,82]. Remarkable achievements have been the incorporation of field inversion techniques to reduce model-data inconsistencies and the direct embedding of constraints like Galilean invariances into the network [34,40,56]. An example from LES of a direct prediction of model coefficients is presented in [54],where the author learned the eddy viscosity from a turbulent database and showed good a priori and a posteriori LES results. The main feature of this work was the saving in computational time by a factor of 2 to 8 compared to an application of the dynamic Smagorinsky's model. One of the earliest applications in learning a viscosity coefficient is due to Sarghini et al, who used an MLP to construct a more computationally efficient deep ANN representation of Bardina's scale similarity model [67]. For compressible turbulence, the authors in [85] propose to fit the coefficients of Clark's closure model through an NN. One fact that is shared by the LES-focused approaches in literature so far is that the question of interaction of discretization and model is often discarded by focusing either on explicitly filtered LES or on global Fourier-based methods with a clear cutoff filter and homogeneous discretization properties. Such practical difficulties are not (yet) included in the modeling process. An interesting exception to this for 2D turbulence is proposed by [45]. Here, the implicit modeling method in LES is followed in the way that different numerical discretization operators for the inviscid fluxes (those with and without numerical dissipation) are chosen based on the classification by a neural network. The training data are obtained from a coarse-grained DNS

field and are distributed into bins corresponding to regions of no, negative, or positive eddy viscosity. While combining numerical operators in such a way locally brings with it its own challenges, the results shown by the authors are promising and offer an interesting way of informing implicitly filtered, implicitly modeled LES. Care must be taken here in the future, however, to ensure the consistency of the filter used to classify the DNS data and the dissipative characteristics of the numerical operators.

3.2 | Closure term estimation

The direct prediction of the closure terms instead of their modeling offers an alternative to the parameter estimation task in the previous section. Here, the unknown terms in Equation (6) are directly approximated by the ML algorithm—either as fluxes or as the forces themselves [5,21,79-81,84,86,90]. Important to stress, however, is that the mapping from the coarse to the fine field is nonunique, and thus, each coarse field is associated with a distribution of corresponding closure terms. This one-to-many mapping is naturally introduced by the information loss induced by the LES filter. For a given filter and thus coarse-grained solution, an infinite number of associated fine-scale fields and closure terms exist—thus, even exact closure terms should only be considered means of the ensemble [49].

The rationale for pursuing the direct estimation of the closure terms is 3-fold: (a) a direct closure avoids all modeling assumptions and is complete in the sense that it includes all the necessary and available information; (b) if the perfect closure is available, this opens the door for fitting and developing more accurate models; and (c) the research question of whether the closure terms can be recovered from coarse-scale data only is of great interest in itself. In terms of the discussion in Section 1.2, approaches from this group have some advantages: while they certainly need large amounts of data to train on, the target quantity is simply the filtered DNS data–no additional, derived quantities need to be available or computed. This reduces a strong source of data inconsistency and makes virtually any DNS data suitable for training. Also, finding the full closure terms instead of their models directly avoids any balancing issues between model terms. Furthermore, physical constraints can be enforced directly on the closure terms themselves without having to consider the interaction of different model terms in order to fulfill the constraints for the entire model. A counterexample of this would be the scale similarity model by Bardina with attached eddy viscosity model, where the contributions of both model terms have to be balanced to achieve a physically consistent net effect. There are, of course, also drawbacks to consider: interpretability of the resulting mapping is further reduced since the functional form and thus the predominant physical effects of the closure terms cannot be recovered from the trained ML model in a straightforward manner. Another, more severe shortcoming observed, for example, by [79] is the lack of long-term stability in this form of closure. This stems from the unavoidable data-model inconsistency during inference in practical simulations. Said inconsistency is caused by the interaction of the model predictions and numerical errors, which induces shifting data statistics and is amplified by error accumulation and self-driving error growth at high wavenumbers. This can either be tackled by removal of this energy through a dissipative mechanism [86] (essentially an additional model term) or the projection of the closure term onto a stable basis with the desired properties [5].

In general, the results for the closure term approximation based on ML methods are highly encouraging. In [86], the authors used a spatial MLP with approx. 500 000 parameters to predict the three components of the subgrid forces of the incompressible momentum equations independently. As inputs, they chose velocity derivatives in the vicinity of the grid point in question. A priori correlations of over 90% with the true subgrid force are reported. In an a posteriori application of the subgrid force together with a dissipative regularization term, the approach outperforms classical closure models. In [81], the authors applied a method for a stochastic model discrimination based on autoregressive models with external influences on the reconstruction of subfilter fluxes in finite-volume LES. Here, the model was not only able to capture the time-space structure of the subgrid fluxes reliably but also identify flow regimes, for example, the near-wall region in a turbulent boundary layer, that require their own local model (an example of an unsupervised clustering method). An approach to estimate the closure forces based on CNNs, which naturally incorporate spatial relationships in the mapping, was investigated in [5], with good success for the reconstruction. Additionally, the influence of an a priori feature selection revealed that the coarse-grained inviscid fluxes contribute noticeably to the training success. While the previous examples almost exclusively build their prediction from spatial data, in [33], the temporal dimension was investigated as an input into NN-based approaches, specifically using the GRU architecture for sequential data (see Section 2.1.3). As an additional focus, the authors investigated the capabilities of the NNs to predict closure terms specific to a given filter kernel. Figure 10 presents the results of the predicted and true closure terms for a database of DHIT flows. As discussed in Section 1.1, the choice of the filter also defines the closure terms. For all filter types investigated, the GRU networks were able to achieve

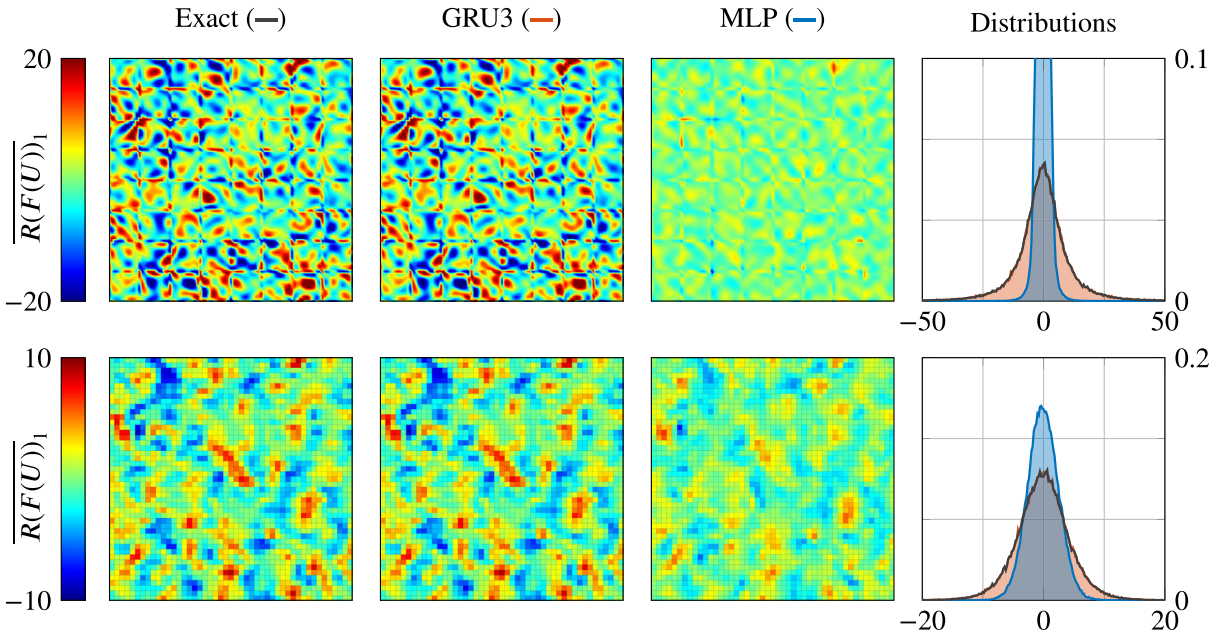

FIGURE 10 Two-dimensional slices of the exact and predicted closure term of the x-momentum equation for a decaying homogeneous isotropic turbulence flow for two filters. Top row: elementwise-L_2 projection onto the space of polynomials of degree $N = 5$, bottom row: top hat filter. The second column is obtained via a temporal NN based on a pointwise gated recurrent unit network, the third column via a spatial multilayer perceptron. Reproduced from [33]

99.9% cross correlation in a priori tests based on a short series of temporal pointwise data of the coarse-scale variables, making the prediction and target visually indistinguishable in the first two columns of Figure 10. These results suggest that the subgrid force terms can be predicted with near-arbitrary precision, which offers the chance to significantly improve practical closure models.

As stated above, a direct closure may suffer from instability due to exponential error growth and is thus not the method of choice without any regularization. This can come in the form of improved model stability or regularization during inference, for example, through an additional dissipation mechanism. An example of the former approach is shown in the left diagram in Figure 11, where the evolution of the kinetic energy for the DHIT case is shown for GRU predictions. The filtered DNS results serve as a reference. While the GRU predictions lead to a very accurate closure and thus LES solution at first, the LES solution diverges strongly soon after. This is a result of a data-model inconsistency and the nonlinear error accumulation of the truncated equations. One possible remedy is sketched in the right panel of Figure 11, where stability training successfully reduced the sensitivity to uncertainties in the inputs during inference [89]. To this end, an additional penalty term with a weighting factor α is added to the loss function during training:

$$(\vec{\theta}_{opt}, g_{opt}) = \underset{\vec{\theta}, g(\vec{\theta})}{\arg\min} \left(\underbrace{\|Y - \hat{Y}\|}_{\text{standard loss}} + \alpha \underbrace{\|\hat{Y} - \tilde{Y}\|}_{\text{penalty}} \right). \tag{10}$$

The noisy output $\tilde{Y}$ is obtained by adding noise $\epsilon(\sigma^2)$ with variance σ^2 to the input vector $\tilde{X} = X + \epsilon(\sigma^2)$ and applying the ANN to this noisy input. The weighting factor α and the noise variance σ^2 are additional hyperparameters which have to be tuned to the specific application. During training, the optimizer thus needs to find a set of parameters which not only minimizes the error on the training dataset but is also robust to random perturbations in the input data (cf. Figure 11). This results in increased stability of the closure model and longer useful predictions. Another example of successful subgrid flux estimation is described in [2]. Here, the authors tackle the problem of the ambiguity of coarse-scale data and closure term by reconstructing them through generative models (a GAN conditioned on a finite-volume solution, cf. Section 2.2.2). Thereby, they do not approximate the exact subgrid flux but a sample with identical statistical properties. Their initial results for closing the viscous Burger's equation (1) show very good a priori and a posteriori agreement with filtered DNS data.

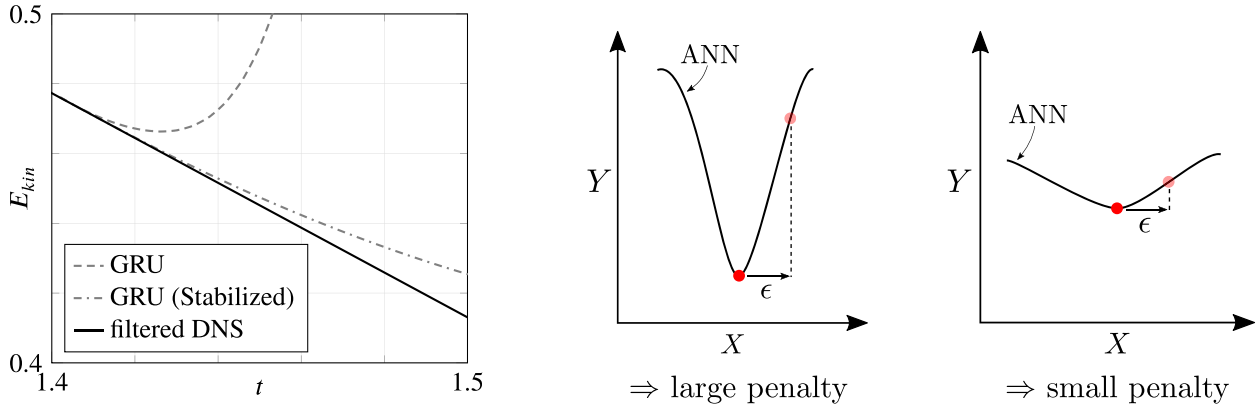

FIGURE 11 *Left*: evolution of the kinetic energy for the decaying homogeneous isotropic turbulence case. The filtered direct numerical simulation data are given as reference. *Right*: stability training to reduce the sensitivity to uncertain input data, basic idea from [89]. Random perturbations ϵ are applied to the input features. The magnitude of the penalty term in the loss function (10) is then proportional to the induced changes in the network output

Summarizing this section, direct estimation of the subgrid forces in the level 2 sense of Equation (9) has a range of advantages over parameter estimation approaches. However, without additional modeling or stabilization, it can lead to a diverging system in the long run. Nonetheless, availability of an accurate estimate of the exact closure at any time can likely help to improve classical modeling approaches. Currently, both spatial and temporal neural networks have been shown to be successful at this task; however, other approaches like kernel methods are likely also suitable. The generalizability of the results and the transfer of the predictions to a practical model are currently open questions. Partially related to the idea of closure term prediction is that of flow field deconvolution or approximate filter inversion, where the deconvolved field is then used to obtain (through scale similarity arguments) an approximation of the exact closure term. Inverting the filter by neural networks has, for example, been proposed in [44], and the reconstruction of the fine scale field through superresolution approaches (essentially the inversion of the cutoff filter) has been proposed in [18].

3.3 | Full PDE modeling

In the rough hierarchy outlined above, the third level of ML augmentation is the actual replacement of traditional PDE solution methods (ie, the discretization of the underlying operators) by a learned approximation that is encapsulated in an ML algorithm. The so-called PINNs are a prominent approach that follows this idea. They can be understood as a combination of supervised (the prediction network) and unsupervised (the residual or constraint) learning methods. For the linear transport equation $\mathcal{N}(u) = \frac{\partial u}{\partial t} + \frac{\partial u}{\partial \vec{x}} = 0$, the associated minimization problem is given by Equation (11) (also cf. L3 in Equation (9)). The task of the PINN is thus to find an approximation $\hat{u}$ under the constraints posed by the PDE. This is achieved in two consecutive steps: first, in a supervised manner, a predictor for the solution field $\hat{u}$ as a function of parameters θ is sought. The functional form $g(\theta)$ of this approximation is established by the ANN, and the parameters are found in the optimization process. The goal of this step is thus to obtain a parameterized function that represents $\hat{u}$ and that is expressed as a differentiable network graph. This graph can now be differentiated with automatic differentiation[4] for any input feature, for example, $\partial\hat{u}/\partial t$. From these derivatives and other atomic operations, the original PDE can be reconstructed—the approximated PDE $\mathcal{N}(\hat{u})$ is, of course, a function of $\vec{\theta}$ and is thus optimizable by the ML method. The optimization problem with respect to the ANN parameters is thus

$$(\vec{\theta}_{opt}, g_{opt}) = \underset{\vec{\theta}, g(\vec{\theta})}{\arg\min} \left(\underbrace{\|u - \hat{u}\|}_{supervised} + \underbrace{\left\|\mathcal{N}(\hat{u})\right\|}_{PDE} \right). \tag{11}$$

[4]Of course, it could also be done analytically, but the graph itself is encapsulated in the network architecture and the framework, so automatic differentiation that is already available through back-propagation is a natural choice.

Clearly, as the loss in the supervised part tends to zero, the residual will also tend to zero as a measure of how well the predicted solution fulfills the underlying PDE: $\hat{u} \to u \Rightarrow \mathcal{N}(\hat{u}) \to 0$. The residual or constraint evaluation does not require any prelabeled training data; the "correct" answer is given by the PDE itself. Thus, this part of the process can be seen as an unsupervised component. The optimization goal is then to find the best fit of both the solution field and the fulfillment of the PDE.

While a lot of work on PINNs for a range of problems exists [60,61], the application of PINNs to turbulence is still in its initial stages. In [29], the authors present a PINN for the prediction of an incompressible laminar and turbulent flow in a DNS setting. Although the topic of this discussion is, strictly speaking, turbulence modeling, that is, a coarse-scale solution to the DNS, the PINN method can also be directly transferred to the LES or RANS equations. To the best of our knowledge, this has not been done yet but would be a natural step. Nonetheless, PINNs have some compelling properties that could be exploited for RANS and LES and are thus discussed here alongside the other approaches. The PINN predictions in [29] for laminar flow achieve low approximation errors, while these errors become considerably larger in the turbulent case. The authors report, in general, a good agreement of short-term predictions, but data indicate that error accumulates in the long run. An interesting outcome of their investigations is that, although no training data on the pressure field is provided, the enforcement of the divergence condition through the constraint part of the network leads to the expected pressure field, without having to split the equations. The efficiency of this approach compared to the classical PDE solution strategies is an open issue, especially since mathematical bounds on the approximation errors and the convergence behavior of this approach have not been established yet. As with any numerical solution methods for PDEs, a tradeoff between speed and accuracy has to be made, and it is not clear how the PINN approach fares here. Also, although the authors note that their approach does not require grids nor does it introduce the classical dispersion and diffusion errors, it remains clear that the approach is not error free and remains an approximation to the true PDE solution. This entails that properties built into classical approximation schemes like conservation are also only approximated by PINNs and that a mathematical analysis of the approach in terms of stability, convergence behavior, and thus efficiency is an open field of research.

In light of the discussion in Section 1.2, PINNs and associated methods offer two desirable features for turbulence modeling or turbulence simulation: as the equations themselves are approximated also all inherent physical constraints are naturally included in the optimization process. In addition, PINNs exploit the equations themselves and thus need very little labeled training data—the vast majority of training data can be unsupervised and thus generated on the fly. This also removes the typical source of training data inconsistency. In addition, the data can be scattered randomly or focused in regions of high solution variance, and the approach allows the direct inclusion of available DNS or experimental data. How far a trained PINN is able to generalize is an issue that needs to be explored further; also, work on interpretability is generally as incomplete as it is for other NN-based approaches. Finally, the question of efficiency arises: Can a PINN be faster than a traditional PDE solver? If so, at what accuracy? Is there a hierarchy of approximations in the sense that a PINN for an LES or RANS solution can indeed be much smaller and thus faster than that for a DNS? Can they be stable if the underlying PDE is in itself an approximate model?

These open questions indicate that there is a lot of research necessary to ascertain the place of PINNs in turbulence modeling and turbulence simulation. In particular, an issue of debate will be the new approximation paradigm introduced by PINNs in that it requires (at least for now) relaxing many of the known and relied-upon properties of classical PDE solvers. However, the advantages of PINNs have also been demonstrated by their original authors; thus, where and how PINNs can fit will have to be established through future research.

4 | CONCLUSION AND OUTLOOK

Progress in turbulence modeling has been rather incremental in the last decades, with a lot of efforts focused on improving existing models in terms of their range of application, generalizability, stability, and performance when combined with different numerical schemes. The development of new models has, so far, predominantly been driven by mathematical and physical considerations. Data-driven approaches have recently gained a strong interest and have been proposed as a "third way" in closure modeling, with the potential of considerably accelerating and possibly unifying model development.

From the authors' perspective, ML-augmented modeling will make neither physical reasoning nor mathematical rigor obsolete. In fact, the opposite is even likely. As discussed in this paper, ML does not necessarily make model development easier or more fool-proof; instead, its successful application requires a careful appreciation of the underlying equations, their properties, their discretizations, and all the intricate and sometimes obscure details of the LES and

RANS methods. In fact, the focus on ML methods might bring an old but often overlooked inconsistency in LES into focus: the separation of a model from the mathematical and physical environment and conditions it was derived for. Based on the theoretical work on explicitly filtered LES and "perfect" numerical schemes, it is easy to subscribe to the notion that there exists, for example, one Smagorinsky's model, and that its associated constant should thus be—based on physical consideration—universal. However, in practical, grid-filtered LES computed with imperfect discretization operators, many variants of the Smagorinsky formulation exist (eg, depending on how the associated velocity gradients are computed), and the considered model interacts with this "dirty" data and environment in complex forms. This issue contributes to large variations reported in LES benchmark results.

ML methods can bring these model-data inconsistencies into sharper focus, and a lot of research efforts are underway to account for them. Thus, success in ML-augmented turbulence modeling will likely also mean a more thorough understanding and appreciation of the other aspects of LES, RANS, and turbulence and help to make the definition of what we can expect from a coarse-grained solution sharper. In this sense, ML can offer capabilities that are indeed complementary to more traditional approaches. From our perspective, the feature extraction or the identification of undiscovered correlations from data are among the most important ones.

The initial successes of ML in improving existing models or reconstructing the closure terms directly are remarkable. What must come next is a consolidation of the many research directions and a generalization of the models to make them actually useful in real applications. This task is made rather difficult by the fact that, although the initial driving force behind data augmentation in turbulence modeling might have been the desire to derive more universal models and reduce empiricism, ML introduces its own new level of hyperparameters, methods, and uncertainties. The resulting ML models can have millions of tunable parameters and may lose all interpretability. Even transferring a published model from one researcher to the next becomes not merely the task of providing a set of equations but the exchange of graphs, training environments, and possibly large amounts of data. Another unanswered issue is the cost-benefit tradeoff, both during training as well as during inference.

Thus, in order to achieve a level of consolidation of the research efforts and the chance to select the most promising approaches to go forward, the next step should be the establishment of a means of comparing and benchmarking the results. Such a database should contain a well-described training dataset, most likely from DNS, open to everyone, and a private test set for the blind evaluation of the proposed models. Setting up such a validation case and administering it should be approached as a community effort.

ACKNOWLEDGMENTS

This research was funded by Deutsche Forschungsgemeinschaft (DFG, German ResearchFoundation) under Germany's Excellence Strategy—EXC 2075—390740016. The authors gratefully acknowledge the support and the computing time provided by the HLRS through the project "hpcdg." Open access funding enabled and organized by Projekt DEAL.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] J. Alcala and I. Timofeyev, *Subgrid-scale parametrization of unresolved scales in forced burgers equation using generative adversarial networks* (*gan*). 2020, arXiv preprint arXiv:2007.06692.

[3] D. Arthur and S. Vassilvitskii, k-means++: The advantages of careful seeding, Technical report 2006-13, Stanford InfoLab, June 2006.

[4] G. K. Batchelor, *The theory of homogeneous turbulence*, Cambridge University Press, Cambridge, MA, 1953.

[5] A. Beck, D. Flad, and C. D. Munz, Deep neural networks for data-driven LES closure models, J. Comput. Phys. **398** (2019), 108910.

[6] A. D. Beck, J. Zeifang, A. Schwarz, and D. G. Flad, A neural network based shock detection and localization approach for discontinuous Galerkin methods, J. Comput. Phys. **423** (2020), 109824.

[7] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, Support vector clustering, J. Mach. Learn. Res. **2** (2001), 125–137.

[8] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the 5th Annual Workshop on Computational Learning Theory, 1992, pp. 144–152.

[9] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, Machine learning for fluid mechanics, Ann. Rev. Fluid Mech. **52** (2020), 477–508.

[10] M. Campbell, A. J. Hoane, and F. Hsu, Deep blue, Artif. Intell. **134** (2002), 57–83.

[11] J. Chan, Z. Wang, A. Modave, J. F. Remacle, and T. Warburton, GPU-accelerated discontinuous galerkin methods on hybrid meshes, J. Comput. Phys. **318** (2016), 142–168.
[12] J. R. Chasnov, The decay of axisymmetric homogeneous turbulence, Phys. Fluids **7** (1995), 600–605.
[13] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, *On the properties of neural machine translation: Encoder-decoder approaches*, Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, October 2014, pp. 103–111.
[14] M. Curcic, A parallel fortran framework for neural networks and deep learning, ACM SIGPLAN Fortran Forum **38** (2019), 4–21.
[15] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. **2** (1989), 303–314.
[16] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, 2017, arXiv preprint arXiv:1702.08608.
[17] K. Duraisamy, G. Iaccarino, and H. Xiao, Turbulence modeling in the age of data, Ann. Rev. Fluid Mech. **51** (2019), 357–377.
[18] K. Fukami, K. Fukagata, and K. Taira, *Super-resolution reconstruction of turbulent flows with machine learning*, 2018, arXiv preprint arXiv:1811.11328.
[19] K. Fukami, Y. Nabae, K. Kawai, and K. Fukagata, Synthetic turbulent inflow generator using machine learning, Phys. Rev. Fluids **4** (2019), 064603.
[20] K. Fukami, T. Nakamura, and K. Fukagata, Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data, Phys. Fluids **32** (2020), 095110.
[21] M. Gamahara and Y. Hattori, Searching for turbulence models by artificial neural network, Phys. Rev. Fluids **2** (2017), 054604.
[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets, Adv. Neural Inf. Process. Syst. **27** (2014), 2672–2680.
[23] J. Haugeland, *Artificial intelligence: The very idea*, 5th ed., MIT Press, Cambridge, MA, 1990.
[24] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 770–778.
[25] G. E. Hinton and R. S. Zemel, Autoencoders, minimum description length and helmholtz free energy, Adv. Neural Inf. Process. Syst. **6** (1994), 3–10.
[26] T. K. Ho, *Random decision forests*, Proceedings of 3rd International Conference on Document Analysis and Recognition, Vol. 1, 1995, pp. 278–282.
[27] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Comput. **9** (1997), 1735–1780.
[28] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. **4** (1991), 251–257.
[29] X. Jin, S. Cai, H. Li, and G.E. Karniadakis, *Nsfnets* (*Navier-Stokes flow nets*): *Physics-informed neural networks for the incompressible Navier-Stokes equations*, 2020. arXiv preprint arXiv:2003.06496.
[30] D. P. Kingma and M. Welling, *Auto-encoding variational Bayes*, 2013. arXiv preprint arXiv:1312.6114.
[31] A. N. Kolmogorov, A refinement of previous hypotheses concerning the local structure of turbulence in a viscous incompressible fluid at high reynolds number, J. Fluid Mech. **13** (1962), 82–85.
[32] N. Krais, A. Beck, T. Bolemann, H. Frank, D. Flad, G. Gassner, F. Hindenlang, M. Hoffmann, T. Kuhn, and M. Sonntag, Flexi: A high order discontinuous galerkin framework for hyperbolic–parabolic conservation laws, Comput. Math. Appl. **81** (2020), 186–219.
[33] M. Kurz and A. Beck, *A machine learning framework for les closure terms*, 2020, arXiv preprint arXiv:2010.03030.
[34] J. N. Kutz, Deep learning in fluid dynamics, J. Fluid Mech. **814** (2017), 1–4.
[35] J. A. Langford and R. D. Moser, Optimal LES formulations for isotropic turbulence, J. Fluid Mech. **398** (1999), 321–346.
[36] J. Larsson, S. Kawai, J. Bodart, and I. Bermejo-Moreno, Large eddy simulation with modeled wall-stress: Recent progress and future directions, Mech. Eng. Rev. **3** (2016), 15-00418.
[37] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature **521** (2015), 436–444.
[38] Y. Lee, H. Yang, and Z. Yin, PIV-DCNN: Cascaded deep convolutional neural networks for particle image velocimetry, Exp Fluids **58** (2017), 171.
[39] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence, J. Turbul. **N31** (2008), 1–29.
[40] J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. **807** (2016), 155–166.
[41] J. Ling and J. Templeton, Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged navier stokes uncertainty, Phys. Fluids **27** (2015), 085103.
[42] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, The expressive power of neural networks: A view from the width, Adv. Neural Inf. Process. Syst. (2017), 6231–6239. https://dl.acm.org/doi/abs/10.5555/3295222.3295371.
[43] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, 1967, pp. 281–297.
[44] R. Maulik and O. San, A neural network approach for the blind deconvolution of turbulent flows, J. Fluid Mech. **831** (2017), 151–181.
[45] R. Maulik, O. San, and J. D. Jacob, Spatiotemporally dynamic implicit large eddy simulation using machine learning classifiers, Physica D **406** (2020), 132409.
[46] R. Maulik, H. Sharma, S. Patel, B. Lusch, and E. Jennings, Deploying deep learning in openfoam with tensorflow, AIAA Scitech 2021 Forum (2021), 1485.
[47] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. **5** (1943), 115–133.
[48] A. Mohan, D. Daniel, M. Chertkov, and D. Livescu, *Compressed convolutional LSTM: An efficient deep learning framework to model high fidelity 3d turbulence*, 2019. arXiv preprint arXiv:1903.00033.

[49] R. D. Moser, S. W. Haering, and G. R. Yalla, Statistical properties of subgrid-scale turbulence models, Ann. Rev. Fluid Mech. **53** (2021), 255–286.

[50] G. Novati, H. L. de Laroussilhe, and P. Koumoutsakos, *Automating turbulence modeling by multi-agent reinforcement learning*, 2020. arXiv preprint arXiv:2005.09023.

[51] M. Oberlack, *Invariant modeling in large-eddy simulation of turbulence*, Annual Research Briefs, Stanford University, Stanford, CA, 1997, 209–220.

[52] D. Oberle, C. D. Pruett, and P. Jenny, Temporal large-eddy simulation based on direct deconvolution, Phys. Fluids **32** (2020), 065112.

[53] J. Ott, M. Pritchard, N. Best, E. Linstead, M. Curcic, and P. Baldi, *A Fortran-Keras deep learning bridge for scientific computing*, 2020. arXiv preprint arXiv:2004.10652.

[54] A. Pal, *Deep learning parameterization of subgrid scales in wall-bounded turbulent flows*, 2019. arXiv preprint arXiv:1905.12765.

[55] S. Pandey, J. Schumacher, and K. R. Sreenivasan, A perspective on machine learning in turbulent flows, J. Turbul. **21** (2020), 1–18.

[56] E. J. Parish and K. Duraisamy, A paradigm for data-driven predictive modeling using field inversion and machine learning, J. Comput. Phys. **305** (2016), 758–774.

[57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, in *Advances in Neural Information Processing Systems*, Vol **32**, H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc, Red Hook, NY, 2019, 8024–8035.

[58] A. Pollard, L. Castillo, L. Danaila, and M. Glauser, *Whither turbulence and big data in the 21st century?* Springer, New York, NY, 2016.

[59] C. D. Pruett, *Toward the de-mystification of les*, Proceedings of the 3rd AFOSR International Conference on DNS/LES, 2001, pp. 1–8.

[60] M. Raissi and G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. **357** (2018), 125–141.

[61] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. **378** (2019), 686–707.

[62] D. Ray and J. S. Hesthaven, An artificial neural network as a troubled-cell indicator, J. Comput. Phys. **367** (2018), 166–191.

[63] R. Rogallo, *Numerical experiments in homogeneous turbulence*, 1981. NASA TM-81315.

[64] R. S. Rogallo and P. Moin, Numerical simulation of turbulent flows, Annu. Rev. Fluid Mech. **16** (1984), 99–137.

[65] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, Psychol. Rev. **65** (1958), 386–408.

[66] G. Santin and B. Haasdonk, *Kernel methods for surrogate modeling*, 2019. arXiv preprint arXiv:1907.10556.

[67] F. Sarghini, G. De Felice, and S. Santini, Neural networks based subgrid scale modeling in large eddy simulations, Comput. Fluids **32** (2003), 97–108.

[68] M. Schmelzer, R. P. Dwight, and P. Cinnella, Discovery of algebraic Reynolds-stress models using sparse symbolic regression, Flow Turbul. Combust. **104** (2020), 579–603.

[69] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. **61** (2015), 85–117.

[70] B. Schölkopf, P. Simard, A. J. Smola, and V. Vapnik, Prior knowledge in support vector kernels, Adv. Neural Inf. Process. Syst. **11** (1998), 640–646.

[71] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature **529** (2016), 484–489.

[72] A. J. Smola and B. Schölkopf, A tutorial on support vector regression, Stat. Comput. **14** (2004), 199–222.

[73] C. G. Speziale, Galilean invariance of subgrid-scale stress models in the large-eddy simulation of turbulence, J. Fluid Mech. **156** (1985), 55–62.

[74] I. Steinwart and A. Christmann, *Support vector machines*, Springer, New York, NY, 2008.

[75] H. Tang, J. Rabault, A. Kuhnle, Y. Wang, and T. Wang, Robust active flow control over a range of reynolds numbers using an artificial neural network trained through deep reinforcement learning, Phys. Fluids **32** (2020), 053605.

[76] R. Temam, Approximation of attractors, large eddy simulations and multiscale methods, Proc. R. Soc. Lond. Ser. A Math. Phys. Sci. **434** (1991), 23–39.

[77] B. Tracey, K. Duraisamy, and J. Alonso, *Application of supervised learning to quantify uncertainties in turbulence and combustion modeling*, Proceedings of the 51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 2013, p. 259.

[78] B. C. Vermeire, F. D. Witherden, and P. E. Vincent, On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools, J. Comput. Phys. **334** (2017), 497–521.

[79] A. Vollant, G. Balarac, and C. Corre, Subgrid-scale scalar flux modelling based on optimal estimation theory and machine-learning procedures, J. Turbul. **18** (2017), 854–878.

[80] A. Vollant, G. Balarac, G. Geraci, and C. E. Corre, *Optimal estimator and artificial neural network as efficient tools for the subgrid-scale scalar flux modeling*, in *Proceedings of the Summer Program*, Stanford University, Stanford, CA, 2014, 435–444.

[81] T. von Larcher, A. Beck, R. Klein, I. Horenko, P. Metzner, M. Waidmann, D. Igdalov, G. Gassner, and C. D. Munz, Towards a framework for the stochastic modelling of subgrid scale fluxes for large eddy simulation, Meteorol. Z. **24** (2015), 313–342.

[82] J. X. Wang, J. L. Wu, and H. Xiao, Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on dns data, Phys. Rev. Fluids **2** (2017), 034603.

[83] C. J. C. H. Watkins and P. Dayan, Q-learning, Mach. Learn. **8** (1992), 279–292.

[84] C. Xie, K. Li, C. Ma, and J. Wang, Modeling subgrid-scale force and divergence of heat flux of compressible isotropic turbulence by artificial neural network, Phys. Rev. Fluids **4** (2019), 104605.

[85] C. Xie, J. Wang, H. Li, M. Wan, and S. Chen, Artificial neural network mixed model for large eddy simulation of compressible isotropic turbulence, Phys. Fluids **31** (2019), 085112.

[86] C. Xie, J. Wang, and E. Weinan, Modeling subgrid-scale forces by spatial artificial neural networks in large eddy simulation of turbulence, Phys. Rev. Fluids **5** (2020), 054606.

[87] Q. Zhang, Y. N. Wu, and S. C. Zhu, *Interpretable convolutional neural networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8827–8836.

[88] X. Z. Zhao, T. Y. Xu, Z. T. Ye, and W. J. Liu, A tensorflow-based new high-performance computational framework for CFD, J. Hydrodyn. **32** (2020), 735–746.

[89] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, *Improving the robustness of deep neural networks via stability training*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4480–4488.

[90] Z. Zhou, G. He, S. Wang, and G. Jin, Subgrid-scale model for large-eddy simulation of isotropic turbulent flows using an artificial neural network, Comput. Fluids **195** (2019), 104319.

[91] L. Zhu, W. Zhang, J. Kou, and Y. Liu, Machine learning methods for turbulence modeling in subsonic flows around airfoils, Phys. Fluids **31** (2019), 015105.

How to cite this article: Beck A, Kurz M. A perspective on machine learning methods in turbulence modeling. *GAMM-Mitteilungen*. 2021;44:e202100002. https://doi.org/10.1002/gamm.202100002

APPENDIX A. NOTES ON PRACTICAL ISSUES OF ML AND TURBULENCE MODELING

In this appendix, we briefly summarize some notes on practical issues that might be helpful to the community at large. They are primarily drawn from our own experience employing ML methods in turbulence modeling in [5,33] and to shock detection and capturing for high-order methods in [6] and from coupling a legacy Fortran code to ML frameworks.

A.1 Frameworks and codes

As mentioned in Section 1.2.1, the question of whether ML-augmented methods will see widespread acceptance and use in the CFD community is an issue of efficiency. The computational efficiency of ML-based turbulence models is determined by two different aspects: the evaluation of the model (the ML algorithm) and the coupling of the models into the flow solver (the ML-to-CFD interface). The latter is especially crucial in order to incorporate ML-based turbulence models into existing legacy codes. While most flow solvers are typically written in Fortran or C/C++, ML libraries like PyTorch and TensorFlow are commonly used with their Python API. In the following section, we will give a quick overview of some proposed coupling mechanisms currently found in literature. As with all the issues discussed herein, the dynamics of the ML development allows us only to give a current snapshot.

In their own research, the authors have employed the forpy[5] software package combined with the TensorFlow serving API to extend the open-source solver FLEXI [32] with ML capabilities. A TensorFlow server is started at the beginning of the simulation with a specified ANN. By means of the forpy package, the Fortran solver then calls Python routines during runtime, which send the input data for the ANN to the TensorFlow server and pass the received results back into FLEXI. While this approach can be used quite easily to extend existing software, the performance losses of this coupling strategy are quite significant due to the Fortran-Python interface.

A recent example of an in situ coupling of the Tensorflow framework with OpenFOAM is given in [46]. Here, the authors avoid the coupling of Python code by utilizing the TensorFlow C API, which can be used to directly link TensorFlow to OpenFOAM. Further details on the coupling are given in a tutorial style. The reported overall performance was comparable for the ML-based surrogate model and the traditional turbulence model. This is especially remarkable since the evaluation of the surrogate model requires significantly more floating-point operations. Overall, this approach seems highly suitable for large-scale applications.

Another recent approach to couple Fortran codes with a common ML library is the Fortran-Keras-Bridge (FKB) [53], which extends the functionality of the underlying Neural Fortran library [14]. The FKB implements the basic

[5]https://github.com/ylikx/forpy

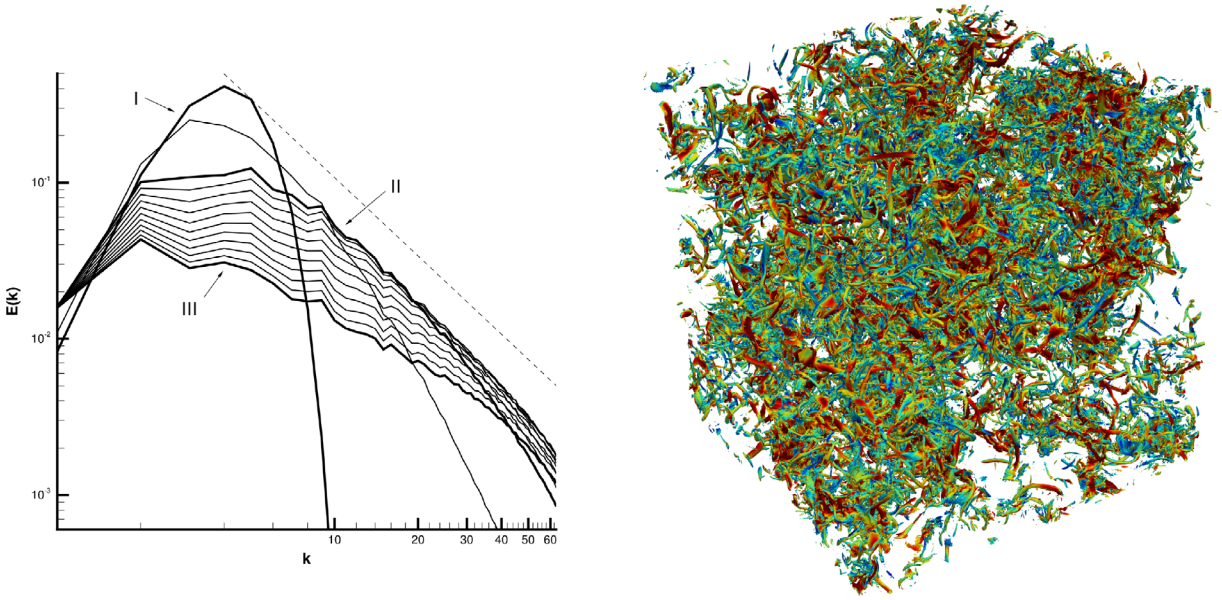

FIGURE A1 *Left*: Temporal evolution of the kinetic energy spectrum from $t = 0\ T^*$ to $t = 2.0\ T^*$ for the decaying homogeneous isotropic turbulence (DHIT); I: initial spectrum, II: start of data collection $t = 1.0\ T^*$, III: end of data collection $t = 2.0\ T^*$, dashed line: $k^{-5/3}$; *Right*: field solution of the DHIT visualized by isosurfaces of the vorticity magnitude colored by velocity magnitude at $t = T^*$

functionalities for training and execution of fully connected ANN in Fortran while allowing us to add more advanced custom layers. The library is thus independent from TensorFlow. Additional tools are provided to convert saved models from the Keras format into a format readable by the FKB and vice versa.

The methods discussed above use a CPU-only implementation of the flow solver. Independent of ML, noticeable work is also dedicated to using the computational potential of GPU acceleration for classical scientific computing by using hybrid programming approaches, for example, [11,78]. Also, first flow solvers leverage TensorFlow solely for its efficiency and GPU abilities for traditional finite difference methods without using any ML [88]. The latter approach can especially be trivially extended to also incorporate the ML capabilities of the TensorFlow library. Such developments might facilitate the integration of ML-based modeling into flow solvers in the future.

A.2 A training dataset

One urgent question that ML and turbulence modeling has to answer is that of overall efficiency in terms of data acquisition costs. Thus, sharing training data is not only reasonable from an economic perspective, but a common database can also serve as a sorely needed benchmark with which to compare the myriad of different approaches touched on here. One such dataset for DHIT has been discussed in the examples in Section 3.2. The dataset consists of 16 DNS runs of the DHIT test case, of which 14 runs are available freely, and 2 runs are kept as hidden test datasets. Each DNS is initialized from the same initial spectrum of the kinetic energy given by Chasnov [12] as

$$E(k, t=0) = \frac{1}{2} a_s {u_0}^2 {k_p}^{-1} \left(\frac{k}{k_p}\right)^s exp\left[-\frac{1}{2} s \left(\frac{k}{k_p}\right)^2\right]. \tag{A1}$$

We chose $s = 4$, ${u_0}^2 = 5$, and $k_p = 4$ and construct the initial velocity components as proposed by [63]. The corresponding pressure field is computed from these initial incompressible velocities by solving the associated Poisson equation in spectral space. The Mach number based on the maximum initial velocity magnitude was set to 0.1 via the mean pressure. The computational domain is a cube of size $[0, 2\pi]^3$ with periodic boundary conditions on all faces. The Reynolds number based on the Taylor mircoscale for the onset of the exponential energy decay is $Re_\lambda \approx 180$. Figure A1 (left) shows the temporal evolution of the energy spectrum of the DNS from the initial condition $t = 0$ (I) to $t = 2\ T^*$ (III), where the eddy turnover timescale is defined from the initial mean velocity and the integral length scale $T^* = \bar{v}_{initial}/L_{int}$. Data collection starts at $t = 1.0\ T^*$ (II). The solution vector U and the time derivative $R(U) = \frac{\partial U}{\partial t}$ are stored at $\Delta t = 0.1\ T^*$ time intervals. Thus, in total, 154 snapshots of the turbulent field are available, and each file has a size of approx. 11 GB.

All DNS computations were conducted with an eighth-order accurate Discontinuous Galerkin Spectral Element discretization of the compressible NSE described in [32]. The computational grid for the DNS consists of 64^3 equispaced, cubical elements, in each of which the solution is approximated by a tensor product polynomial basis of degree $N = 7$, leading to 512^3 spatial degrees of freedom in total. The data are stored in the HDF5 format and can be interpolated onto an equidistant grid for easier handling. Access to these data is currently available upon emailing the authors. In the future, integration into an online database is planned. Another useful database with data on forced HIT with higher Re_λ but fewer data points is hosted by the Johns Hopkins University and can be found at http://turbulence.pha.jhu.edu [39]. Certainly, there are other datasets that we are not aware of.