Differentiable physics-enabled closure modeling for Burgers' turbulence

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 015017 https://doi.org/10.1088/2632-2153/acb19c

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
10 November 2022

REVISED
19 December 2022

ACCEPTED FOR PUBLICATION
9 January 2023

PUBLISHED
9 February 2023

PAPER

Differentiable physics-enabled closure modeling for Burgers' turbulence

Varun Shankar[1], Vedant Puri[1], Ramesh Balakrishnan[2], Romit Maulik[2,*] and Venkatasubramanian Viswanathan[1,*]

[1] Carnegie Mellon University, Pittsburgh, PA, United States
[2] Argonne National Laboratory, Lemont, IL, United States
[*] Authors to whom any correspondence should be addressed.

E-mail: rmaulik@anl.gov and venkvis@cmu.edu

Keywords: turbulence, burgers, subgrid-stress modeling, differentiable physics, machine learning, neural operators

Abstract

Data-driven turbulence modeling is experiencing a surge in interest following algorithmic and hardware developments in the data sciences. We discuss an approach using the differentiable physics paradigm that combines known physics with machine learning to develop closure models for Burgers' turbulence. We consider the one-dimensional Burgers system as a prototypical test problem for modeling the unresolved terms in advection-dominated turbulence problems. We train a series of models that incorporate varying degrees of physical assumptions on an *a posteriori* loss function to test the efficacy of models across a range of system parameters, including viscosity, time, and grid resolution. We find that constraining models with inductive biases in the form of partial differential equations that contain known physics or existing closure approaches produces highly data-efficient, accurate, and generalizable models, outperforming state-of-the-art baselines. Addition of structure in the form of physics information also brings a level of interpretability to the models, potentially offering a stepping stone to the future of closure modeling.

1. Introduction

Simulation of fluid flow for practical applications is characterized by high Reynolds number turbulent flows over complex geometries [1, 2]. The temporal and spatial scales of motion in such flows span several orders of magnitude [3]. The cost of direct numerical simulations (DNSs), which attempt to capture all energy containing lengthscales in the flow, grows superlinearly with the Reynolds number of the flow [4–6], limiting the scope of fully resolving simulations to canonical geometries at medium Reynolds numbers [3].

Turbulence closure models have been developed for cases where resolving and evolving the entirety of velocity and pressure spectra is not possible. Techniques such as large Eddy simulations (LES) and Reynolds averaged Navier–Stokes (RANS) allow for cheaper calculations of high Reynolds number flows by resolving only a portion of the spectrum and 'modeling' the rest [7]. Put simply, the goal of turbulence modeling is to find a closed system that can describe the time-evolution of observables, such as the resolved velocity spectrum, via a 'closure' map that allows one to untangle the dependence of said observables on unresolved variables [8–10].

The problem of turbulence closure modeling has been extensively studied for over a century, dating back to Boussinesq's Eddy viscosity hypothesis in 1877 [11]. More recently, machine learning (ML) based methods have utilized for modeling fluid flow problems [12–20]. The success of ML is built upon the ability of deep neural networks to approximate functions in high-dimensional spaces with a number of parameters that does not scale exponentially with the dimension of the problem space [21]. As the underlying problem in closure modeling is of finding the functional mapping between high-dimensional vector spaces, deep neural networks can be utilized in a supervised learning framework.

However, the performance of ML models to turbulence modeling has largely been unsatisfactory. A probable reason is that while neural networks are touted to break the curse of dimensionality, the cost of

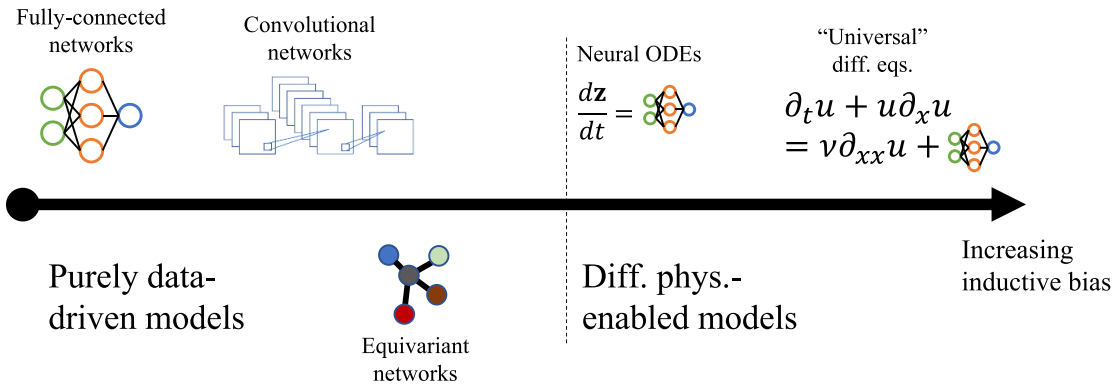

Figure 1. Data-driven models can incorporate different inductive biases, from locality in convolutional networks, symmetries in equivariant networks, to differential equation forms in universal differential equations. Models on the right are constrained by known physics, such as a dynamical system or advective and dissipative terms, leading to interpretable models where the learning problem has been restricted to unknown or residual physics.

attaining an accurate model may be larger. The cost of generating high-fidelity data for the model does scale with dimensionality [7], and even if data is readily available, ML models need to be trained on large number of samples to accurately approximate even simple functions. In other words, an optimal closure model may be present in the space of functions that a neural network architecture can express, but there is no clear path towards realizing it.

Some success has been seen, however, by augmenting neural network models with physical priors. For example, Ling *et al* [22, 23]. improved upon conventional ML models by embedding Galilean invariance in the prediction of Reynolds stress tensor for RANS simulations. Shankar *et al* [24] enhanced the output of a convolutional neural network model on homogeneous isotropic turbulence by imposing the divergence-free constraint on the model output. Both approaches can be understood as attempts to narrow down the space of functions that a neural network architecture can represent.

Rackauckas *et al* [25]. further develops inductive biases in data-driven learning by directly embedding trainable parameters in a model's governing differential equations. Training such parameters against a loss function that depends on the solution to the governing system requires passing gradient information through a differential equation solve. Therefore a solver that is compatible with automatic-differentiation (AD) toolchains in needed. Directly backpropagating gradients is computationally expensive for large solves, so the adjoint optimization method [26–29] is utilized. Computational fluid dynamics (CFD) codes such as SU2 have utilized the adjoint method for sensitivity analysis and shape optimization [30].

This idea is formalized into the notion of Differentiable Physics [31], defined as the set of scientific avenues emerging from the marriage of state-of-the-art partial differential equation (PDE) solvers with differentiable programming. The promise of differentiable physics is that ML models can be developed to learn specific unknown or residual physics, leading to more interpretable models than classical ML approaches. We envision the landscape of data-driven models on an axis of increasing inductive biases in figure 1.

In this work, we comprehensively test the ability of ML techniques to develop turbulence closure models that are resolution independent, interpretable in the language of flow physics, and generalizable across a range of Reynolds numbers. We consider the one-dimensional (1D) Burgers system as the ideal test problem for modeling the energy spectra observed in advection-dominated turbulence problems, and narrow our focus to the task of subgrid-stress modeling. We test a range of differentiable physics models, that vary from a black-box approach to training coefficients in state-of-the-art turbulence models. Experiments are conducted over a range of Reynolds numbers and grid resolutions to assess the generalizability of the model, and over multiple resolution-independent architectures. We find that physics-based inductive biases are critical to the development of data-efficient and accurate ML closures, and that these ML closures can outperform state-of-the-art closure baselines in a wide range of flows.

1.1. Our contributions

- We present a novel approach to learning data-driven closures by pairing non-local neural operators, which are grid independent, and a differentiable PDE solver, which allows for *a-posteriori* training on the velocity field directly. This is unique to existing approaches that either segregate the solver from the training, learning some intermediate quantity such as the Reynolds stress, or that use grid dependent neural architectures such as convolutional nets to develop a non-local closure.

- We design, train, and assess a large suite of PDE-described closure models that allow for significant interpretability of the data-driven models and investigation into the effect of traditional fluids approximations, such as the Boussinesq hypothesis, on the closure learning problem.
- We present resolution-invariant data-driven closure models, which to our knowledge has not been done in the literature. We rigorously test our models against existing baselines and show that this method can consistently outperform baselines on a variety of grids without retraining.
- Much of the literature on ML turbulence models focus on developing local closure corrections because these models are naively scalable and generalizable to larger meshes. We choose to develop non-local closures that require the full state of the system as input and quantitatively show that this choice results in lower test error compared to a local model with similar architecture on the Burgers system.
- We propose a modification to the classic Smagorinsky closure model that uses a neural network to compute a spatially and temporally evolving Smagorinsky constant. We show that it systematically reduces error over the baseline Smagorinsky model, as well as existing non-local dynamic Smagorinsky models, and that the model can maintain the improvement across a variety of grid resolutions.

2. Background

2.1. Burgers turbulence

Turbulent flows are characterized by a competition between viscous forces, which damp out velocity fluctuations by converting kinetic energy into heat, and inertial forces, such as gravitation or pressure gradients. These tend to generate and preserve velocity and pressure fluctuations [32]. In low Reynolds number flows, viscous forces dominate and the flow is near perfectly damped, meaning that any arbitrary fluctuation in the velocity field would be smoothed out in no time [33]. The velocity fields adjusts almost instantaneously to any changes in the pressure gradient that drives the flow. At high Reynolds numbers, however, viscous forces may not be strong enough to dampen out velocity fluctuations, and even tiny disturbances may cause the entire flow to destabilize into turbulence [34]. Energy dissipation primarily happens at the smaller scales as the rate of dissipation scales with the wavenumber squared.

We restrict the scope of this work to the task of subgrid stress modeling in advection-dominated turbulence problems. The unforced, viscous Burgers problem, equation (1), an advection-diffusion type problem [35] that exhibits an energy cascade similar to the Navier–Stokes system, is the perfect test bench. The Burgers system is governed by

$$\partial_t u + u\partial_x u = \nu \partial_{xx} u, \tag{1}$$

where $u(x,t)$, the solution to the Burgers system, represents a time-varying velocity field and ν is the kinematic viscosity. The 1D viscous Burgers system is a canonical PDE that is widely used as a simplified case study for subgrid model development and analysis [35–38]. The nonlinear advective term coupled with viscous dissipation leads to similar multiscale phenomena observed in more complex flow dynamics. Equation (1) is deterministic which allows us to directly compare trajectories rather than statistical averages, speeding up model training. We insulate the problem from arbitrary effects of boundary conditions by considering equation (1) on the entire real line, $\mathbb{R}$, which is approximated by periodic boundary conditions on an interval. The Burgers equation can also be written in its conservative form

$$\partial_t u + \frac{1}{2}\partial_x u^2 = \nu \partial_{xx} u, \tag{2}$$

which is typically more amenable to numerical calculation.

2.2. Large eddy simulations

As high Reynolds number flows have a wide energy spectrum, DNS of such flows is often computationally intractable, and a turbulence model is needed. LES resolves the flow variables starting from the intertial range up to a cut-off length, Δ, that depends upon the computation grid [39, 40]. Critically, an LES calculation ignores the smallest lengthscales in the flow, which are most computationally expensive to resolve. Filtering is accomplished by

$$\begin{aligned} u(\boldsymbol{x},t) &= \overline{u}(\boldsymbol{x},t) + u'(\boldsymbol{x},t) \\ \overline{u}(\boldsymbol{x},t) &= G_\Delta \star u(\boldsymbol{x},t), \end{aligned} \tag{3}$$

where G_Δ is a low-pass filtering operator. Quantities of interest are interpolated onto a computational grid that can resolve flow features only up to lengthscale Δ. The grid has much lower resolution than what is

needed to fully resolve the flow, and evolving flow variables on this grid lead to saving computational resources.

Filtering the Burgers equation in this manner leads to

$$\partial_t \overline{u} + \frac{1}{2}\partial_x \overline{u}^2 = \nu \partial_{xx} \overline{u} - \frac{1}{2}\partial_x \overline{u'^2}, \tag{4}$$

The LES equations for the Burgers system. Evolving $\overline{u}$ per equation (4) is not possible as one does not have access to the subgrid stress term, $\eta := \frac{1}{2}\overline{u'u'}$, and ignoring it leads to an incorrect flow field and numerical instabilities. For the Navier–Stokes equations, ignoring subgrid stresses leads to numerical instabilities spurious high-frequency modes [41]. The closure problem of LES is of modeling the effect of η on the dynamics of $\overline{u}$. This is called subgrid stress modeling. As one does not have access to the unresolved quantities, subgrid stress models are formulated in terms of the resolved quantities.

Kolmogorov hypothesized that the finer scales are largely similar [4], and are primarily responsible for energy dissipation. One approach to closure modeling is to take out small amounts of energy from the high-frequency modes in $\overline{u}$ by applying a mild low-pass filter at every timestep. Mullen and Fisher [41]. has applied this methodology to the Navier–Stokes equations in the CFD code NEK5000 [42], and Layton *et al* [43]. has proven existence of unique strong solutions to the resulting continuum model for the same.

Another approach is to directly model the commutative error term, η in terms of $\overline{u}$ with a map $\eta(\overline{u})$, and evolve the flow variables as per equation (4). Hypothesizing a functional form for additive closures of this sort is highly problem dependent. A common hypothesis is the eddy viscosity family of models [44, 45] that introduce an 'effective' or 'eddy' viscosity, ν_t, to the momentum equation for enhancing dissipation. This leads to a functional form of $\eta = \partial_x \nu_t \partial_x \overline{u}$, where ν_t could be a fixed scalar based on known flow physics and computational grid, or be defined by a transport equation. In section 4 we consider several such models and apply them to the Burgers problem.

3. Method

3.1. Numerical methods

We compute solutions to the Burgers equation via a pseudospectral method [46]. The periodic 1D domain is discretized with a regular grid of N grid points. The periodic nature of the solutions allow for expansion in a Fourier basis, as per

$$u(x,t) = \sum_{k=-N/2}^{N/2} \hat{u}_k(t) e^{ikx}. \tag{5}$$

The linear term of the equation can be computed pointwise in Fourier space

$$\partial_{xx}\hat{u}(k,t) = -k^2 \hat{u}(k,t), \tag{6}$$

however the nonlinear term introduces interactions between all the wavemodes and thus requires more care for efficient computation. Therefore, it is more suitable to compute the quadratic term u^2 pointwise in real space at the N collocation points, then compute the spatial derivative in Fourier space.

The spatial discretization allows for a method of lines approach to time evolution. The discrete Fourier modes from the spectral basis expansion produces a system of N ordinary differential equations (ODEs) that can be solved with standard ODE integration schemes. We choose the Tsitouras adaptive 5th order scheme [47].

3.2. Problem statement

Accurate simulation of the governing Burgers equation imposes certain restrictions on the temporal and spatial discretization. As we use an adaptive time-stepping scheme with fixed error tolerance, we focus on errors arising from the spatial discretization. The spatial grid should be sufficiently dense to resolve the smallest dissipative length scales in the flow, typically much smaller than the characteristic length scales of the flow.

We wish to approximate solutions to the Burgers equation on a considerably coarser grid of M points, by exploring a family of data-driven models enabled by algorithmic differentiation of the temporal and spatial discretization schemes. Each of the models is described by a PDE or set of PDEs, where certain terms in the equations are computed by deep neural networks. Thus, we enable significant interpretability of the models as network outputs can be readily identified as components of traditional closure modeling approaches.

3.3. Models

In this section, we provide a description of the various models that we use in our experiments. First, we make clear our notational conventions.

$u(x,t)$ is the spatiotemporal velocity field we are modeling. $\overline{u}(x,t)$ is the filtered velocity field on the LES grid. We drop the parentheses for brevity. η represents an unknown closure variable we are modeling. The exact description of this variable differs across models. In some models, an additional transport equation is introduced to govern the dynamics of the closure variable. The $\bullet$ and the ∇ symbols correspond to the time derivative and spatial derivative of the field respectively. A subscript 0 represents the initial condition of the field.

Neural networks are indicated with subscript θ. These functions are parameterized by a large number of weights with functional form described in the subsequent section. Network inputs and outputs are written as:

$$c, d = f_\theta(a, b; e), \tag{7}$$

where a and b are explicit inputs to the network, and c and d are outputs of the network with implicit dependence on variable e. For example, $\eta = f_\theta(u; x)$ represents a network that takes the spatially varying velocity field as input and produces a spatially varying η field as output.

The ground truth equation, or *none* model, is given by:

$$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u}. \tag{8}$$

Since there is no closure term or trainable networks, we use this as a baseline comparison for our learned models.

We also compare our models to a couple baseline closure models [48], the constant Smagorinsky model (further denoted as *smag-const*), and the dynamic Smagorinsky model (further denoted as *smag-dyn*). The constant Smagorinsky model [49] is given by:

$$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\nu_t\nabla\overline{u}) \tag{9}$$

$$\nu_t = (C_s\delta x)^2|\nabla\overline{u}|, \tag{10}$$

where ν_t is the eddy viscosity, C_s is the Smagorinsky constant, and δx is the grid spacing. While there are no trainable networks in this model, C_s can be optimized via hand tuning or gradient descent.

The dynamic Smagorinsky model [50, 51] is similar to the constant model, however the term $(C_s\delta x)^2$ is determined from the state of the system. Studies have shown that C_s is not constant and may be variable with different flow characteristics [52]. The model uses a second test filter, denoted by $\sim$ with filter scale $\tilde{\delta}x$. We fix the filter ratio $\kappa = \tilde{\delta}x/\delta x = 2$. Specifically,

$$(C_s\delta x)^2 = \frac{\langle HM\rangle}{\langle M^2\rangle}, \tag{11}$$

$$H = \nabla(\widetilde{\overline{u}^2}/2) - \nabla(\widetilde{\overline{u}^2}/2), \tag{12}$$

$$M = \kappa^2\nabla(|\nabla\widetilde{\overline{u}}|\nabla\widetilde{\overline{u}}) - \nabla(\widetilde{|\nabla\overline{u}|\nabla\overline{u}}), \tag{13}$$

where $\langle\rangle$ denotes averaging over the spatial domain.

The rest of the models in this section contain trainable networks in functional forms that rely on varying degrees of assumptions typically used in closure analysis. At one extreme, we have model *resnet*, which uses the classical ResNet architecture [53] to predict the coarse-grained flow field at the next time step:

$$(\overline{u}_{t+\Delta t}, \eta_{t+\Delta t}) = (\overline{u}_t, \eta_t) + f_\theta(\overline{u}_t, \eta_t, \nu; x)\Delta t, \tag{14}$$

$$\eta_0 = g_\theta(\overline{u}_0; x), \tag{15}$$

where Δt is a coarse time step. This model makes no assumptions about the physical description of the system, including the fact that it may be described as a PDE. The closure variable η is used only to lift the model into a higher-dimensional space governed by the neural network. The network may choose to learn $\eta_t = 0$, or it may leverage η in the computation of $\overline{u}_t$. Projecting or transforming an input into a

higher-dimensional subspace is a common practice in ML, where a large latent space of hidden channels eases learning. This is a core component of augmented neural ODEs [54], which have shown to generalize better than standard neural ODEs.

anode, an augmented neural ODE, is a small modification to *resnet* that assumes an ODE inductive bias. The model is given by:

$$\dot{\overline{u}}, \dot{\eta} = f_\theta(\overline{u}, \eta, \nu; x) \tag{16}$$

$$\eta_0 = g_\theta(\overline{u}_0; x), \tag{17}$$

where f_θ explicitly describes the RHS of an ODE governing the discretized flow field. The dynamics of $\overline{u}$ and η are determined purely by the network f_θ with no additional terms. This involves only a small change in the code of *resnet*, changing the solver from explicit Euler to an adaptive scheme.

Filtering the governing equations as in equation (4) introduces an additional closure term to the equations. Model *direct* models the entirety of this term directly as an additive correction:

$$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \eta \tag{18}$$

$$\eta = f_\theta(\nabla\overline{u}, \nu; x). \tag{19}$$

Model *transport-I* leverages the fact that the closure term can be written as the divergence of an unknown subgrid stress. Here, η represents this stress analogue in 1D, which is additionally transported with another dynamical equation:

$$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla\eta \tag{20}$$

$$\dot{\eta} = \alpha \cdot \nu\nabla^2\eta - \beta \cdot \overline{u}\nabla\eta \tag{21}$$

$$\alpha, \beta = f_\theta(\nabla\overline{u}, \nabla\eta, \nu; x) \tag{22}$$

$$\eta_0 = g_\theta(\overline{u}_0, \nu; x). \tag{23}$$

The coefficients of the dissipative and convective terms in the η transport equation are given by neural networks.

Model *transport-II* uses the Boussinesq hypothesis that the subgrid stress has a linear relationship with the strain in the flow. η represents an eddy viscosity analogue:

$$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\eta\nabla\overline{u}) \tag{24}$$

$$\dot{\eta} = \alpha \cdot \nu\nabla^2\eta - \beta \cdot \overline{u}\nabla\eta \tag{25}$$

$$\alpha, \beta = f_\theta(\nabla\overline{u}, \nabla\eta, \nu, \delta x; x) \tag{26}$$

$$\eta_0 = g_\theta(\overline{u}_0, \nu; x). \tag{27}$$

Model *transport-I-p* is nearly identical to *transport-I*, however the η transport equation has an additional production term γ:

$$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla\eta \tag{28}$$

$$\dot{\eta} = \alpha \cdot \nu\nabla^2\eta - \beta \cdot \overline{u}\nabla\eta + \gamma \tag{29}$$

$$\alpha, \beta, \gamma = f_\theta(\nabla\overline{u}, \nabla\eta, \nu; x) \tag{30}$$

$$\eta_0 = g_\theta(\overline{u}_0, \nu; x). \tag{31}$$

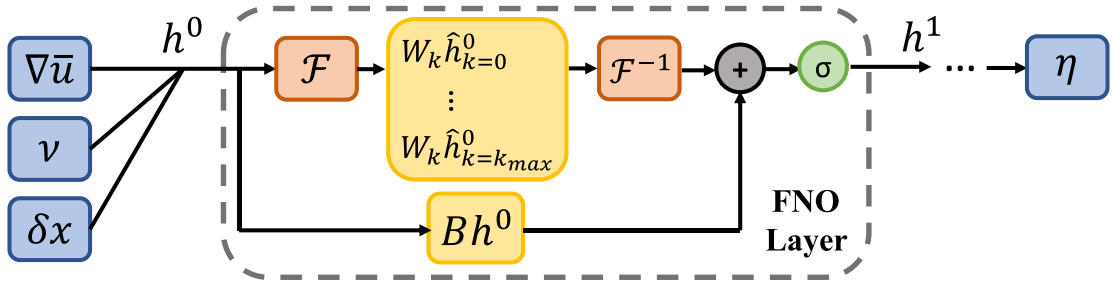

Figure 2. An example schematic of the FNO neural network architecture for the function f_θ in *smag-ml*. At each time step, the network takes as input $\nabla\bar{u}$, ν, and δx, computing the η field that corresponds to a spatially varying subgrid term. Within the network, there are two FNO layers that include a local linear bias from matrix B and a transformation of the Fourier modes $\hat{h}_k$ with matrix W_k. These are summed and passed through a non-linearity to the next layer.

Model *transport-II-p* is equivalent to *transport-II* except for the additional production term:

$$\dot{\bar{u}} = \nu\nabla^2\bar{u} - \bar{u}\nabla\bar{u} + \nabla(\eta\nabla\bar{u}) \tag{32}$$

$$\dot{\eta} = \alpha\cdot\nu\nabla^2\eta - \beta\cdot\bar{u}\nabla\eta + \gamma \tag{33}$$

$$\alpha,\beta,\gamma = f_\theta(\nabla\bar{u},\nabla\eta,\nu;x) \tag{34}$$

$$\eta_0 = g_\theta(\bar{u}_0,\nu;x). \tag{35}$$

The last model, *smag-ml*, attempts to improve on the standard constant Smagorinsky model by learning a spatially varying Smagorinsky constant, computed via neural network:

$$\dot{\bar{u}} = \nu\nabla^2\bar{u} - \bar{u}\nabla\bar{u} + \nabla(\nu_t\nabla\bar{u}) \tag{36}$$

$$\nu_t = (\eta\delta x)^2|\nabla\bar{u}| \tag{37}$$

$$\eta = f_\theta(\nabla\bar{u},\nu,\delta x;x). \tag{38}$$

All of the models used in our analysis are tabulated in table 5 of the appendix.

3.4. Neural network architecture

To leverage our choice of pseudospectral discretization, we use Fourier neural operators (FNOs) as our network architecture [55]. All functions with subscript θ in the previous section correspond to an FNO architecture. FNOs combine pointwise in real space and pointwise in Fourier space linear operations with pointwise nonlinearities in real space to enable learning of a large suite of resolution-invariant function classes, including linear and nonlinear operators.

Given an input feature $h^n \in \mathbb{R}^{N\times c_{\text{in}}}$, where c_{in} is the number of channels or scalar fields, an FNO layer is defined as:

$$h^{n+1} = \sigma(Bh^n + \mathscr{F}^{-1}(W_k\mathscr{F}(h^n)_k)), \tag{39}$$

where $B \in \mathbb{R}^{c_{\text{in}}\times c_{\text{out}}}$ is a 2-dimensional tensor that is contracted along c_{in} identically pointwise in real space and $W \in \mathbb{R}^{k_{\max}\times c_{\text{in}}\times c_{\text{out}}}$ is a three-dimensional tensor that is contracted along c_{in} pointwise in Fourier space for each wavemode k up to $k_{\max}$. Therefore, the Fourier transform of the input is truncated at $k_{\max}$. The output of these operations are summed and a pointwise nonlinearity σ is applied. The parameters of the layer are given by the tensors W and B. Figure 2 illustrates the algorithm for $\eta = f_\theta(\nabla\bar{u},\nu,\delta x;x)$ in the *smag-ml* model.

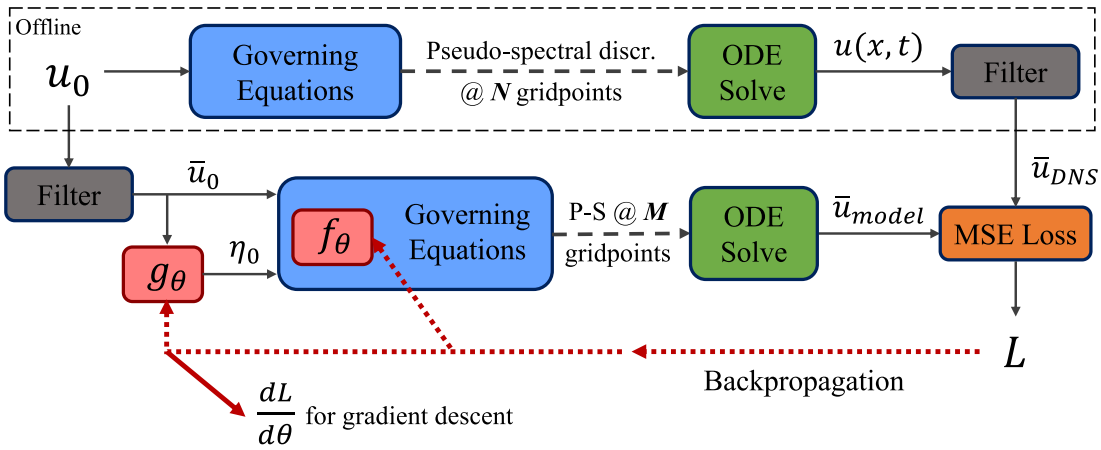

Figure 3. A schematic of the training procedure for learning the parameters in the data-driven models. An initial velocity field is evolved (offline) according to the governing equations on a DNS-resolving grid and filtered to generate ground truth data. During the optimization loop, the same field is filtered and evolved with the data-driven model to compute an estimate for the filtered field. The gradients are backpropagated through the ODE solve and spatial discretization to update the parameters.

3.5. Optimization

Network parameters are optimized with respect to the loss function:

$$L = \sum_{i=1}^{T}\sum_{j=1}^{M} \frac{(\bar{u}_{\text{model}}(x_j,t_i) - \bar{u}_{\text{DNS}}(x_j,t_i))^2}{M*T} \tag{40}$$

$$\bar{u}_{\text{DNS}} = G \star u_{\text{DNS}}, \tag{41}$$

where u_{DNS} is the ground truth velocity field resolved on a grid of N points, $\bar{u}_{\text{DNS}}$ is filtered using sharp low-pass filter G with cutoff $k = M/2$, x_j correspond to the M collocation points on the coarse grid, and t_i correspond to the T timesteps in the discretized solution trajectory.

Optimization is achieved with the gradient-based optimizer Adam, shown to be effective for deep learning applications [56]. Gradients are computed with the AD platform available in *Pytorch* via reverse-mode autodifferentiation, or backpropagation [57]. We show a schematic of the training procedure in figure 3.

Most of the operations in our solution algorithm consist of basic linear operations and pointwise nonlinearities, operations that have formed the foundation of classical multi-layer perceptron ML architectures. Deep learning libraries have readily supported backpropagation through these operations for many years and thus we do not discuss implementation details further. However, we highlight two slightly more complex components of the algorithm corresponding to aspects of the spatial and temporal discretization and discuss their gradient computation.

3.5.1. Gradients of the Fourier transform

In reverse-mode gradient accumulation [58], we wish to compute vector-Jacobian products (VJPs) [59, 60] of each function f in the algorithm:

$$\text{VJP} = \mathbf{v}^T f'(\mathbf{x}), \tag{42}$$

given the cotangent vector $\mathbf{v}$, input $\mathbf{x}$, and Jacobian $f'(\mathbf{x})$. The VJP can then be passed to the previous function in the computational graph to compute the next VJP.

We leverage spectral basis expansion to compute spatial derivatives in the flow and inside the FNO layers. This is implemented with a discrete Fourier transform (DFT) operation. Since the DFT is ultimately a linear operation

$$\mathscr{F}_{\text{DFT}}(\mathbf{x}) = \mathbf{F}\mathbf{x}, \tag{43}$$

with DFT matrix $\mathbf{F}$, the Jacobian of the transform is simply the matrix $\mathbf{F}$. To compute the VJP, we can equivalently perform matrix-vector multiplication with the adjoint operator $\mathbf{F}^*$. Since $\mathbf{F}^* = N\mathbf{F}^{-1}$, we can compute the VJP as an inverse Fourier transform multiplied by the size of the signal. The benefit of this approach comes from the ability to implement fast Fourier transform [61] (FFT) algorithms during the backwards pass as well, instead of constructing the entire DFT matrix explicitly.

3.5.2. Propagating gradients through time integration

Time evolution of the velocity field is accomplished by solving the ODE resulting from spatial discretization of the field using an arbitrary integration method. A naive implementation of AD would propagate gradients through each of basic operations in the solver. Depending on the solver tolerance, order, and trajectory length, this may result in a very large number of operations, and all intermediate steps must be saved in memory for AD to work. In our experiments, backpropagating through even moderately sized trajectory lengths, networks, and batch sizes could exceed the 16GBs of memory on our NVIDIA V100 GPU using naive autograd.

Therefore, a more memory efficient approach is desired. We use the adjoint method in time, which has been proposed for use in ML applications by Chen *et al* [26]. The forward ODE system is given by the evolution of the discrete Fourier modes:

$$\frac{\mathrm{d}\hat{\mathbf{u}}_k(t)}{\mathrm{d}t} = f(\hat{\mathbf{u}}_k(t), \theta), \tag{44}$$

where θ represents any learnable parameters in the model. The loss can be written as an integral:

$$L = \int_{t_0}^{t_T} l(\hat{\mathbf{u}}_k(t))\mathrm{d}t. \tag{45}$$

The adjoint method produces a second ODE system that evolves the adjoint variable λ backwards in time, given by the equation:

$$\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} = -\lambda(t)^T \frac{\partial f(\hat{\mathbf{u}}_k(t), \theta)}{\partial \hat{\mathbf{u}}_k} + \frac{\partial l(\hat{\mathbf{u}}_k(t))}{\partial \hat{\mathbf{u}}_k}, \tag{46}$$

and an integral:

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_{t_T}^{t_0} \lambda(t)^T \frac{\partial f(\hat{\mathbf{u}}_k(t), \theta)}{\partial \theta}\mathrm{d}t, \tag{47}$$

to compute the parameter sensitivities. The RHS of the equations involve VJPs that are efficiently evaluated with AD. The adjoint system can again be solved with an arbitrary solver, potentially even different than the forward solver. Specifically, we use the interpolated adjoint method, which reduces the memory footprint by constructing an interpolation of the forward solution, which appears in the adjoint equations. During the forward pass, the solution is saved at a selected few timesteps and reused in the adjoint computation via interpolation during the backwards pass.

4. Experiments and results

To assess the quality and practicality of our proposed models, we seek to study model performance across a range of Burgers solutions.

4.1. Viscosity generalization and temporal stability

In this experiment, we examine model generalizability with respect to initial conditions of the velocity field and the viscosity parameter ν. In addition, we investigate stability of the models by extrapolating in time relative to the training data.

The models are trained using the optimization procedure described in the previous section on a ground truth dataset of 100 Burgers solutions. We vary the initial conditions of the velocity field by randomizing the Fourier modes of the initial field. The kth Fourier mode is denoted by $\hat{u}_k$. The dataset is generated with the parameters tabulated in table 1.

The neural networks in the trainable models contain two FNO layers with $k_{\max} = 16$, 128 hidden channels, and appropriate input and output sizes. The first layer uses a rectified linear unit (ReLU) nonlinearity while the second uses the identity. The models are trained for 500 epochs using the Adam optimizer with learning rate varying from 10^{-3} to 10^{-1} depending on the model. Generally, models such as *smag-ml* that are more constrained by inductive biases required larger learning rates.

Model performance is gauged using the mean-squared error (MSE) of the predictions relative to the ground truth solution on an unseen test set. We leverage two test sets to assess generalizability across viscosities and stability in time. The first test set consists of 25×25 solutions—25 discrete viscosities in the range of $10^{-4.5} - 10^{-2}$ and 25 trajectories, or different initial conditions, per viscosity. We compute the model error for each prediction and plot the average ensemble error as a function of viscosity.

Table 1. Parameters and their distributions used to generate the training set for the first experiment, including the viscosity ν, the Fourier modes of the initial velocity field $\hat{u}_k$, the timespan for integration t, the DNS grid resolution N, and the coarse grid resolution M.

Parameter	Range
ν	$[10^{-4}, 10^{-3})$
$\hat{u}_k$	$([-0.5, 0.5) + [-0.5, 0.5)i)\, e^{-0.5k}$, $k = 0..32$
t	$[0, 3)$
N	4096
M	64

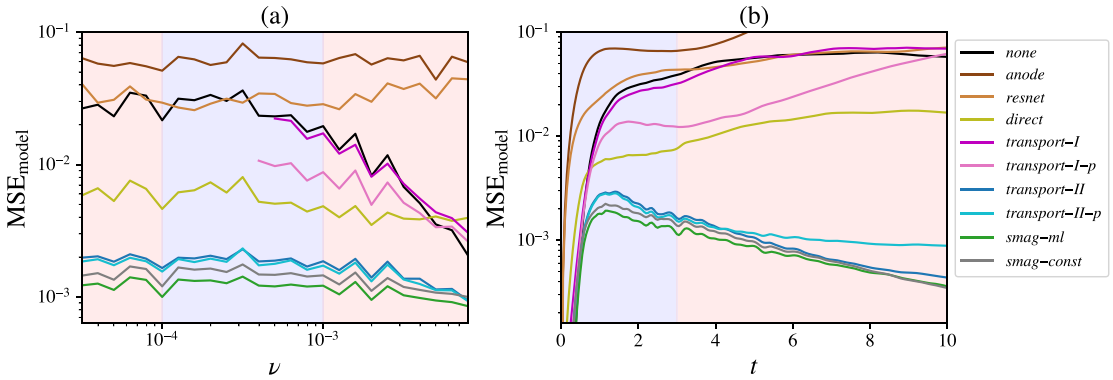

Figure 4. Blue shaded regions indicate the section of the plot corresponding to the training regime, while red shading indicates a test regime. (a) Plot of the ensemble mean-squared error relative to the ground truth DNS for each model as a function of viscosity. All models generally improve over the no closure baseline except *anode* and *resnet*, but only *smag-ml* improves of the constant Smagorinsky model. (b) Ensemble MSE for each model as a function of time. The inclusion of a production term gives rise to anomalous behavior in the extrapolation region.

The second test set contains 25 trajectories at $\nu = 10^{-3}$. Here, we evolve the trajectories between $t = [0, 10)$, over 3 times longer than the training set, to see how the models behave outside of the training regime. The average error across all trajectories is plotted as a function of time.

Two non-trainable models *none* and *smag-const* provide some bounding benchmarks for model comparison. In figure 4(a), at the upper end, the *none* model with no closure results in high error at low viscosities, where the dissipative length scales are small, with progressively lower error at higher viscosities where the dissipative length scales become larger. An effective closure model should at the very least perform better than this, otherwise no additional physics have been learned to improve over the baseline governing equations solved on a coarse grid. At the lower end, the *smag-const* model is a well-known closure model that has been shown to perform well in a wide range of viscosities. An ideal model would produce lower loss than this baseline, indicating that the neural closure functional form can show improvements over classical closure models from theory.

Model performance is delineated well by model accuracy relative to the baselines in figure 4(a). Model *anode* has the highest loss, greater than the *none* baseline, thus showing no improvement over the governing equations. Given that the *anode* model contains no inherent physics or structural form to the PDE, this result is unsurprising as the model must learn both the governing dynamics and any closure approximations. Despite *resnet* having a similar architecture to *anode*, *resnet* can achieve marginally better accuracy, although it is still no better than the *none* baseline. Theoretically, *anode* should be able to learn the same optima as *resnet*, since *anode* has the same functional form as *resnet*, except solved using an adaptive time-stepper. However, actually reaching those optima can be challenging during optimization due to the ODE inductive bias.

Models *transport-I* and *transport-I-p* show marginal improvement over *none*, with *transport-I-p* the better of the two. These models were particularly difficult to train, producing very stiff ODEs that made integration difficult. This is apparent in the fact that some of the viscosities are not shown as the stiffness of those systems led to intractable solutions. We hypothesize that this could be due to overfitting the training set, since all of the training samples could be solved in a reasonable amount of time.

Model *direct* lies firmly between the two benchmarks. Accuracy is relatively uniform with little dependency on viscosity. Thus, while *direct* consistently improves over *none* at low viscosities, the advantage decreases with increasing viscosity, becoming a disadvantage at the highest viscosities.

Table 2. Table of evaluation times for the models on the second test set. Times are normalized by the *none* model, which is the cheapest differential equation-based model to compute.

Model	Relative evaluation time
none	1.0
anode	2.54
resnet	0.37
direct	2.97
transport-I	3.36
transport-I-p	3.40
transport-II	3.48
transport-II-p	3.52
smag-ml	3.24
smag-const	1.36

Models *transport-II* and *transport-II-p* have accuracies comparable to *smag-const* and perform almost identically to each other within the viscosity test. Generally, we see a consistent positive bias over the baseline closure benchmark that diminishes only at the highest viscosities.

Model *smag-ml* is the only model with lower error than *smag-const* everywhere in figure 4(a). Given that the model leverages the functional form of the classical Smagorinsky model and only adds a spatial dependency, again this result is perhaps unsurprising. However, it is promising that the improvements persist beyond the training region.

Overall, besides models *none*, *transport-I*, and *transport-I-p*, accuracy is mostly independent of the viscosity within this range. While models have been trained on viscosities between 10^{-4} and 10^{-3}, they can be used on viscosities well outside this range, with little to no erroneous behavior outside the training regime.

Figure 4(b) shows model accuracy on the second test set. The blue region indicates the time span the model was trained on, while the red region shows the extrapolation range. The model hierarchy by accuracy is generally preserved here, i.e. *anode* performs the worst, *smag-ml* the best, and *transport-II* and *transport-II-p* are comparable to *smag-const*.

However, the stability test highlights the effect of the production term in the models. The models largely have a monotonically decreasing slope in the error over time, except for *anode*, *transport-I-p*, and *transport-II-p*, which display aberrant behavior in the extrapolation range. The inclusion of the production term results in improvements within the training time regime, and in the case of *transport-I* and *transport-I-p*, the difference is significant. Outside of this range, the production term can hinder the solution and lead to growing errors. Deviations between *transport-II* and *transport-II-p* can be seen at $t > 5$, and error from *transport-I-p* grows to match *transport-I* at $t = 10$.

In addition, there is a clear separation of models *transport-II*, *transport-II-p*, *smag-ml*, and *smag-const* with regards to the rest of the models tested. These four models show decreasing MSE after $t \approx 1$, while the rest of the models' error grows, indicating potential instabilities. The origin of the disparity stems from the usage of the Boussinesq hypothesis in the better-performing four models, enforcing that the viscous forces are by design dissipative. Since there is no external forcing, both the DNS and the Boussinesq models eventually decay to zero, as does the MSE. This physical prior is imposing some known behavior (dissipative system), while still leaving room for the neural network to learn the closure variable, resulting in significantly more accurate models.

Table 2 shows the model evaluation times relative to *none*. All of the models except for *resnet* are more expensive than this baseline, which is expected considering that the models require more operations and Fourier transforms to compute than the governing dynamics. We found that the model evaluation times have limited correlation with the number of operations needed at each time step, and are more heavily influenced by the stiffness of the resulting ODE system, given the adaptive integration scheme. Thus, *resnet* is much more efficient than other models because it requires the fewest function evaluations, coinciding with the fixed Euler time steps.

We also show training curves for this experiment in figure 5, including the train loss and training time as a function of epoch. We point out the train loss of *resnet* in figure 5(a), which is vastly different than the test loss in figure 4, different behavior from the other models tested. We believe the use of differential equation inductive biases in the other models close the learning gap between training and testing data distributions, promoting data-efficient and generalizable models. Figure 5(b) clearly demonstrates the effect of ODE stiffness. Since the number of operations per function evaluation in each model is fixed, one might expect the training time to be linear with epoch. However, as models like *transport-I* and *direct* evolve in training, the curves become steeper, indicating more function evaluations required for time integration.

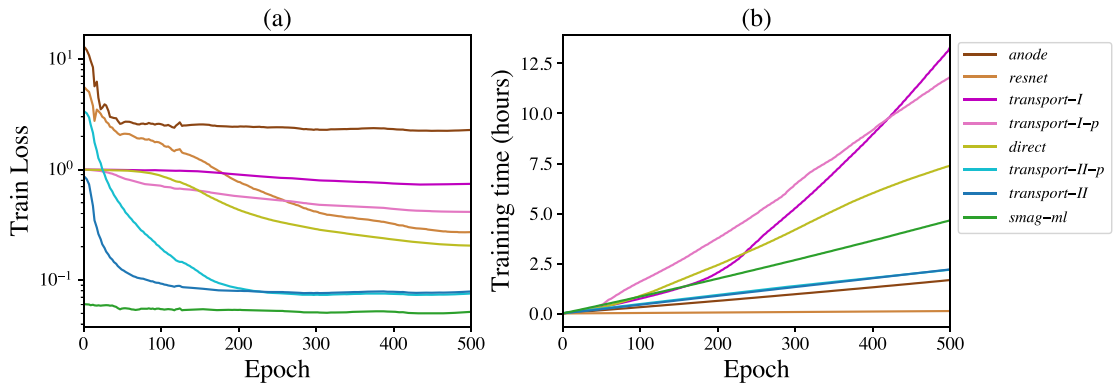

Figure 5. Training curves for the models in the first experiment. (a) MSE loss of the models, normalized by *none*, as a function of epoch. There is significant variability between the models in time to convergence and magnitude of the overall change in loss. (b) Training time as a function of epoch. For some models, the computational cost of evaluation remains constant, leading to linear behavior. For others, the computational cost increases with training iteration, demonstrating non-linear behavior with a positive second derivative.

Table 3. Parameters and their distributions used to generate the training set for the second experiment, including the viscosity ν, the Fourier modes of the initial velocity field $\hat{u}_k$, the timespan for integration t, the DNS grid resolution N, and the coarse grid resolutions M.

Parameter	Range
ν	$[10^{-4}, 10^{-3})$
$\hat{u}_k$	$[-0.05, 0.05) + [-0.05, 0.05)i$, $k = 0..32$
t	$[0, 6)$
N	8192
M	64, 128, 256, 512

4.2. Resolution invariance

In the previous experiment, we have shown that many of the models generalize well over a wide range of viscosities and time. In this experiment, we wish to examine our approach with regards to the coarse-grain scale. In the previous experiment, we fixed $M = 64$, here we will vary M and thus the cut-off of the low-pass filter, $k = M/2$.

Again, we generate a training dataset of 100 Burgers solutions. Since we desire for all the cutoffs $M/2$ to lie in the inertial region of the flow spectrum, we slightly modify the distribution of the initial condition, the time span for integration, and the ground truth grid density N, realizing a wider inertial range. We divide the set into 4 batches with $M = 64, 128, 256, 512$. The dataset parameters are tabulated in table 3.

We train the two best performing models from the previous experiment, *transport-II* and *smag-ml*, as well as two modifications to the *smag-ml* model to understand the impact of certain neural network architectural choices. These models are trained for 200 epochs, which was sufficient for convergence, again using the ADAM optimizer with learning rate varying from 10^{-3} to 10^{-1} depending on the model.

Specifically, we add two models, *smag-ml-small* and *smag-ml-local*. *smag-ml-small* reduces the number of hidden channels in the network to 2 from 128. Since the FNO architecture requires each of the channels to be Fourier transformed and inverse transformed at each timestep, the size of the hidden layer has a large impact on the computational efficiency of the model. Therefore, we wish to see how small the model can be made without sacrificing accuracy. *smag-ml-local* switches the FNO layers for standard linear layers resulting in a purely local closure correction. The FNO is a non-local operator given the pointwise multiplication in Fourier space. Replacement with a linear layer gives only pointwise multiplication by a linear operator in real space. Thus, we can compare the efficacy of a non-local functional form with a local functional form.

To test the models, we generate a new test set that includes variation in M. Here, we look at a set of 13×25 trajectories—13 discrete M's in the range of 2^6–2^9, including interpolation between the training discretizations, and 25 samples per M. The ν and $\hat{u}_k$ values are drawn from the same distributions as the training set, and $t = [0, 20)$.

Figure 6 shows the MSE of the models normalized by the constant Smagorinsky baseline. Since the training dataset encompasses variations in both ν and M, we train the Smagorinsky constant via gradient descent on the same training dataset as the models to obtain the optimal constant instead of hand-tuning. The training procedure converged clearly to a value of $C_s = 0.4$ as shown in figure 7. We also include a

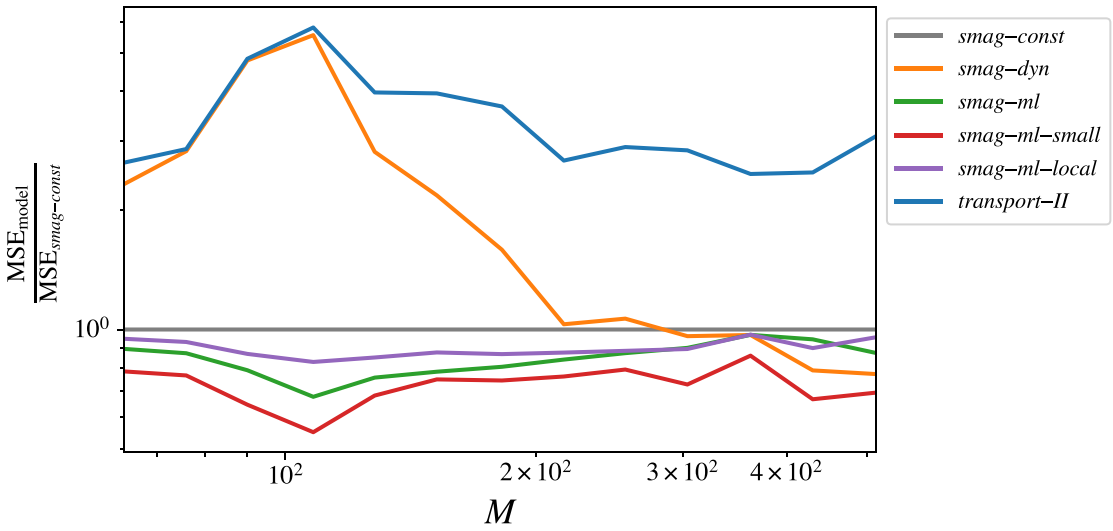

Figure 6. The ensemble mean-squared error of various models normalized by the MSE of the constant Smagorinsky model, one of the baseline we hope to improve over. The adaptive ML Smagorinsky models can reduce the error up to 40%, with *smag-ml-small* demonstrating the best performance. *smag-dyn* can only achieve lower error than *smag-const* at high resolutions, where the inertial region is more fully resolved.

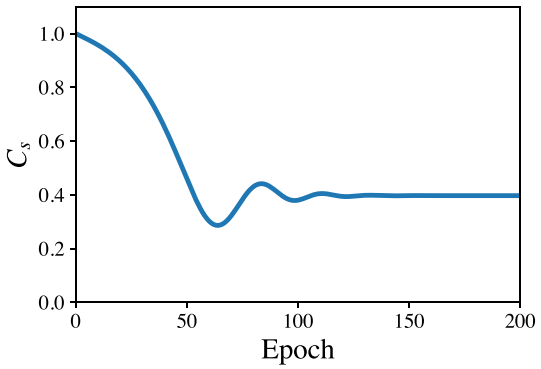

Figure 7. Convergence of the Smagorinsky constant C_s to 0.4 as optimized by gradient descent on the training dataset.

comparison to the non-trainable *smag-dyn* baseline, given that most of the models in this experiment are Smagorinsky model derivatives.

Similar to the previous experiment, figure 6 shows that model *transport-II* results in greater error than *smag-const*, while the adaptive Smagorinsky approaches can improve over the constant model. Even when the ideal C_s is fit to the data, the spatiotemporally varying models outperform the baseline. Within the adaptive trainable Smagorinsky models, the local correction can improve over the constant baseline at all resolutions, though not as much as the other two models. The two non-local models generally perform better than all others, which fits with our understanding that turbulence is a non-local phenomenon. However, surprisingly, the *smag-ml-small* model ultimately shows a universal advantage over both the dynamic and constant Smagorinsky baselines. Comparing the training and validation losses of the models provides some insight. The overparameterized *smag-ml* model overfits the training set resulting in worse performance on the test set. While this may be ameliorated by a larger training dataset, we do not pursue this route and leave this study for future work. Instead, we show that we can learn an effective and computationally efficient correction to the baseline Smagorinsky model with limited training data, due to our differentiable physics approach to end-to-end training of ML models.

Table 4 lists the normalized evaluation times for the models in this experiment. While the FNO-based models are unequivocally more expensive than any of the baselines, *smag-ml-local* provides an interesting compromise between computational cost and accuracy, achieving lower error than the baselines everywhere except for the highest resolutions, where *smag-dyn* improves over *smag-const*.

The training curves for the models are shown in figure 8. From figure 8(a), convergence was achieved in 200 epochs and from figure 8(b), these models did not display the same kind of non-linear behavior in the

Table 4. Table of evaluation times for the models used in the resolution experiment. Times are normalized by the *none* model, which is the cheapest to compute.

Model	Relative evaluation time
none	1.0
smag-const	1.33
smag-dyn	2.35
smag-ml	3.65
smag-ml-small	3.47
smag-ml-local	1.77
transport-II	3.97

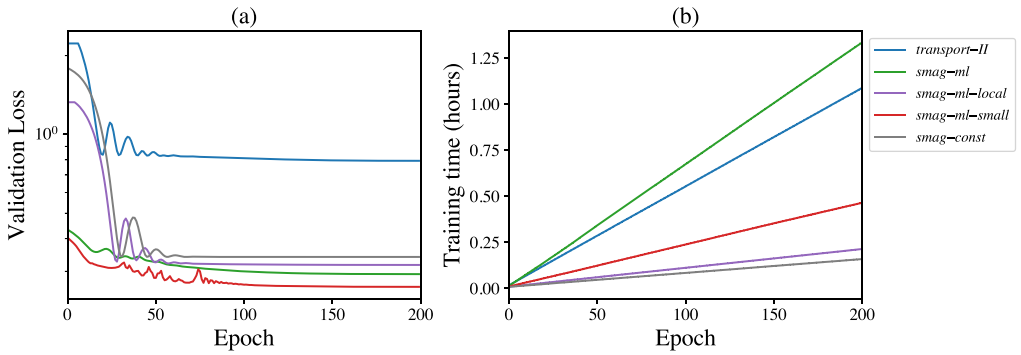

Figure 8. Training curves for the models in the second experiment. (a) MSE loss of the models, normalized by *none*, as a function of epoch. The inductive biases of these models lead to fast convergence. (b) Training time as a function of epoch. Although *smag-ml* and *smag-ml-small* have comparable final evaluation costs, the differences become significant during training, in terms of overall optimization time.

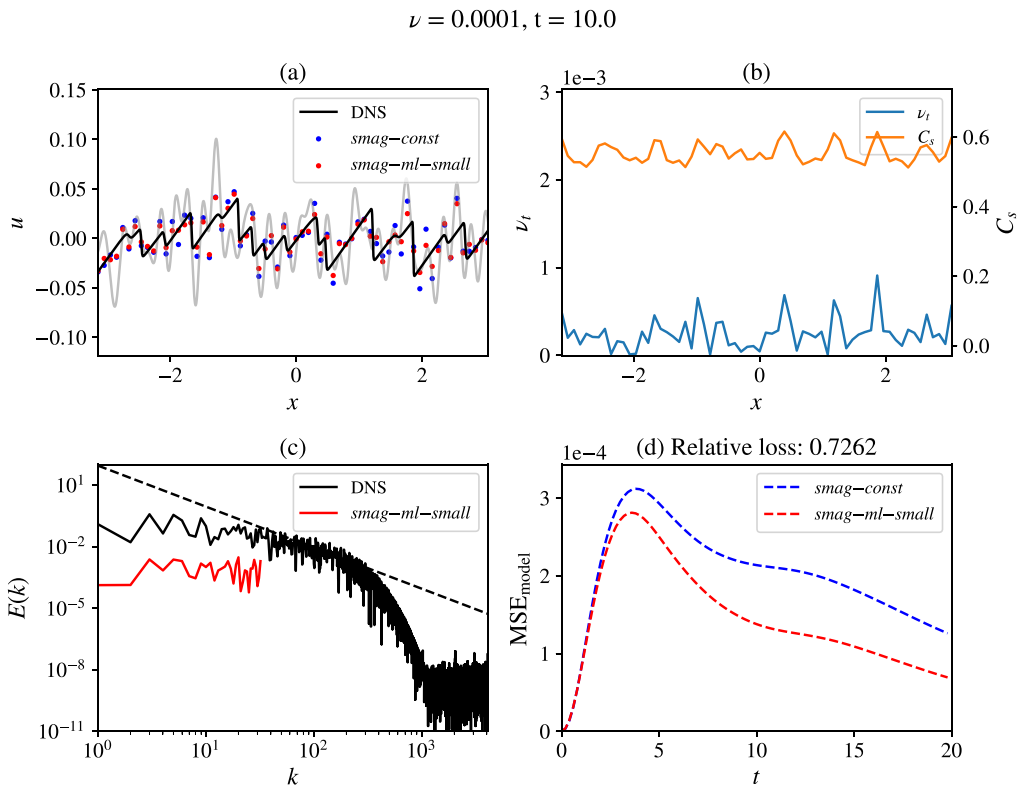

Figure 9. Statistics and snapshots comparing results from closure model *smag-ml-small* to a baseline and DNS with $M = 64$. (a) Sample snapshot of the velocity field from DNS, the *smag-const* baseline, and *smag-ml-small*. (b) Sample snapshot of the closure model eddy viscosity field and spatially varying Smagorinsky constant from *smag-ml-small*. (c) Snapshot of the energy spectra of the sample from DNS and *smag-ml-small*. The cutoff $M/2$ lies in the very upper region of the inertial range. (d) MSE of the model and baseline over time.

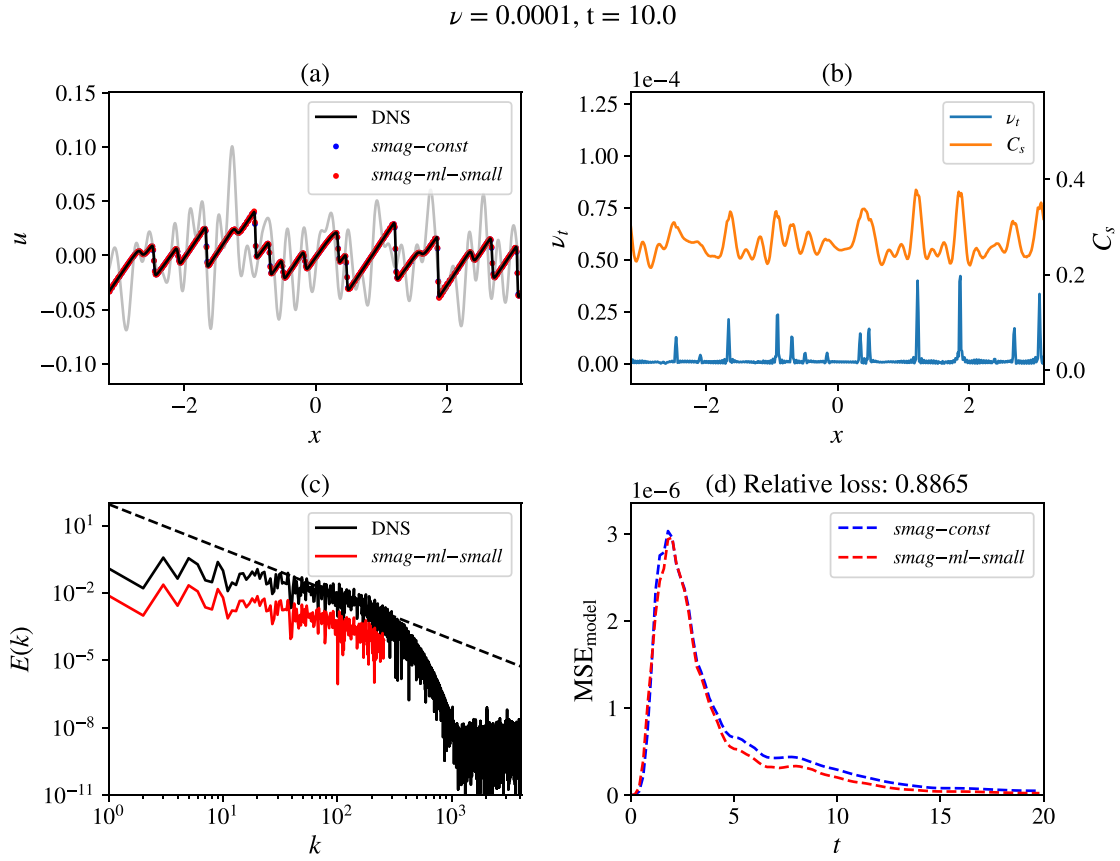

Figure 10. Statistics and snapshots comparing results from closure model *smag-ml-small* to a baseline and DNS with $M = 512$. (a) Sample snapshot of the velocity field from DNS, the *smag-const* baseline, and *smag-ml-small*. (b) Sample snapshot of the closure model eddy viscosity field and spatially varying Smagorinsky constant from *smag-ml-small*. (c) Snapshot of the energy spectra of the sample from DNS and *smag-ml-small*. The cutoff $M/2$ lies very close to the dissipative range. (d) MSE of the model and baseline over time.

training time. *smag-ml-small* was much faster to train than *smag-ml*, despite the comparable evaluation times.

Lastly, we show some examples of predictions from model *smag-ml-small* compared to *smag-const*. Figure 9 displays results from $M = 64$, where the cut-off wavenumber is close to the energy containing region of the spectra. Figure 10 presents results from $M = 512$, where the cutoff wavenumber is on the other side of the inertial region close to the dissipative region. The snapshots of the velocity field in (a) and the Smagorinsky constant and eddy viscosity in (b) are taken at $t = 10$. (c) Compares the DNS and model energy spectra at $t = 10$, where the cutoff wavenumber is indicated by the model spectrum. (d) Illustrates the MSE loss of the model *smag-ml-small* and *smag-const* over time, including the relative loss of the model to the constant Smagorinsky baseline.

5. Conclusions

In this work, we systematically investigate the efficacy of various ML based closure models for Burgers turbulence using an end-to-end learning framework. We aim to develop a better understanding of how the functional form of the data-driven closure impacts its ability to accurately model subgrid scales, and seek to design a data-driven closure that is broadly applicable and ideally outperforms baseline closure models. Our models are discretization independent and thus defined by their PDE form and neural network hyperparameters. Models are trained in an end-to-end fashion, meaning the entire closure PDE is solved with a pseudo-spectral method at each iteration of the optimization procedure. Differentiable programming closes the gap between typical *a priori* learning and separate *a posteriori* testing that is common in ML closure modeling today. Given a differentiable solution algorithm, models can be trained via gradient descent directly on an *a posteriori* loss function. This approach allows for great flexibility in model design. Just as neural network hyperparameters can be easily tuned and adjusted, the PDE form of the closure can be modified to incorporate various degrees of known physics or physical assumptions. Once the high level PDE

has been defined, the system is solved and evaluated on the loss function, the MSE of the velocity field relative to a ground truth DNS simulation.

We assess the models on a wide range of Burgers systems to determine their generalizability. Given the spread of Reynolds numbers encountered in flow problems and the need for a closure model that can accommodate this range, we test our models' performance on unseen Burgers systems with viscosities spanning 2.5 orders of magnitude, including interpolation of the training viscosities and extrapolation outside of this distribution. We also check the stability of the models by extrapolation in time.

We find that models that incorporate more physical assumptions result in lower errors on the test set than those without such inductive biases. For example, use of the Boussinesq hypothesis in *transport-II* produces significantly better approximations than *transport-I*, which only models a stress analogue directly. The *resnet* and *anode* models have no assumptions of the PDE form, meaning they must learn all the complex dynamics of the Burgers system with limited training data, resulting in unsuccessful models. Increasing the size of the training set may have allowed for improved learning, however we are specifically interested in data-efficient models given the cost of generating ground truth DNS simulations. At the other extreme, we leverage all the assumptions of the baseline Smagorinsky model and only apply a small modification to learn a spatially varying Smagorinsky constant. Here, we are able to show universal improvement over the baseline Smagorinsky models at all viscosities.

We take our analysis further by demonstrating the resolution invariance of our closure approach. Subgrid closures model flow scales smaller than their cutoff wavenumber, $k = M/2$ in our case. This cut-off should lie in the inertial region of the energy spectra, but where exactly within the region it lies may be variable depending on the coarse grid resolution and system parameters. A universal closure model should be able to adapt to different cutoff wavenumbers, which impacts the range of energy scales it must model. In our second experiment, we examine the adaptive trainable Smagorinsky models in more detail with regards to different coarse grid resolutions. We train the model on four resolutions ranging from cutoffs very close to the dissipative region to cutoffs close to the energy containing region. We show that the model can continue to outperform the constant model at all resolutions, including interpolation between the training resolutions. To the best of our knowledge, this is the first use of a differentiable physics based closure on varying uniform grids.

Finally, we show the impact of two neural network architectural choices—the size of the hidden layer in the network and the non-locality of the FNO functional form. It is typical in ML approaches to create very large networks with 10 s of thousands of parameters to create very expressive networks that can act as universal approximators. However, we have included a great deal of inductive bias in our model simply by leveraging the Smagorinsky form, meaning our network only has to learn a small correction to see improvement. We demonstrate that a small network of 520 parameters is enough to learn an effective correction, even showing lower loss than the larger model which can overfit the small training set. Lastly, we see the impact of the non-local FNO by testing a model with linear layers. While indeed the local adaptive model improves over the baseline, the advantage is not as significant as the non-local model, although its computational efficiency is greater.

Closure modeling remains a significant challenge in the fluid dynamics community. ML is enabling a new class of data-driven models, but their performance on arbitrary flows can be difficult to characterize. We use the Burgers system as a test bed for developing subgrid models in a differentiable framework, training on an *a posteriori* loss and testing on a large variety of Burgers systems. We show that a simple correction to existing closure models using data-efficient neural networks can result in better performance.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgment

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1745016 awarded to VS. The authors from CMU acknowledge the support from the Technologies for Safe and Efficient Transportation University Transportation Center, and Mobility21, A United States Department of Transportation National University Transportation Center. This work was supported in part by Oracle Cloud credits and related resources provided by the Oracle for Research program. RM is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

Appendix

Table 5. Model descriptions.

Name	Equations	Time-stepper	Layer type	Hidden channels	Notes
none	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u}$	Tsit5	N/A	N/A	
smag-const	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\nu_t\nabla\overline{u})$, $\nu_t = (C_s\delta x)^2\lvert\nabla\overline{u}\rvert$	Tsit5	N/A	N/A	This model contains no neural networks, but C_s is a potentially trainable parameter.
smag-dyn	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\nu_t\nabla\overline{u})$, $\nu_t = (C_s\delta x)^2\lvert\nabla\overline{u}\rvert$, $(C_s\delta x)^2 = \frac{\langle HM\rangle}{\langle M^2\rangle}$, $H = \nabla(\widetilde{\overline{u}}^2/2) - \nabla(\widetilde{\overline{u}^2}/2)$, $M = \kappa^2\nabla(\lvert\nabla\widetilde{\overline{u}}\rvert\nabla\widetilde{\overline{u}}) - \nabla(\widetilde{\lvert\nabla\overline{u}\rvert\nabla\overline{u}})$	Tsit5	N/A	N/A	
resnet	$(\overline{u}_{t+\Delta t}, \eta_{t+\Delta t}) = (\overline{u}_t, \eta_t) + f_\theta(\overline{u}_t, \eta_t, \nu; x)\Delta t$, $\eta_0 = g_\theta(\overline{u}_0; x)$	Euler	FNO	128	
anode	$\dot{\overline{u}}, \dot{\eta} = f_\theta(\overline{u}, \eta, \nu; x)$, $\eta_0 = g_\theta(\overline{u}_0; x)$	Tsit5	FNO	128	
direct	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \eta$, $\eta = f_\theta(\nabla\overline{u}, \nu; x)$	Tsit5	FNO	128	
transport-I	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla\eta$, $\dot{\eta} = \alpha\cdot\nu\nabla^2\eta - \beta\cdot\overline{u}\nabla\eta$, $\alpha, \beta = f_\theta(\nabla\overline{u}, \nabla\eta, \nu; x)$, $\eta_0 = g_\theta(\overline{u}_0, \nu; x)$	Tsit5	FNO	128	
transport-II	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\eta\nabla\overline{u})$, $\dot{\eta} = \alpha\cdot\nu\nabla^2\eta - \beta\cdot\overline{u}\nabla\eta$, $\alpha, \beta = f_\theta(\nabla\overline{u}, \nabla\eta, \nu, \delta x; x)$, $\eta_0 = g_\theta(\overline{u}_0, \nu; x)$	Tsit5	FNO	128	The input δx to f_θ is only used in the resolution study.
transport-I-p	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla\eta$, $\dot{\eta} = \alpha\cdot\nu\nabla^2\eta - \beta\cdot\overline{u}\nabla\eta + \gamma$, $\alpha, \beta, \gamma = f_\theta(\nabla\overline{u}, \nabla\eta, \nu; x)$, $\eta_0 = g_\theta(\overline{u}_0, \nu; x)$	Tsit5	FNO	128	
transport-II-p	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\eta\nabla\overline{u})$, $\dot{\eta} = \alpha\cdot\nu\nabla^2\eta - \beta\cdot\overline{u}\nabla\eta + \gamma$, $\alpha, \beta, \gamma = f_\theta(\nabla\overline{u}, \nabla\eta, \nu; x)$, $\eta_0 = g_\theta(\overline{u}_0, \nu; x)$	Tsit5	FNO	128	
smag-ml	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\nu_t\nabla\overline{u})$, $\nu_t = (\eta\delta x)^2\lvert\nabla\overline{u}\rvert$, $\eta = f_\theta(\nabla\overline{u}, \nu, \delta x; x)$	Tsit5	FNO	128	The input δx to f_θ is only used in the resolution study.
smag-ml-local	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\nu_t\nabla\overline{u})$, $\nu_t = (\eta\delta x)^2\lvert\nabla\overline{u}\rvert$, $\eta = f_\theta(\nabla\overline{u}, \nu, \delta x; x)$	Tsit5	Linear	128	
smag-ml-small	$\dot{\overline{u}} = \nu\nabla^2\overline{u} - \overline{u}\nabla\overline{u} + \nabla(\nu_t\nabla\overline{u})$, $\nu_t = (\eta\delta x)^2\lvert\nabla\overline{u}\rvert$, $\eta = f_\theta(\nabla\overline{u}, \nu, \delta x; x)$	Tsit5	FNO	2	

ORCID iDs

Varun Shankar https://orcid.org/0000-0002-0332-8840
Vedant Puri https://orcid.org/0000-0002-2724-6591
Ramesh Balakrishnan https://orcid.org/0000-0001-8516-5485
Romit Maulik https://orcid.org/0000-0001-9731-8936
Venkatasubramanian Viswanathan https://orcid.org/0000-0003-1060-5495

References

[1] Deville M O, Fischer P F and Mund E H 2002 *High-Order Methods for Incompressible Fluid Flow* vol 9 (Cambridge: Cambridge University Press)
[2] Karniadakis G E 1999 Simulating turbulence in complex geometries *Fluid Dyn. Res.* **24** 343
[3] Moin P and Mahesh K 1998 Direct numerical simulation: a tool in turbulence research *Annu. Rev. Fluid Mech.* **30** 539–78
[4] Kolmogorov A 1941 The local structure of turbulence in incompressible viscous fluid for very large Reynolds' numbers *Akademiia Nauk SSSR Dokl.* **30** 301–5
[5] Coleman G and Sandberg R 2010 *A primer on direct numerical simulation of turbulence - methods, procedures and guidelines* AFM 09/01a School of Engineering Sciences Aerospace Engineering, University of Southampton (available at: http://eprints.soton.ac.uk/id/eprint/66182)
[6] Orszag S A 1970 Analytical theories of turbulence *J. Fluid Mech.* **41** 363–86
[7] Pope S B 2000 *Turbulent Flows* (Cambridge: Cambridge University Press)
[8] Sagaut P 2006 *Large Eddy Simulation for Incompressible Flows: An Introduction* (Berlin: Springer Science and Business Media)
[9] Adams N 2007 Mathematics of large eddy simulation of turbulent flows. by LC Berselli, T. Iliescu & WJ layton. Springer, 2006. 348 pp. 3 *J. Fluid Mech.* **582** 473–5
[10] Qi D and Harlim J 2022 Machine learning-based statistical closure models for turbulent dynamical systems *Phil. Trans. R. Soc.* A **380** 20210205
[11] Layton W J 2014 The 1877 Boussinesq conjecture: turbulent fluctuations are dissipative on the mean flow
[12] Brunton S L, Noack B R and Koumoutsakos P 2020 Machine learning for fluid mechanics *Annu. Rev. Fluid Mech.* **52** 477–508
[13] Vinuesa R and Brunton S L 2021 The potential of machine learning to enhance computational fluid dynamics (arXiv:2110.02085)
[14] Duraisamy K, Iaccarino G and Xiao H 2019 Turbulence modeling in the age of data *Annu. Rev. Fluid Mech.* **51** 357–77
[15] Kochkov D, Smith J A, Alieva A, Wang Q, Brenner M P and Hoyer S 2021 Machine learning–accelerated computational fluid dynamics *Proc. Natl Acad. Sci.* **118** e2101784118
[16] Watt-Meyer O, Brenowitz N D, Clark S K, Henn B, Kwa A, McGibbon J, Perkins W A and Bretherton C S 2021 Correcting weather and climate models by machine learning nudged historical simulations *Geophys. Res. Lett.* **48** e2021GL092555
[17] Vlachas P R, Byeon W, Wan Z Y, Sapsis T P and Koumoutsakos P 2018 Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks *Proc. R. Soc.* A **474** 20170844
[18] Thuerey N, Weißenow K, Prantl L and Hu X 2020 Deep learning methods for Reynolds-averaged Navier-Stokes simulations of airfoil flows *AIAA J.* **58** 25–36
[19] Portwood G D, Nadiga B T, Saenz J A and Livescu D 2021 Interpreting neural network models of residual scalar flux *J. Fluid Mech.* **907** A23
[20] Wang R, Kashinath K, Mustafa M, Albert A and Yu R 2020 Towards physics-informed deep learning for turbulent flow prediction *Proc. 26th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining* (New York: Association for Computing Machinery) pp 1457–66
[21] Weinan E 2020 Machine learning and computational mathematics *Commun. Comput. Phys.* **28** 1639–70
[22] Ling J, Kurzawski A and Templeton J 2016 Reynolds averaged turbulence modelling using deep neural networks with embedded invariance *J. Fluid Mech.* **807** 155–66
[23] Milani P M, Ling J and Eaton J K 2021 Turbulent scalar flux in inclined jets in crossflow: counter gradient transport and deep learning modelling *J. Fluid Mech.* **906** A27
[24] Shankar V, Portwood G, Mohan A, Mitra P, Rackauckas C, Wilson L, Schmidt D and Viswanathan V 2020 Learning non-linear spatio-temporal dynamics with convolutional neural odes *Third Workshop on Machine Learning and the Physical Sciences* (*NeurIPS 2020*)
[25] Rackauckas C, Ma Y, Martensen J, Warner C, Zubov K, Supekar R, Skinner D, Ramadhan A and Edelman A 2021 Universal differential equations for scientific machine learning (arXiv:2001.04385 [cs.LG])
[26] Chen T Q, Rubanova Y, Bettencourt J and Duvenaud D 2018 Neural ordinary differential equations *CoRR* (arXiv:1806.07366)
[27] Farrell P E, Ham D A, Funke S W and Rognes M E 2013 Automated derivation of the adjoint of high-level transient finite element programs *SIAM J. Sci. Comput.* **35** C369–93
[28] Errico R M 1997 What is an adjoint model? *Bull. Am. Meteorol. Soc.* **78** 2577–92
[29] Sigmund O and Maute K 2013 Topology optimization approaches *Struct. Multidiscip. Optim.* **48** 1031–55
[30] Palacios F, Colonno M, Aranake A, Campos A, Copeland S, Economon T, Lonkar A, Lukaczyk T, Taylor T and Alonso J 2013 Stanford university unstructured (su2): an open-source integrated computational environment for multi-physics simulation and design *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* (https://doi.org/10.2514/6.2013-287)
[31] Ramsundar B, Krishnamurthy D and Viswanathan V 2021 Differentiable physics: a position piece (arXiv:2109.07573)
[32] Smits A Marusic I 2013 Wall-bounded turbulence *Phys. Today* **66** 24–30
[33] Thomas L 1953 The stability of plane Poiseuille flow *Phys. Rev.* **91** 780
[34] Roberts M S 2020 Fluid instabilities and transition to turbulence *Computational Overview of Fluid Structure Interaction* ed Ghaedi K, Alhusseny A, Nasser A and Al-Zurf N (Rijeka: IntechOpen) ch 2
[35] Frisch U and Bec J 2000 Burgulence (arXiv:NLIN/0012033)
[36] Falkovich G and Sreenivasan K R 2006 Lessons from hydrodynamic turbulence Technical Report Abdus Salam International Centre for Theoretical Physics
[37] Love M 1980 Subgrid modelling studies with Burgers' equation *J. Fluid Mech.* **100** 87–110
[38] LaBryer A, Attar P and Vedula P 2015 A framework for large eddy simulation of Burgers turbulence based upon spatial and temporal statistical information *Phys. Fluids* **27** 035116
[39] Lesieur M and Metais O 1996 New trends in large-eddy simulations of turbulence *Annu. Rev. Fluid Mech.* **28** 45–82
[40] Boris J P, Grinstein F F, Oran E S and Kolbe R L 1992 New insights into large eddy simulation *Fluid Dyn. Res.* **10** 199
[41] Mullen J S and Fischer P F 1999 Filtering techniques for complex geometry fluid flows *Commun. Numer. Methods Eng.* **15** 9–18
[42] Paul J W L Fischer F and Kerkemeier S G 2008 nek5000 (available at: http://nek5000.mcs.anl.gov)
[43] Layton W J, Pruett C D and Rebholz L G 2010 Temporally regularized direct numerical simulation *Appl. Math. Comput.* **216** 3728–38
[44] Frisch U and Kolmogorov A N 1995 *Turbulence: The Legacy of AN Kolmogorov* (Cambridge: Cambridge University Press)

[45] Schmitt F G 2007 About Boussinesq's turbulent viscosity hypothesis: historical remarks and a direct evaluation of its validity *C. R. Mec.* **335** 617–27
[46] Boyd J, Marilyn T and Eliot P 2000 *Chebyshev and Fourier Spectral Methods* 2nd edn Dover Books on Mathematics (Mineola, NY: Dover Publications)
[47] Tsitouras C 2011 Runge-Kutta pairs of order 5(4) satisfying only the first column simplifying assumption *Comput. Math. Appl.* **62** 770–5
[48] Maulik R and San O 2018 Explicit and implicit LES closures for Burgers turbulence *J. Comput. Appl. Math.* **327** 12–40
[49] Smagorinsky J 1963 General circulation experiments with the primitive equations: I. the basic experiment *Mon. Weather Rev.* **91** 99–164
[50] Germano M, Piomelli U, Moin P and Cabot W H 1991 A dynamic subgrid-scale eddy viscosity model *Phys. Fluids* A **3** 1760–5
[51] Lilly D K 1992 A proposed modification of the Germano subgrid-scale closure method *Phys. Fluids* A **4** 633–5
[52] Smagorinsky J 1993 Some historical remarks on the use of nonlinear viscosities *Large Eddy Simul. Complex Eng. Geophys. Flows* **3** 36
[53] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (https://doi.org/10.1109/CVPR.2016.90)
[54] Dupont E, Doucet A and Teh Y W 2019 Augmented Neural ODEs *Advances in Neural Information Processing Systems* (*Vancouver, Canada*) ed H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox and R Garnett
[55] Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A and Anandkumar A 2020 Fourier neural operator for parametric partial differential equations (arXiv:2010.08895)
[56] Kingma D P and Ba J 2015 ADAM: a method for stochastic optimization *3rd Int. Conf. on Learning Representations* (*ICLR 2015*) (*Conf. Track Proc.*) (*San Diego, CA, USA, 7–9 May, 2015*) ed Y Bengio and Y LeCun
[57] Paszke A *et al* 2019 Pytorch: an imperative style, high-performance deep learning library *CoRR* (arXiv:1912.01703)
[58] Speelpenning B 1980 Compiling fast partial derivatives of functions given by algorithms (available at: https://doi.org/10.2172/5254402)
[59] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L and Lerer A 2017 Automatic differentiation in Pytorch *NIPS 2017 Workshop on Autodiff*
[60] Griewank A and Walther A 2008 *Evaluating Derivatives* 2nd edn (Philadelphia, PA: Society for Industrial and Applied Mathematics) (https://doi.org/10.1137/1.9780898717761)
[61] Frigo M and Johnson S G 2005 The design and implementation of FFTW3 *Proc. IEEE* **93** 216–31