J. Fluid Mech. (2022), *vol.* 949, A25, doi:10.1017/jfm.2022.738

Learned turbulence modelling with differentiable fluid solvers: physics-based loss functions and optimisation horizons

Björn List[1],†, Li-Wei Chen[1] and Nils Thuerey[1]

[1]Departement of Informatics, Technical University of Munich, D-85748 Garching, Germany

(Received 22 December 2021; revised 27 June 2022; accepted 21 August 2022)

In this paper, we train turbulence models based on convolutional neural networks. These learned turbulence models improve under-resolved low-resolution solutions to the incompressible Navier–Stokes equations at simulation time. Our study involves the development of a differentiable numerical solver that supports the propagation of optimisation gradients through multiple solver steps. The significance of this property is demonstrated by the superior stability and accuracy of those models that unroll more solver steps during training. Furthermore, we introduce loss terms based on turbulence physics that further improve the model accuracy. This approach is applied to three two-dimensional turbulence flow scenarios, a homogeneous decaying turbulence case, a temporally evolving mixing layer and a spatially evolving mixing layer. Our models achieve significant improvements of long-term *a posteriori* statistics when compared with no-model simulations, without requiring these statistics to be directly included in the learning targets. At inference time, our proposed method also gains substantial performance improvements over similarly accurate, purely numerical methods.

Key words: machine learning, turbulence modelling

1. Introduction

Obtaining accurate numerical solutions to turbulent fluid flows remains a challenging task, and is subject to active research efforts in fluid dynamics (Argyropoulos & Markatos 2015) and adjacent fields including climate research (Aizinger *et al.* 2015) and the medical sciences (Bozzi *et al.* 2021). Direct numerical simulation (DNS), which attempts to fully resolve the vast scale of turbulent motion, is prohibitively expensive in many flow scenarios and is thus often adverted by using turbulence models. For instance, Reynolds-averaged Navier–Stokes (RANS) modelling has successfully been deployed to complex flow problems such as aircraft shape design and optimisation of turbo-machinery

† Email address for correspondence: bjoern.list@tum.de

(Argyropoulos & Markatos 2015). However, the temporally averaged solutions from RANS simulations lack concrete information about instantaneous vortex movements in the flow. Thus, large eddy simulation (LES) constitutes another common choice for turbulence modelling, providing a time-sensitive perspective to the turbulent flows (Pope 2004). The computational expense of LES is nevertheless still substantial, and their applicability remains restricted (Choi & Moin 2012; Slotnick *et al.* 2014; Yang 2015).

The persistent challenges of traditional approaches motivate the use of machine learning, in particular deep learning, for turbulence modelling (Duraisamy, Iaccarino & Xiao 2019). The reduced complexity of steady-state RANS made these set-ups a promising target for early efforts of machine learning-based turbulence. As a result, substantial progress has been made towards data-driven prediction of RANS flow fields, vastly outperforming pure numerical solvers in the process (Ling, Kurzawski & Templeton 2016; Bhatnagar *et al.* 2019; Thuerey *et al.* 2020).

Contrasting data-driven RANS modelling, further studies were motivated by the additional challenges of predicting transient turbulence. Some of these target performance gains over numerical models by moving the temporal advancement to a reduced-order embedding, where Koopman-based approaches have been an effective choice for constructing these latent spaces (Lusch, Kutz & Brunton 2018; Eivazi *et al.* 2021). In the domain of deep learning-based fluid mechanics, these studies are also among the first to explore the effects of recurrent application of neural networks on training. A related approach by Li *et al.* (2020) moved the learned temporal integrator to Fourier space, with successful applications to a range of problems, including Navier–Stokes flow. An extensive comparison of different turbulence prediction architectures is provided by Stachenfeld *et al.* (2021), and includes applications to multiple flow scenarios.

While turbulence prediction aims to remove the numerical solver at inference time, other concepts in machine learning turbulence try to integrate a learned model in the solver. In the following, we will refer to approaches characterised by this integration of neural networks into numerical solvers as hybrid methods. Some of these efforts target the data-driven development of LES models. An early work showcased the capability of neural networks to reproduce the turbulent viscosity coefficient (Sarghini, De Felice & Santini 2003). Furthermore, Maulik *et al.* (2019) proposed a supervised machine learning method to infer the subgrid scale (SGS) stress tensor from the flow field, and achieved promising results on the two-dimensional decaying turbulence test cases. Herein, the *a priori* evaluations served as a learning target and could be accurately reproduced, however, *a posteriori* evaluations were not always in direct agreement. Beck, Flad & Munz (2019) trained a data-driven closure model based on a convolutional neural network (CNN) and demonstrated good accuracy at predicting the closure on a three-dimensional homogeneous turbulence case, albeit stating that using their trained model in LES is not yet possible. Related prediction capabilities with trade-offs in terms of model stability of a similar supervised approach were reported by Cheng *et al.* (2019). Xie *et al.* (2019) utilised a similar approach on compressible flows, later expanding their method to multi-scale filtering (Xie *et al.* 2020). Park & Choi (2021) studied possible formulations for the input to the neural network and evaluated their results on a turbulent channel flow.

Beyond the supervised learning methods covered so far, Novati, de Laroussilhe & Koumoutsakos (2021) proposed a multi-agent reinforcement learning approach, where the LES viscosity coefficient was inferred by local agents distributed in the numerical domain. Their hybrid solver achieved good results when applied to a forward simulation. These previous studies on machine learning-based turbulence models lead to two fundamental observations. Firstly, sufficiently large networks parameterise a wide range of highly

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

nonlinear functions. Their parameters, i.e. network weights, can be trained to identify and differentiate turbulent structures and draw modelling conclusions from these structures, which yields high accuracy towards *a priori* statistics. Secondly, the feedback from supervised training formulations cannot express the long-term effects of these modelling decisions, and thus cannot provide information about the temporal stability of a model. While reinforcement learning provides long temporal evolutions, its explorative nature makes this method computationally expensive. To exploit the benefits of data-driven training such as supervised models, and simultaneously provide training feedback over long time horizons, a deeper integration of neural network models in numerical solvers is necessary.

Further research achieved this deep integration by training networks through differentiable solvers and adjoint optimisation for partial differential equations. Such works initially focused on learning-based control tasks (de Avila Belbute-Peres *et al.* 2018; Holl, Thuerey & Koltun 2020). By combining differentiable solvers with neural network models, optimisation gradients can propagate through solver steps and network evaluations (Thuerey *et al.* 2021). This allows for targeting of loss formulations that require a temporal evolution of the underlying partial differential equation. These techniques were shown to overcome the stability issues of supervised methods, and thus provided a basis for hybrid methods in unsteady simulations. By integrating CNNs into the numerical solver, Um *et al.* (2020) found models to improve with increased time horizons seen during training, which resulted in a stable learned correction function that was capable of efficiently improving numerical solutions to various partial differential equations. Similarly, Kochkov *et al.* (2021) found differentiable solver architectures to be beneficial for training turbulence models. While this work estimates substantial performance gains over traditional techniques for first-order time integration schemes, we will evaluate a different solver that is second order in time, putting more emphasis on an evaluation with appropriate metrics from fluid mechanics.

In another related approach, Sirignano, MacArt & Freund (2020) proposed a learned correction motivated by turbulence predictions in LES of isotropic turbulence, and later expanded on this by studying similar models in planar jets (MacArt, Sirignano & Freund 2021). Here, *a posteriori* statistics served as a training target, and the authors also compared the performance of models trained on temporally averaged and instantaneous data. However, the study did not investigate the temporal effects of hybrid solvers and their training methodologies in more detail.

In this paper, we seek to develop further understanding of turbulence modelling with hybrid approaches. In an effort to bridge the gap between the previously mentioned papers, we want to address a series of open questions. Firstly, no previous adjoint-based learning approach has been evaluated on a range of turbulent flow scenarios. While this has been done for other, purely predictive learning tasks (Li *et al.* 2020; Stachenfeld *et al.* 2021), we will demonstrate the applicability of adjoint-based training of hybrid methods in multiple different scenarios. Secondly, there is little information on the choice of loss functions for turbulence models in specific flow scenarios. Previous studies have focused on matching ground truth data. Their optimisation procedures did not emphasise specific fluid dynamical features that might be particularly important in the light of long-term model accuracy and stability. Thirdly, previous works on adjoint optimisation have not studied in detail how the number of unrolled steps seen during training affects the neural network models' *a posteriori* behaviour. While previous work on flow prediction reported good results when using multiple prediction steps during training (Lusch *et al.* 2018; Eivazi *et al.* 2021), we want to explore how this approach behaves with learned turbulence models in hybrid solvers. In order to provide insights into these questions, we utilise a CNN to train a corrective forcing term through a differentiable solver, which allows an end-to-end training that is flexible towards the number of unrolled steps, loss formulations and training targets. We then show that the same network architecture can achieve good accuracy with respect to *a posteriori* metrics of three different flow scenarios. In our method, we relax the timestep requirements usually found in unsteady turbulence modelling, such as LES, by downscaling our simulations such that a constant Courant–Friedrichs–Lewy (CFL) ratio is maintained. By implication, a learned model is trained to (i) take the classical sub-grid-scale closure into account, (ii) approximate temporal effects and (iii) correct for discretisation errors. It is worth noting that a network trained for these three targets combines their treatment into one output, with the result that these treatments cannot be separated at a network-output level. Instead, our *a posteriori* evaluations show that neural network models can learn to account for all three of these elements.

The turbulence models are trained and evaluated on three different, two-dimensional flow cases: the isotropic decaying turbulence, a temporally developing mixing layer as well as the spatially developing mixing layer. We show that, in all cases, training a turbulence model through an increasing number of unrolled solver steps enhances the model accuracy and thus demonstrate the benefits of a differentiable solver. Unless stated otherwise, all of the evaluations in the coming sections were performed on out-of-sample data and show the improved generalising capabilities of models trained with the proposed unrollment strategy.

Our unrollment study extends to 60 simulation steps during training. The long solver unrollments involve recurrent network applications, which can lead to training instabilities caused by exploding and diminishing gradients. We introduce a custom gradient stopping technique that splits the gradient calculations into non-overlapping subranges, for which the gradients are evaluated individually. This techniques keeps the long-term information from all unrolled steps, but stops the propagation of gradients through a large number of steps and thus avoids the training instabilities.

Furthermore, our results indicate that accurate models with respect to *a posteriori* turbulence statistics are achieved without directly using them as training targets. Nonetheless, a newly designed loss formulation inspired by *a posteriori* evaluations and flow physics is shown to yield further improvements. Finally, we provide a performance analysis of our models that measures speed ups of up to 14 with respect to comparably accurate solutions from traditional solvers.

The remainder of this paper is organised as follows. In § 2, we give an overview of our methodology and the solver–network interaction. A description and evaluation of experiments with the isotropic decaying turbulence case is found in § 3, which is followed by similar studies regarding the temporally developing mixing layer and the spatially developing mixing layer in §§ 4 and 5, respectively. Section 6 studies the effect our method of splitting back-propagated gradients into subranges. A comparison of computational costs at inference time can be found in § 7, while § 8 contains concluding thoughts.

2. Learning turbulence models

In this paper, we study neural networks for turbulence modelling in incompressible fluids. These flows are governed by the Navier–Stokes equations

$$\left.\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\boldsymbol{\cdot}\nabla\boldsymbol{u} &= -\nabla p + \frac{1}{Re}\nabla^2\boldsymbol{u} + \boldsymbol{f}, \\ \nabla\boldsymbol{\cdot}\boldsymbol{u} &= 0, \end{aligned}\right\} \tag{2.1}$$

where $\boldsymbol{u} = [u\ v]^{\mathrm{T}}$, p and Re are the velocity field, pressure field and Reynolds number respectively. The term $\boldsymbol{f} = [f_x\, f_y]^{\mathrm{T}}$ represents an external force on the fluid. In the context of turbulent flows, an accurate solution to these equations entails either resolving and numerically simulating all turbulent scales, or modelling the turbulence physics through an approximative model.

Our aim is to develop a method that enhances fluid simulations by means of a machine learning model. In particular, we aim to improve the handling of fine temporal and spatial turbulence scales that are potentially under-resolved, such that the influence of these scales on the larger resolved motions needs to be modelled. The function that approximates these effects is solely based on low-resolution data and is herein parameterised by a CNN. The network is then trained to correct a low-resolution numerical solution during the simulation, such that the results coincide with a downsampled high-resolution dataset. Within this hybrid approach, the turbulence model directly interacts with the numerical solver at training and at inference time. To achieve this objective, we utilise differentiable solvers, i.e. solvers which provide derivatives with respect to their output state. Such solvers can be seen as part of the differentiable programming methodology in deep learning, which is equivalent to employing the adjoint method from classical optimisation (Giles *et al.* 2003) in the context of neural networks. The differentiability of the solver enables the propagation of optimisation gradients through multiple solver steps and neural network evaluations.

2.1. *Differentiable PISO solver*

Our differentiable solver is based on the semi-implicit pressure-implicit with splitting of operators (PISO) scheme introduced by Issa (1986), which has been used for a wide range of flow scenarios (Kim & Benson 1992; Barton 1998). Each second-order time integration step is split into an implicit predictor step solving the discretised momentum equation, followed by two corrector steps that ensure the incompressibility of the numerical solution to the velocity field. The Navier–Stokes equations are discretised using the finite-volume method, while all cell fluxes are computed to second-order accuracy.

The solver is implemented on the basis of TensorFlow (TF) (Abadi 2016), which facilitates parallel execution of linear algebra operations on the graphics processing unit (GPU), as well as the differentiability of said operations. Additional functions exceeding the scope of TF are written as custom operations and implemented using compute unified device architecture (CUDA) programming. This approach allows us to seamlessly integrate initially unsupported features such as sparse matrix operations in the TF graph. More details about the solver can be found in Appendix A, where the solver equations are listed in Appendix A.1, implementation details in Appendix A.2 and a verification is conducted in Appendix A.3. Figure 1 gives a brief overview of the solver procedure.

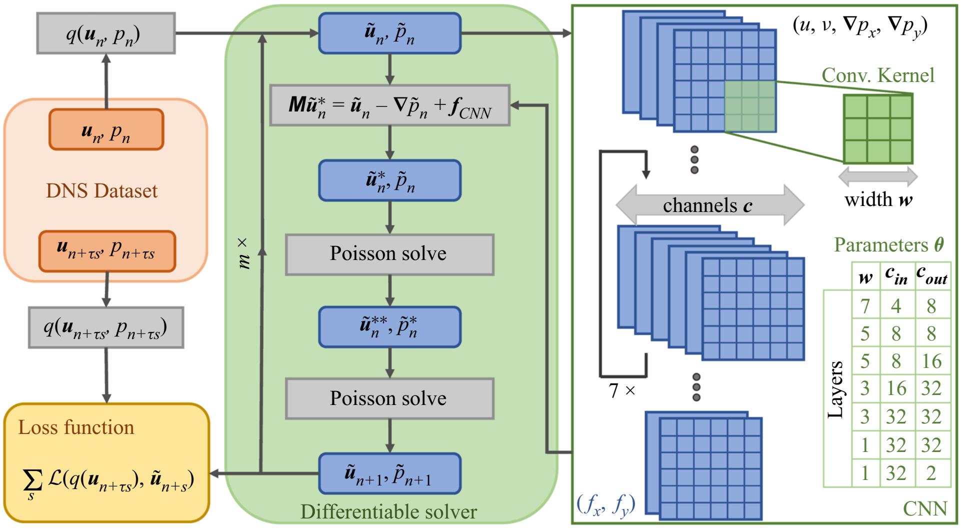
In the following, we will denote a PISO solver step $\mathcal{S}$ as

$$(\boldsymbol{u}_{n+1}, p_{n+1}) = \mathcal{S}(\boldsymbol{u}_n, p_n, \boldsymbol{f}_n), \tag{2.2}$$

where $\boldsymbol{u}_n$, p_n and $\boldsymbol{f}_n$ represent discretised velocity, pressure and forcing fields at time t_n.

2.2. *Neural network architecture*

Turbulence physics strongly depends on the local neighbourhood. Thus, the network has to infer the influence of unresolved scales for each discrete location based on the surrounding flow fields. This physical relation can be represented by discrete convolutions, where each output value is computed based solely on the surrounding computational cells as well


$q(\boldsymbol{u}_n, p_n)$
$\boldsymbol{u}_n, p_n$
DNS Dataset
$\boldsymbol{u}_{n+\tau s}, p_{n+\tau s}$
$q(\boldsymbol{u}_{n+\tau s}, p_{n+\tau s})$
Loss function
$\sum_s \mathcal{L}(q(\boldsymbol{u}_{n+\tau s}), \tilde{\boldsymbol{u}}_{n+s})$
$m\times$
$\tilde{\boldsymbol{u}}_n, \tilde{p}_n$
$\boldsymbol{M}\tilde{\boldsymbol{u}}_n^* = \tilde{\boldsymbol{u}}_n - \nabla\tilde{p}_n + \boldsymbol{f}_{CNN}$
$\tilde{\boldsymbol{u}}_n^*, \tilde{p}_n$
Poisson solve
$\tilde{\boldsymbol{u}}_n^{**}, \tilde{p}_n^*$
Poisson solve
$\tilde{\boldsymbol{u}}_{n+1}, \tilde{p}_{n+1}$
Differentiable solver
$(u, v, \nabla p_x, \nabla p_y)$
Conv. Kernel
width $\boldsymbol{w}$
channels $\boldsymbol{c}$
$7\times$
Parameters $\boldsymbol{\theta}$
Layers
(f_x, f_y)
CNN

w	c_{in}	c_{out}
7	4	8
5	8	8
5	8	16
3	16	32
3	32	32
1	32	32
1	32	2

Figure 1. Solver procedure of the PISO scheme and its interaction with the convolutional neural network; data at time t_n are taken from the DNS dataset and processed by the downsampling operation q that yields downsampled representations of the input fields, before entering the differentiable solver; the solver unrollment performs m steps, each of which is corrected by the CNN, and is equivalent to τ high-resolution steps; the optimisation loss takes all resulting (intermediate) timesteps.

as a convolutional weighting kernel. This formulation introduces a restricted receptive field for the convolution and ensures the local dependence of its output (Luo *et al.* 2016). Chaining multiple of these operations results in a deep CNN, which has been successfully used in many applications ranging from computer vision and image recognition (Albawi, Mohammed & Al-Zawi 2017) to fluid mechanics and turbulence research (Beck *et al.* 2019; Lapeyre *et al.* 2019; Guastoni *et al.* 2021).

We use a fully convolutional network with 7 convolutional layers and leaky rectified linear unit (ReLU) activations, containing $\sim 82 \times 10^3$ trainable parameters. As illustrated in figure 1, our CNN takes the discretised velocity and pressure gradient fields as input. This formulation contains full information of the field variable states, and enables the modelling of both temporal and spatial effects of turbulence, as well as correction of numerical inaccuracies. However, any principles of the modelled physics, such as Galilean invariance in the case of SGS closure, must be learnt by the network itself. The choice of network inputs is by no means trivial, but shall not be further studied in this paper. Refer to Choi & Moin (2012); Xie *et al.* (2019, 2020) and MacArt *et al.* (2021) for in-depth analyses. The output of our networks is conditioned on its weights θ, and can be interpreted as a corrective force $\boldsymbol{f}_{CNN}(\tilde{\boldsymbol{u}}_n, \nabla\tilde{p}_n|\theta) : \mathbb{R}^{\tilde{N}_x\times\tilde{N}_y\times 4} \rightarrow \mathbb{R}^{\tilde{N}_x\times\tilde{N}_y\times 2}$ to the under-resolved simulation of the Navier–Stokes equations (2.1) with domain size $\tilde{N}_x \times \tilde{N}_y$. This force directly enters the computational chain at PISO's implicit predictor step. As a consequence, the continuity equation is still satisfied at the end of a solver step, even if the simulation is manipulated by the network forcing. For a detailed description of the network structure, including CNN kernel sizes, initialisations and padding, refer to Appendix B.

2.3. *Unrolling timesteps for training*

Our method combines the numerical solver introduced in § 2.1 with the modelling capabilities of CNNs as outlined in § 2.2. As also illustrated in figure 1, the resulting data-driven training algorithm works based on a dataset $(\boldsymbol{u}(t_n), p(t_n))$ consisting of high-resolution $(N_x \times N_y)$ velocity fields $\boldsymbol{u}(t_n) \in \mathbb{R}^{N_x \times N_y \times 2}$ and corresponding pressure fields $p(t_n) \in \mathbb{R}^{N_x \times N_y}$ for the discrete time t_n. In order to use these DNS data for training under-resolved simulations on different grid resolutions, we define a downsampling procedure $q(\boldsymbol{u}, p) : \mathbb{R}^{N_x \times N_y \times 3} \to \mathbb{R}^{\tilde{N}_x \times \tilde{N}_y \times 3}$, that takes samples from the dataset and outputs the data $(\tilde{\boldsymbol{u}}_n, \tilde{p}_n)$ at a lower target resolution $(\tilde{N}_x \times \tilde{N}_y)$ via bilinear interpolation. This interpolation provides a simple method of acquiring data at the shifted cell locations of different discretisations. It can be seen as a repeated linear interpolation to take care of two spatial dimensions. The resampling of DNS data is used to generate input and target frames of an optimisation step. For the sake of simplicity, we will denote a downsampled member of the dataset consisting of velocities and pressure as $\tilde{q}_n = q(\boldsymbol{u}(t_n), p(t_n))$. Similarly, we will write $\tilde{\boldsymbol{f}}_n = \boldsymbol{f}_{CNN}(\tilde{\boldsymbol{u}}_n, \nabla \tilde{p}_n | \theta)$. Note that the network operates solely on low-resolution data and introduces a corrective forcing to the low-resolution simulation, with the goal of reproducing the behaviour of a DNS. We formulate the training objective as

$$\min_{\theta}(\mathcal{L}(\tilde{q}_{n+\tau}, \mathcal{S}_{\tau}(\tilde{q}_n, \tilde{\boldsymbol{f}}_n))), \tag{2.3}$$

for a loss function $\mathcal{L}$ that satisfies $\mathcal{L}(x, y) \to 0$ for $x \approx y$. By this formulation, the network takes a downsampled DNS snapshot and should output a forcing which makes the flow fields after a low-resolution solver step closely resemble the next downsampled frame. The temporal increment τ between these subsequent frames is set to match the timesteps in the low-resolution solver $\mathcal{S}$, which in turn are tuned to maintain Courant numbers identical to the DNS.

Um *et al.* (2020) showed that similar training tasks benefit from unrolling multiple temporal integration steps in the optimisation loop. The optimisation can then account for longer-term effects of the network output on the temporal evolution of the solution, increasing accuracy and stability in the process. We utilise the same technique and find it to be critical for the long-term stability of turbulence models. Our notation from equations (2.2) and (2.3) is extended to generalise the formulation towards multiple subsequent snapshots. When training a model through m unrolled steps, the optimisation objective becomes

$$\min_{\theta}\left(\sum_{s=0}^{m} \mathcal{L}(\tilde{q}_{n+s\tau}, \mathcal{S}_{\tau}^{s}(\tilde{q}_n, \tilde{\boldsymbol{f}}_n))\right), \tag{2.4}$$

where $\mathcal{S}^s$ denotes the successive execution of s solver steps including network updates, starting with the initial fields q_i. By this formulation the optimisation works towards matching not only the final, but also all intermediate frames. Refer to Appendix A.1 for a detailed explanation of this approach, including equations for optimisation and loss differentiation.

2.4. *Loss functions*

As introduced in (2.3), the training of deep CNNs is an optimisation of its parameters. The loss function $\mathcal{L}$ serves as the optimisation objective and thus has to assess the quality of the network output. Since our approach targets the reproduction of DNS-like behaviour on a coarse gird, the chosen loss function should consequently aim to minimise the distance between the state of a modelled coarse simulation and the DNS. In this context, a natural

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

choice is the $\mathcal{L}_2$ loss on the sth unrolled solver step

$$\mathcal{L}_2 = \sqrt{(\tilde{\boldsymbol{u}}_s - q(\boldsymbol{u}_{s\tau})) \cdot (\tilde{\boldsymbol{u}}_s - q(\boldsymbol{u}_{s\tau}))}, \tag{2.5}$$

since this formulation drives the optimisation towards resembling a desired outcome. Therefore, the $\mathcal{L}_2$ loss trains the network to directly reproduce the downsampled high-resolution fields, and the perfect reproduction from an ideal model gives $\mathcal{L}_2 = 0$. Since the differentiable solver allows us to unroll multiple simulation frames, we apply this loss formulation across a medium-term time horizon and thus also optimise towards multi-step effects. By repeatedly taking frames from a large DNS dataset in a stochastic sampling process, a range of downsampled instances are fed to the training procedure. While the DNS dataset captures all turbulence statistics, they are typically lost in an individual training iteration. This is due to the fact that training mini-batches do not generally include sufficient samples to represent converged statistics, and no specific method is used to satisfy this criterion. This means that data in one training iteration solely carry instantaneous information. Only the repeated stochastic sampling from the dataset lets the network recover awareness of the underlying turbulence statistics. The repeated matching of instantaneous DNS behaviour thus encodes the turbulence statistics in the training procedure. While the $\mathcal{L}_2$ loss described in (2.5) has its global minimum when the DNS behaviour is perfectly reproduced, in practice, we find that it can neglect the time evolution of certain fine scale, low amplitude features of the solutions. This property of the $\mathcal{L}_2$ loss is not unique to turbulence modelling and has previously been observed in machine learning in other scientific fields such as computer vision (Yu *et al.* 2018). To alleviate these shortcomings, we include additional loss formulations, which alter the loss landscape to avoid these local minima.

We define a spectral energy loss $\mathcal{L}_E$, designed to improve the accuracy of the learned model on fine spatial scales. It is formulated as the log-spectral distance of the spectral kinetic energies at the sth step

$$\mathcal{L}_E = \sqrt{\int_k \log\left(\frac{\tilde{E}_s(k)}{E^q_{s\tau}(k)}\right)^2 \mathrm{d}k}, \tag{2.6}$$

where $\tilde{E}_s(k)$ denotes the spectral kinetic energy of the low-resolution velocity field at wavenumber k, and $E^q_{s\tau}$ represents the same quantity for the downsampled DNS data. In practice, this loss formulation seeks to equalise the kinetic energy in the velocity field for each discrete wavenumber. The log rescaling of the two respective spectra regularises the relative influence of different spatial scales. This energy loss elevates the relative importance of fine scale features.

Our final aim is to train a model that can be applied to a standalone forward simulation. The result of a neural network-modelled low-resolution simulation step should therefore transfer all essential turbulence information, such that the same model can once again be applied in the subsequent step. The premises of modelling the unresolved behaviour are found in the conservation equation for the implicitly filtered low-resolution kinetic energy in tensor notation

$$\frac{\partial \tilde{E}_f}{\partial t} + \tilde{u}_i \frac{\partial \tilde{E}_f}{\partial x_i} + \frac{\partial}{\partial x_j}\tilde{u}_i\left[\delta_{ij}\tilde{p} + \tau^r_{ij} - \frac{2}{Re}\tilde{S}_{ij}\right] = -\epsilon_f - \mathcal{P}^r, \tag{2.7}$$

where $\tilde{E}_f$ denotes the kinetic energy of the filtered velocity field, τ^r_{ij} represents the SGS stress tensor, $\tilde{S}_{ij} = \frac{1}{2}(\partial\tilde{u}_i/\partial x_j + \partial\tilde{u}_j/\partial x_i)$ is the resolved rate of strain and ϵ_f and $\mathcal{P}^r$

are sink and source terms for the filtered kinetic energy. These terms are defined as $\epsilon_f = (2/Re)\tilde{S}_{ij}\tilde{S}_{ij}$ and $\mathcal{P}^r = -\tau^r_{ij}\tilde{S}_{ij}$. The viscous sink ϵ_f represents the dissipation of kinetic energy due to molecular viscous stresses at grid-resolved scales. In hybrid methods, this viscous dissipation at grid level ϵ_f is fully captured by the numerical solver. On the contrary, the source term $\mathcal{P}^r$ representing the energy transfer from resolved scales to residual motions cannot be computed, because the SGS stresses τ^r_{ij} are unknown. One part of the modelling objective is to estimate these unresolved stresses and the interaction of filtered and SGS motions. Since the energy transfer between these scales $\mathcal{P}^r$ depends on the filtered rate of strain $\tilde{S}_{ij}$, special emphasis is required to accurately reproduce the filtered rate of strain tensor. This motivates the following rate of strain loss at the sth unrolled solver step

$$\mathcal{L}_S = \sum_{i,j} |\tilde{S}_{ij,s} - S^q_{ij,s\tau}|, \tag{2.8}$$

where $S^q_{ij,s}$ denotes the rate of strain of the downsampled high-resolution velocity field. This loss term ensures that the output of a hybrid solver step carries the information necessary to infer an accurate forcing in the subsequent step.

While our models primarily focus on influences of small-scale motions on the large-scale resolved quantities, we now draw attention to the largest scale, the mean flow. To account for the mean flow at training time, an additional loss contribution is constructed to match the multi-step statistics and written as

$$\mathcal{L}_{MS} = \|\langle \boldsymbol{u}_s\rangle^m_{s=0} - \langle \tilde{\boldsymbol{u}}_{s\tau}\rangle^m_{s=0}\|_1, \tag{2.9}$$

where $\langle\rangle^m_{s=0}$ denotes an averaging over the m unrolled training steps with iterator s. This notation resembles Reynolds averaging, albeit being focused on the shorter time horizon unrolled during training. Matching the averaged quantities is essential to achieving long-term accuracy of the modelled simulations for statistically steady simulations, but lacks physical meaning in transient cases. Therefore, this loss contribution is solely applied to statistically steady simulations. In this case, the rolling average $\langle\rangle^m_{s=0}$ approaches the steady mean flow for increasing values of m.

Our combined turbulence loss formulation as used in the network optimisations additively combines the aforementioned terms as

$$\mathcal{L}_T = \lambda_2\mathcal{L}_2 + \lambda_E\mathcal{L}_E + \lambda_S\mathcal{L}_S + \lambda_{MS}\mathcal{L}_{MS}, \tag{2.10}$$

where λ denotes the respective loss factor. Their exact values are mentioned in the flow scenario specific chapters. Note that these loss terms, similar to the temporal unrolling, do not influence the architecture or computational performance of the trained neural network at inference time. They only exist at training time to guide the network to an improved state with respect to its trainable parameters. In the following main sections of this paper, we use three different turbulence scenarios to study effects of the number of unrolled steps and the whole turbulence loss $\mathcal{L}_T$. An ablation on the individual components of $\mathcal{L}_T$ is provided in Appendix F. We start with employing the training strategy on isotropic decaying turbulence.

3. Two-dimensional isotropic decaying turbulence

Isotropic decaying turbulence in two dimensions provides an idealised flow scenario (Lilly 1971), and is frequently used for evaluating model performance (San 2014;

Maulik *et al.* 2019; Kochkov *et al.* 2021). It is characterised by a large number of vortices that merge at the large spatial scales whilst the small scales decay in intensity over time. We use this flow configuration to explore and evaluate the relevant parameters, most notably the number of unrolled simulation steps as well as the effects of loss formulations.

In order to generate training data, we ran a simulation on a square domain with periodic boundaries in both spatial directions. The initial velocity and pressure fields were generated using the initialisation procedure by San & Staples (2012). The Reynolds numbers are calculated as $Re = (\hat{e}\hat{l})/\nu$, with the kinetic energy $\hat{e} = (\langle u_i u_i \rangle)^{1/2}$ and the integral length scale $\hat{l} = \hat{u}/\hat{\omega}$ and $\hat{\omega} = (\langle \omega_i \omega_i \rangle)^{1/2}$. The Reynolds number of this initialisation was $Re = 126$. The simulation was run for a duration of $T = 10^4 \Delta t_{DNS} = 100\hat{t}$, where the integral time scale is calculated as $\hat{t} = 1/\hat{\omega}$ at the initial state. During the simulation, the backscatter effect transfers turbulence energy to the larger scales, which increases the Reynolds number (Kraichnan 1967; Chasnov 1997). In our dataset, the final Reynolds number was $Re = 296$. Note that despite this change in Reynolds number, the turbulence kinetic energy is still decreasing and the flow velocities will decay to zero.

Our aim is to estimate the effects of small-scale turbulent features on a coarser grid based on fully resolved simulation data. Consequently, the dataset should consist of a fully resolved DNS and satisfy the resolution requirements. In this case the square domain $(L_x, L_y) = (2\pi, 2\pi)$ was discretised by $(N_x, N_y) = (1024, 1024)$ grid cells and the simulation evolved with a timestep satisfying CFL $= 0.3$.

We trained a series of models on downsampled data, where spatial and temporal resolutions were decreased by a factor of 8, resulting in an effective simulation size reduction of $8^3 = 512$. Our best performing model was trained through 30 unrolled simulation steps. This is equivalent to $1.96\hat{t}$ for the initial simulation state. Due to the decaying nature of this test case, the integral time scale increases over the course of the simulation, while the number of unrolled timesteps is kept constant. As a consequence, the longest unrollments of 30 steps cover a temporal horizon similar to the integral time scale. All our simulation set-ups will study unrollment horizons ranging up to the relevant integral time scale, and best results are achieved when the unrollment approaches the integral time scale. For training the present set-up, the loss factors from equation (2.10) were chosen as $(\lambda_2, \lambda_E, \lambda_S, \lambda_{MS}) = (10, 5 \times 10^{-2}, 1 \times 10^{-5}, 0)$.

To evaluate the influence of the choice of loss function and the number of unrolled steps, several alternative models were evaluated. Additionally, we trained a model with a traditional supervised approach. In this setting, the differentiable solver is not used, and the training is performed purely on the basis of the training dataset. In this case, the corrective forcing is added after a solver step is computed. The optimisation becomes

$$\min_{\theta} (\mathcal{L}(q_{n+\tau}, \boldsymbol{f}_{CNN}(\mathcal{S}_\tau(q_n)))). \tag{3.1}$$

The equations for the supervised training approach are detailed in Appendix A.1. Furthermore, a LES with the standard Smagorinsky model was included in the comparison. A parameter study targeting the Smagorinsky coefficient revealed that a value of $C_s = 0.008$ handles the physical behaviour of our set-up best. See Appendix D for details. An overview of all models and their parameters is given in table 1.

After training, a forward simulation was run for comparison with a test dataset. For the test data, an entirely different, randomly generated initialisation was used, resulting in a velocity field different from the simulation used for training. The test simulations were advanced for $1000\Delta t = 80\hat{t}$.

Note that the temporal advancement of the forward simulations greatly surpasses the unrolled training horizon, which leads to instabilities with the supervised and 1-step

Name	Loss	Steps	$\hat{t}$	MSE at t_1	MSE at t_2
NoModel	—	—	—	2.78×10^{-3}	0.057
LES	—	—	—	2.69×10^{-3}	0.051
$\mathrm{NN}_{sup,T}$	$\mathcal{L}_T$	1	0.07	1.52×10^{-3}	0.369
$\mathrm{NN}_{1,T}$	$\mathcal{L}_T$	1	0.07	1.65×10^{-3}	0.046
NN_{10}	$\mathcal{L}_2$	10	0.66	4.23×10^{-4}	0.018
$\mathrm{NN}_{10,T}$	$\mathcal{L}_T$	10	0.66	4.25×10^{-4}	0.022
$\mathrm{NN}_{30,T}$	$\mathcal{L}_T$	30	1.98	4.09×10^{-4}	0.021

Table 1. Training details for models trained on the isotropic turbulence case, wtih NN representing various versions of our neural network models; mean squared error (MSE) evaluated at $t_1 = 64\Delta t \approx 5\hat{t}$ and $t_2 = 512\Delta t \approx 40\hat{t}$.

model, and ultimately to the divergence of their simulations. Consequently, we conclude that more unrolled steps are critical for the applicability of the learned models and do not include the 1-step model in further evaluations. While an unrollment of multiple steps also improves the accuracy of supervised models, these models nevertheless fall short of their differentiable counterparts, as seen in a deeper study in Appendix E.

We provide visual comparisons of vorticity snapshots in figure 2, where our method's improvements become apparent. The network-modelled simulations produce a highly accurate evolution of vorticity centres, and comparable performance cannot be achieved without a model for the same resolution. We also investigate the resolved turbulence kinetic energy spectra in figure 3. Whilst the no-model simulation overshoots the DNS energy at its smallest resolved scales, the learned model simulations perform better and match the desired target spectrum. Figure 4 shows temporal evolutions of the domain-wide-resolved turbulence energy and the domain-wide-resolved turbulence dissipation rate. The turbulence energy is evaluated according to $E(t) = \int u_i'(t)u_i'(t)\,\mathrm{d}\varOmega$, where u_i' is the turbulent fluctuation. We calculate the turbulence dissipation as $\epsilon(t) = \int \langle \mu(\partial u_i' \partial u_i' / \partial x_j \partial x_j)\rangle\,\mathrm{d}\varOmega$. Simulations with our CNN models strongly agree with the downsampled DNS.

All remaining learned models stay in close proximity to the desired high-resolution evolutions, whereas the LES-modelled and no-model simulations significantly deviate from the target. Overall, the neural network models trained with more unrolled steps outperformed others, while the turbulence loss formulation $\mathcal{L}_T$ also had a positive effect.

In particular, the backscatter effect is crucial for simulations of decaying turbulence (Kraichnan 1967; Smith, Chasnov & Waleffe 1996). The CNN adequately dampens the finest scales, as seen in the high wavenumber section of the energy spectrum (figure 3), it also successfully boosts larger scale motions. In contrast, the no-model simulation lacks dampening at the finest scales and cannot reproduce the backscatter effect on the larger ones. On the other hand, the dissipative nature of the Smagorinsky model used in the LES leads to under-sized spectral energies across all scales. Especially the spectral energies of no-model and LES around wavenumber $k = 10$ show large deviations form the ground truth, while the CNN model accurately reproduces its behaviour. These large turbulent scales are the most relevant to the resolved turbulence energy and dissipation statistics, which is reflected in figure 4. Herein, the neural network models maintain the best approximations, and high numbers of unrolled steps show the best performance at long simulation horizons. The higher total energy of the neural network modelled simulations can be attributed to the work done by the network forcing, which is visualised together

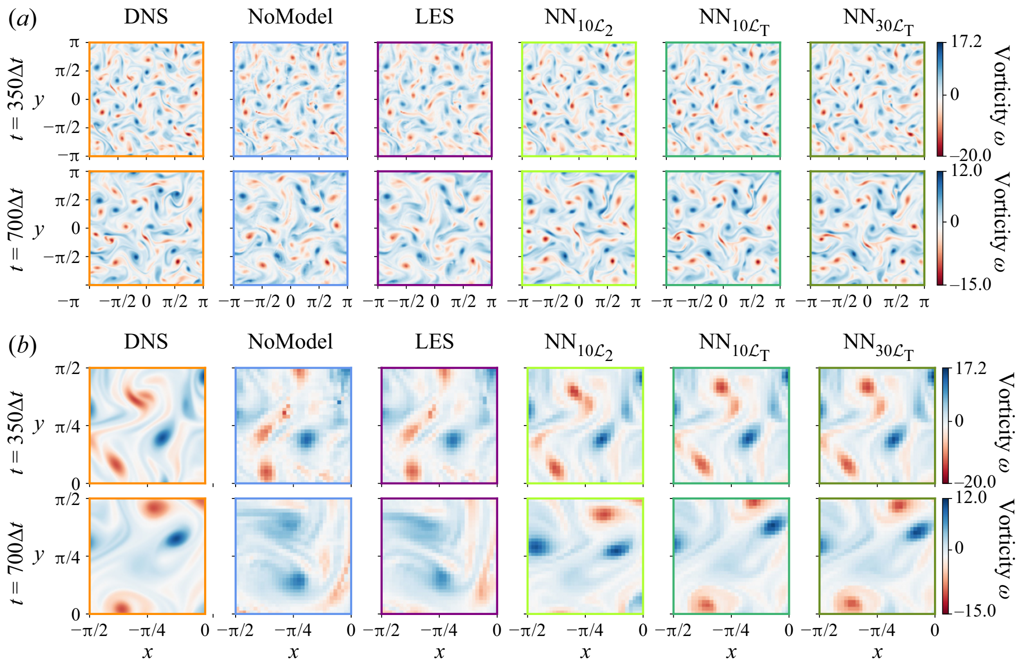


Figure 2. Vorticity visualisations of DNS, no-model, LES, and learned model simulations at $t = (350, 700)\,\Delta t$ on the test dataset, zoomed-in version below.

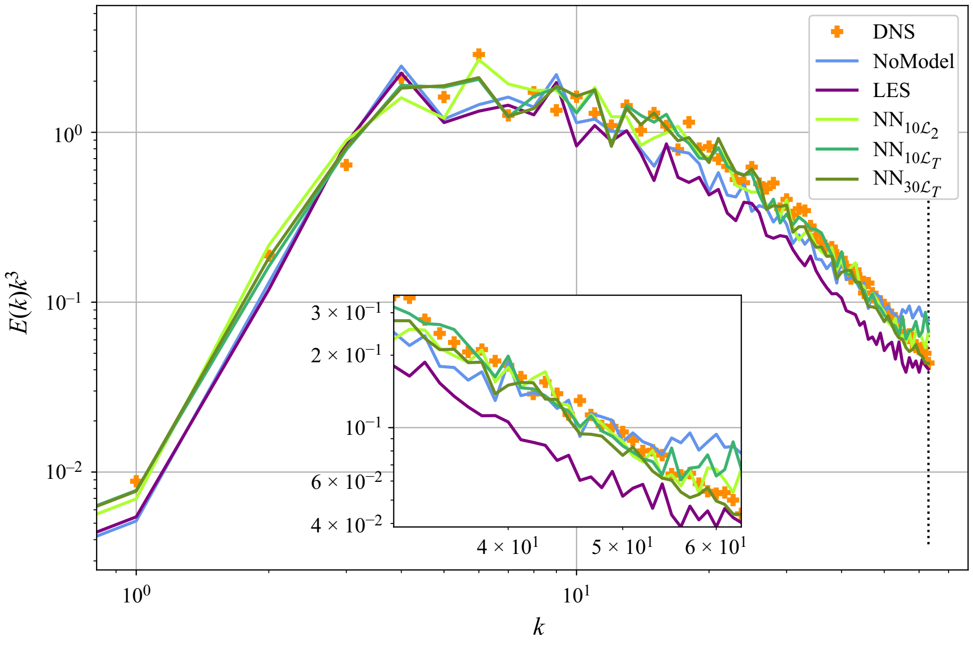


Figure 3. Resolved turbulence kinetic energy spectra of the downsampled DNS, no-model, LES and learned model simulations; the learned 30-step model matches the energy distribution of downsampled DNS data; the vertical line represents the Nyquist wavenumber of the low-resolution grid.

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

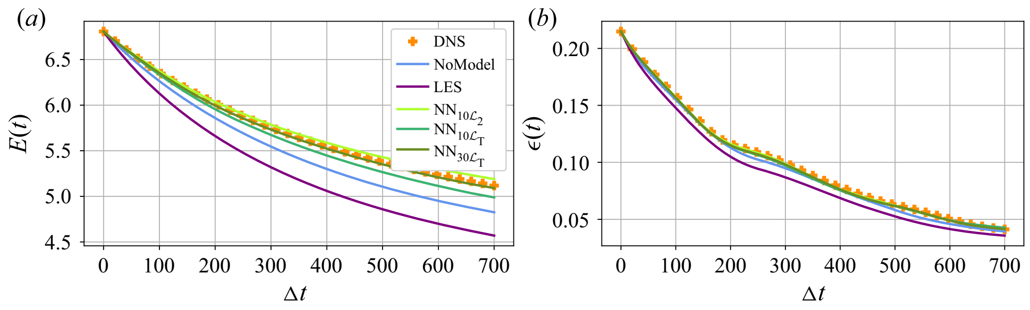


Figure 4. Comparison of DNS, no-model, LES and learned model simulations with respect to resolved turbulence kinetic energy over time (*a*); and turbulence dissipation rate over time (*b*).

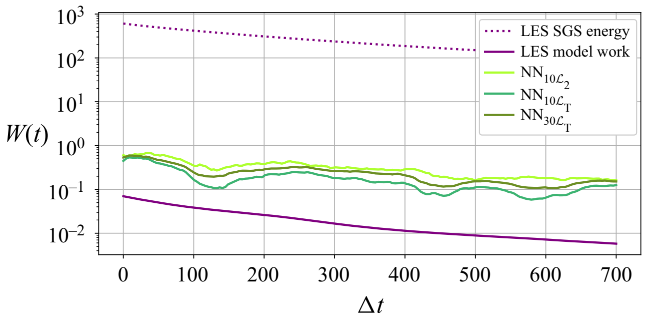


Figure 5. NN-model work on the flow field, work by the LES model and the estimated SGS energies from LES.

with the SGS stress tensor work from the LES simulation as well as its SGS energy in figure 5. This analysis reveals that the neural networks do more work on the system as the LES model does, which explains the higher and more accurate turbulence energy in figure 4 and the spectral energy behaviour at large scales in figure 3.

4. Temporally developing planar mixing layers

Next, we apply our method to the simulation of two-dimensional planar mixing layers. Due to their relevance to applications such as chemical mixing and combustion, mixing layers have been the focus of theoretical and numerical studies in the fluid-mechanics community. These studies have brought forth a large set and good understanding of *a posteriori* evaluations, like the Reynolds-averaged turbulent statistics or the vorticity and momentum thickness. Herein, we use these evaluations to assess the accuracy of our learned models with respect to metrics that are not directly part of the learning targets.

Temporally evolving planar mixing layers are the simplest numerical representation of a process driven by the Kelvin–Helmholtz instability in the shear layer. They are sufficiently defined by the Reynolds number, domain sizes, boundary conditions and an initial condition. Aside from the shear layer represented by a tanh-profile, the initial flow fields feature an oscillatory disturbance that triggers the instability, leading to the roll up of the shear layer. This has been investigated by theoretical studies involving linear stability analysis (Michalke 1964) and numerical simulation (Rogers & Moser 1994). Our set-up is based on the work by Michalke (1964), who studied the stability of the shear layer and proposed initialisations that lead to shear layer roll up. As initial condition, we add

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

	Set-up	a	ω_Ψ
Train	1	6.0	0.7
	2	3.3	1.5
Test	3	9.0	0.3

Table 2. Perturbation details for initial conditions of temporal mixing layer training and test datasets.

randomised modes to the mean profile, resulting in the streamfunction

$$\Psi(x, y) = y + \tfrac{1}{2}\ln(1 + \mathrm{e}^{-4y}) + a((\alpha y)^2 + 1)\,\mathrm{e}^{-(\alpha y)^2}\cos(\omega_\Psi x), \tag{4.1}$$

where a is the amplitude of the perturbation, α parameterises the decay of the perturbation in y-direction and ω_Ψ represents the perturbation frequency. The initial flow field can then be calculated by

$$u(x, y) = \frac{\partial \Psi}{\partial y}, \quad v(x, y) = -\frac{\partial \Psi}{\partial x}. \tag{4.2a,b}$$

At the initial state this results in a velocity step $\Delta U = U_2 - U_1 = 1$ and a vorticity thickness of $\delta_\omega = \Delta U/(\partial U/\partial y|_{max}) = 1$, where velocities marked as U represent mean-stream quantities. Thus, U_2 and U_1 are the fast and slow mean velocities of the shear layer. The computational domain of size $(L_x, L_y) = (40\pi, 20\pi)$ is discretised by $(N_x, N_y) = (1024, 512)$ grid cells for the high-resolution dataset generation. The streamwise boundaries are periodic, while the spanwise boundaries in the y-direction are set to a free-slip boundary, where $\partial u/\partial y|_{\Omega_y} = 0$, $v|_{\Omega_y} = 0$ and $p|_{\Omega_y} = 0$. The Reynolds number based on the unperturbed mean profile and the vorticity thickness is calculated to be $Re = \Delta U \delta_\omega/\nu = 250$ for all randomised initialisations. The simulations are run for $T = 420 = 12\ 000\Delta t_{DNS}$. Our dataset consists of three simulations based on different initialisations. Their perturbation details are found in table 2. Two of these simulations were used as training datasets, while all of our evaluation is performed on the remaining one as the extrapolation test dataset.

Following the approach in § 3, the model training uses a $8\times$ downscaling in space and time. The loss composition was set to $(\lambda_2,\ \lambda_E,\ \lambda_S,\ \lambda_{MS}) = (100,\ 2,\ 5 \times 10^{-2},\ 0)$. We used the same CNN architecture as introduced earlier, although due to the difference in boundary conditions a different padding procedure was chosen (see Appendix B). To illustrate the impact of the turbulence loss $\mathcal{L}_T$ and an unrolling of 60 numerical steps, we compare with several variants with reduced loss formulations and fewer unrolling steps. The maximum number of 60 unrolled steps corresponds to $16t_{\delta_\theta}$ integral time scales computed on the momentum thickness as $t_{\delta_\theta} = \delta_\theta/\Delta U$. With the shear layer growing, the momentum thickness increases 7-fold, which decreases the number of integral time scales to 2 for 60 steps of unrollment. Table 3 shows details of the model parameterisations. To avoid instabilities in the gradient calculation that could ultimately lead to unstable training, we split the back-propagation into subranges for the 60-step model. This method stabilises an otherwise unstable training of the 60-step model, and a split into 30-step long back-propagation subranges performs best. Such a model is added to the present evaluations as $\mathrm{NN}_{60,\mathcal{L}_T}$. Detailed results regarding the back-propagation subranges are discussed in § 6.

The trained models were compared with a downsampled DNS and a no-model simulation, all sharing the same starting frame from the test dataset. This test dataset

Name	Loss	Steps	t_{δ_θ}	MSE at t_e
NoModel	—	—	—	1.25×10^{-3}
NN_{10}	$\mathcal{L}_2$	10	2.7	3.19×10^{-4}
$\mathrm{NN}_{10,\mathcal{L}_T}$	$\mathcal{L}_T$	10	2.7	3.31×10^{-5}
$\mathrm{NN}_{30,\mathcal{L}_T}$	$\mathcal{L}_T$	30	8.1	2.26×10^{-5}
$\mathrm{NN}_{60,\mathcal{L}_T}$	$\mathcal{L}_T$	60	16.1	1.93×10^{-5}

Table 3. Model details for unrollment study; MSE with respect to DNS from test data at $t_e = 512\Delta t$.

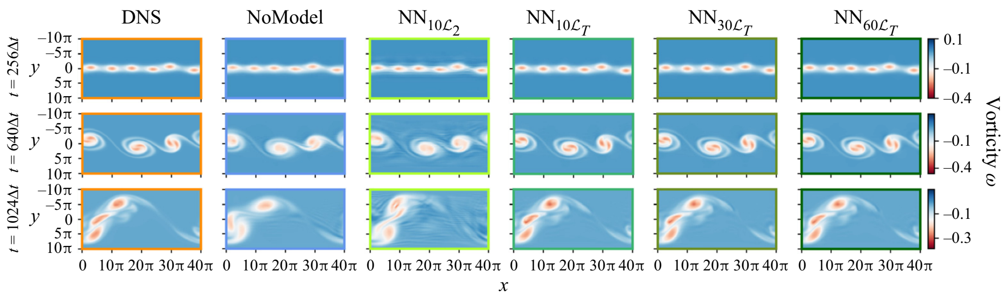


Figure 6. Vorticity visualisations of DNS, no-model and learned model simulations at $t = (256, 640, 1024)\Delta t$ on the test dataset.

changes the initial condition, where different perturbation frequencies and amplitudes result in a variation in vortex roll-up and vortex merging behaviour of the mixing layer. The resulting numerical solutions were compared at three different evolution times $t = [256\ 640\ 1024]\Delta t$. Figure 6 shows the vorticity heat map of the solutions. Qualitatively, the simulations corrected by the CNN exhibit close visual proximity to the DNS by boosting peaks in vorticity where applicable, and additionally achieve a dampening of spurious oscillations.

These observations are matched by corresponding statistical evaluations. The statistics are obtained by averaging the simulation snapshots along their streamwise axis and the resulting turbulence fluctuations were processed for each evaluation time. Figure 7 shows that all $\mathcal{L}_T$-models closely approximate the DNS reference with respect to their distribution of resolved turbulence kinetic energy and Reynolds stresses along the cross-section, while the no-model simulation clearly deviates. Note that the mixing process causes a transfer of momentum from fast to slow moving sections through the effects of turbulent fluctuations. The shear layer growth is thus dominated by turbulent diffusion. Consequently, accurate estimates of the turbulent fluctuations are necessary for the correct evolution of the mixing layer. These fluctuations are most visible in the Reynolds stresses $u'v'$, and an accurate estimation is an indicator for well-modelled turbulent momentum diffusion. The evaluations also reveal that unrolling more timesteps during training gains additional performance improvements. These effects are most visible when comparing the 10-step and 60-step model in a long temporal evolution, as seen in the Reynolds stresses in figure 7. The evaluation of resolved turbulence kinetic energies shows that the models correct for the numerical dissipation of turbulent fluctuations, while, in contrast, there is an underestimation of kinetic energy in the no-model simulation. While longer unrollments generally yield better accuracy, it is also clear that 30 steps come close to saturating the model performance in this particular flow scenario. With the integral time scales

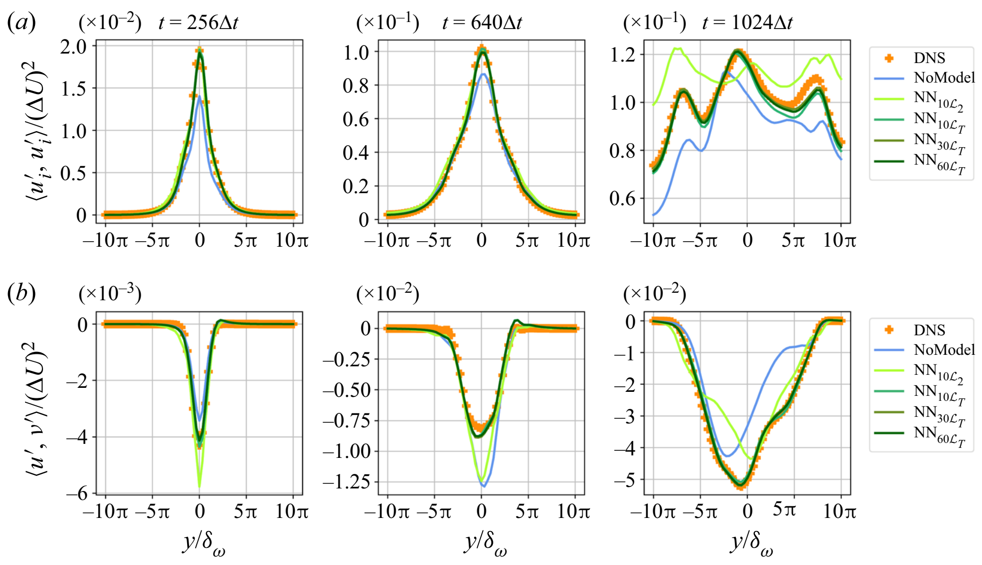


Figure 7. Comparison of DNS, no-model and learned model simulations with respect to resolved turbulence kinetic energy (*a*) and Reynolds stresses (*b*).

mentioned earlier, it becomes clear that 30 simulation steps capture one integral time scale of the final simulation phase, i.e. the phase of the decaying simulation that exhibits the longest time scales. One can conclude that an unrollment of one time scale is largely sufficient, and further improvements of unrolling 2 time scales with 60 steps are only minor.

The resolved turbulence kinetic energy spectra are evaluated to assess the spatial scales at which the corrective models are most active. The spectral analysis at the centreline is visualised in figure 8, whilst the kinetic energy obtained from fluctuations across the cross-section with respect to streamwise averages is shown in figure 9. These plots allow two main observations: firstly, the deviation of kinetic energy mostly originates from medium-sized spatial scales, which are dissipated by the no-model simulation but are accurately reconstructed by the neural network trained with $\mathcal{L}_T$. This effect is connected to the dampening of vorticity peaks in the snapshots in figure 6. Secondly, the fine-scale spectral energy of the no-model simulation has an amplitude similar to the DNS over long temporal horizons (figure 9). This can be attributed to numerical oscillations rather than physical behaviour. These numerical oscillations, as also seen in the snapshots in figure 6, exist for the no-model simulation but are missing in the $\mathcal{L}_T$-modelled simulations. Training a model without the additional loss terms in $\mathcal{L}_T$ from (2.10), i.e. only with the $\mathcal{L}_2$ from (2.5), yields a model that is inaccurate and results in unphysical oscillations. It does not reproduce the vorticity centres, and is also unstable over long temporal horizons. Herein, non-physical oscillations are introduced, which also show up in the cross-sectional spectral energies and vorticity visualisations. We thus conclude that best performance can be achieved with a network trained with $\mathcal{L}_T$, which learns to dampen numerical oscillations and reproduces physical fluctuations across all spatial scales.

It is worth noting that our method is capable of enhancing an under-resolved simulation across a wide range of turbulent motions. The vortex size in the validation simulation ranges from $7\delta_{\omega 0}$ at the starting frame to $60\delta_{\omega 0}$ after evolving for $1200\Delta t$. This timespan encompasses two vortex merging events, both of which cannot be accurately reproduced

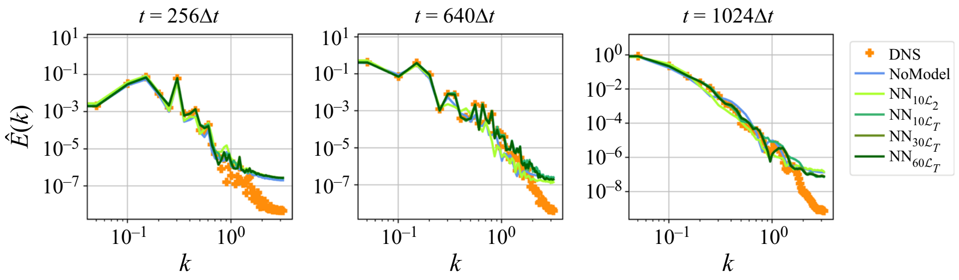


Figure 8. Centreline kinetic energy spectra for the downsampled DNS, no-model and learned model simulations.

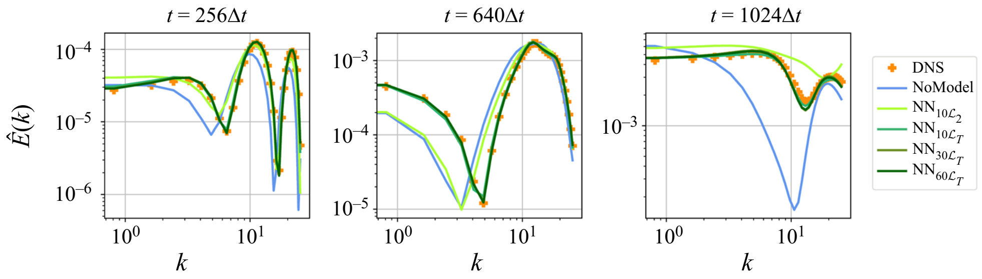


Figure 9. Cross-sectional kinetic energy spectra of the downsampled DNS, no-model and learned model simulations.

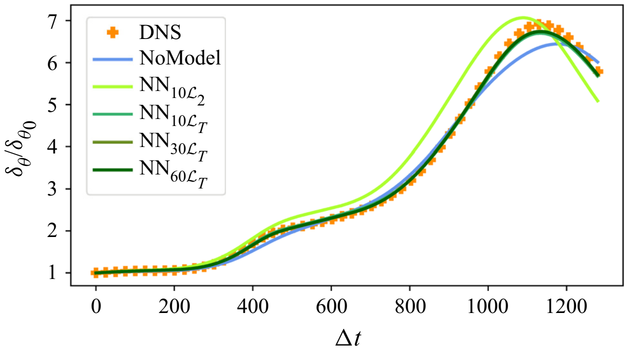


Figure 10. Momentum thickness of DNS, no-model and learned model simulations, evaluated based on the streamwise averages.

with a no-model, or a $\mathcal{L}_2$-model simulation, but are captured by the $\mathcal{L}_T$-trained network models. This is shown in the comparison of the momentum thicknesses over time in figure 10. The reproduction of turbulence statistics (figure 7) yields, in the long term, an accurate turbulent diffusion of momentum and mixing layer growth for the models trained with $\mathcal{L}_T$. On the contrary, the $\mathcal{L}_2$ model fails to reproduce the vortex cores and deviates with respect to the momentum thickness for long temporal horizons.

5. Spatially developing planar mixing layers

In contrast to the temporally developing mixing layers investigated in § 4, the spatially developing counterpart features a fixed view on a statistically steady flow field, which

	$\epsilon_1/\bar{U}$	K_1	Ω_1	$\epsilon_2/\bar{U}$	K_2	Ω_2
Train	**0.075**	0.4π	0.22	**0.025**	0.3π	0.11
	0.060	0.4π	0.22	**0.040**	0.3π	0.11
	0.050	0.4π	0.22	**0.050**	0.3π	0.11
	0.040	0.4π	0.22	**0.060**	0.3π	0.11
	0.025	0.4π	0.22	**0.075**	0.3π	0.11
Test	**0.082**	0.4π	0.22	**0.018**	0.3π	0.11

Table 4. Perturbation details for the inlet condition of training and test datasets.

introduces a new set of challenges to the learning task. While the main difficulty in previous transient simulations was the modelling of an evolving range of turbulent scales, the statistically steady nature of the spatially developing mixing layer requires a reproduction of the turbulent statistics in its own statistically steady state. This in turn necessitates long-term accuracy and stability.

Spatially mixing layers develop from an instability in the shear layer. This instability is driven by a disturbance at the inlet, whose characteristics have great effect on the mixing layer growth (Ho & Huang 1982). In a simulation environment, these disturbances are realised by a Dirichlet inlet boundary condition, where temporally varying perturbations are added to a steady mean flow profile. As proposed by Ko, Lucor & Sagaut (2008), a suitable inlet condition including perturbations can be written as

$$u_{in}(y,t) = 1 + \frac{\Delta U}{2}\tanh(2y) + \sum_{d=1}^{N_d} \epsilon_d (1 - \tanh^2(y/2)) \cos(K_d y) \sin(\Omega_d t), \tag{5.1}$$

where the number of perturbation modes $N_d = 2$ holds for our simulations. Furthermore, we used inviscid wall conditions for the two y-normal spanwise boundaries, and the outflow boundary was realised by a simple Neumann condition with a stabilising upstream sponge layer. For all simulations, we set the characteristic velocity ratio $\Delta U = 1$ and the vorticity thickness to $\delta_\omega = 1$. The vorticity-thickness Reynolds number is set to $Re_{\delta_\omega} = \Delta U \delta_\omega/\nu = 500$. To generate the DNS dataset, this set-up was discretised by a uniform grid with $(N_x, N_y) = (2048, 512)$ resolving the domain of size $(L_x, L_y) = (256, 64)$. The timesteps were chosen such that CFL$= 0.3$ and the temporal evolution was run for 7 periods of the slowest perturbation mode $i = 2$ to reach a statistically steady state, before subsequent frames were entered into the dataset. A further 28 periods of the slowest perturbation mode were simulated to generate 32 000 samples of the statistically steady state. The training dataset consists of 5 such simulations with different perturbations, as summarised in table 4. A downsampling ratio of $8\times$ in space and time was again chosen for the learning set-up. The input to the network was set to include only the main simulation frame without the sponge layer region.

Our best performing model applied the turbulence loss $\mathcal{L}_T$, with the loss factors set to $(\lambda_2, \lambda_E, \lambda_S, \lambda_{MS}) = (50, 0.5, 2, 0.5)$, and an unrollment of 60 solver steps. The timespan covered by these 60 solver steps is comparable to a full period of the slowest perturbation mode. Using the roll-up frequency of the spatial mixing layer as the basis for the time scale $t_{f_\omega} = 1/f_\omega$, 60 solver steps unroll $0.85 t_{f_\omega}$. As we detail in the following, our test metrics show that this approach of unrolling roughly one integral time scale yields the best results.

First, we evaluate the influence of unrollment in this test case. Once again, we show comparisons with additional set-ups, the parametric details of which can be found in table 5. Similar to the temporal mixing layer, the 60 step model was trained using a gradient stopping technique. A 30-step back-propagation subrange performed best again by maintaining long-term information while avoiding instabilities in the gradient calculation. This model is described as $\mathrm{NN}_{60,\mathcal{L}_T}$ in this section. Details regarding the method are explained in § 6. The table shows that the simulation with the 60-step neural network outperforms the no-model baseline by an order of magnitude. For these evaluations, we assessed the model capabilities by running a CNN-corrected forward simulation. This simulation was initialised with a downsampled frame from the DNS test dataset in its fully developed state. This test dataset is generated with different inflow conditions, where the inlet forcing lies outside of the training range, making these evaluations an out-of-sample generalisation test. The variation in inlet forcing affects the location and intensity of the mixing layer roll-up and vortex merging. The simulation was run for $5000\Delta t$, or 36 periods of the slowest perturbation mode in order to obtain data from a statistically stable state. Despite this time frame being orders of magnitude longer than what is seen by the models at training time, the 60-step model retains a stable simulation that closely matches the behaviour of the DNS reference. Interestingly, this longer unrollment, of the order of one integral time scale, is crucial to arrive at a stable model. The models trained with shorter unrollment exhibit various degrees of spurious oscillations, especially the 10-step model. These oscillations most likely originate from slight deviations in turbulent structures (e.g. vortex roll-up) inferred by the network. Since short unrollment models have never seen any further development of these self-exited structures, applying said models eventually causes even stronger unphysical oscillations downstream. As before, we omit purely data-driven models trained with pre-computed simulation states. These produce undesirable solutions within a few timesteps of simulating the test cases. The vorticity visualisations after half a period of the slowest perturbation mode ($70\Delta t$) and after four periods or one flow through time ($600\Delta t$) are shown in figures 11(*a*) and 11(*b*), and compared with DNS and the no-model simulation. The early evaluation in figure 11(*a*) reveals a severe loss of detail in the no-model simulation, even after a short time horizon. Over this timespan, the learned model achieves a close visual reproduction. Additionally, the later vorticity heat map in figure 11(*b*) shows a delayed roll-up in the no-model simulation, whereas the learned model maintains the roll-up location and shows improved accuracy. This behaviour is clarified by the Reynolds-averaged properties of the simulations, for which resolved Reynolds stresses and turbulence kinetic energies were calculated on the basis of the respective statistically steady simulations. As shown in figure 12, the no-model statistics severely deviate from the targeted DNS. In contrast, the corrective forcing inferred by the trained models approximates these statistics more accurately. The delayed roll-up of the no-model simulation and the improvement of the modelled ones is connected to the Reynolds stresses. The Reynolds stresses indicate turbulent diffusion of momentum, and figure 12 shows that the CNN learned to encourage turbulent fluctuations at the start of the mixing layer. The fluctuations trigger the shear layer instability and feed the roll-up, with decisive implications for the downstream development of the mixing layer. Especially the long unrollment of 60 steps benefits the model performance. Evaluations at locations downstream of the initial roll-up see the accuracy of 10 and 30 step models deteriorate in direct comparison with the 60-step model.

These observations regarding the Reynolds stresses extend to the resolved turbulence kinetic energies (figure 12), where the same turbulent fluctuations yield an accurate reproduction of the DNS. The learned models are not limited to a specific spatial scale,

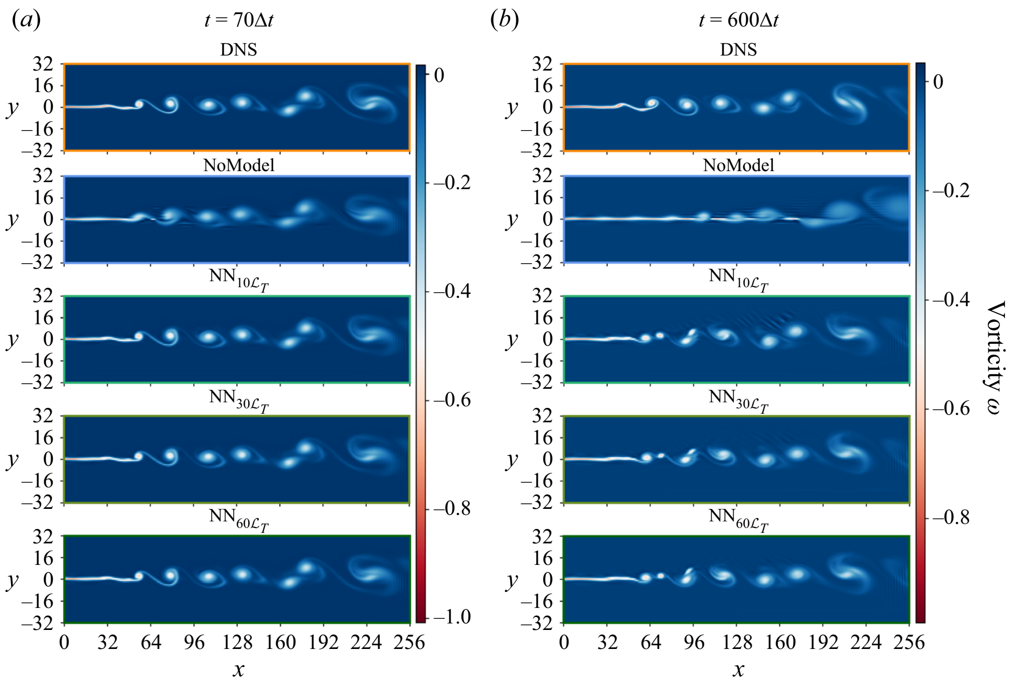


Figure 11. Vorticity heat maps of the spatial mixing layer simulations at (*a*) $t = 70\Delta t$, and (*b*) $t = 600\Delta t$, on the test dataset.

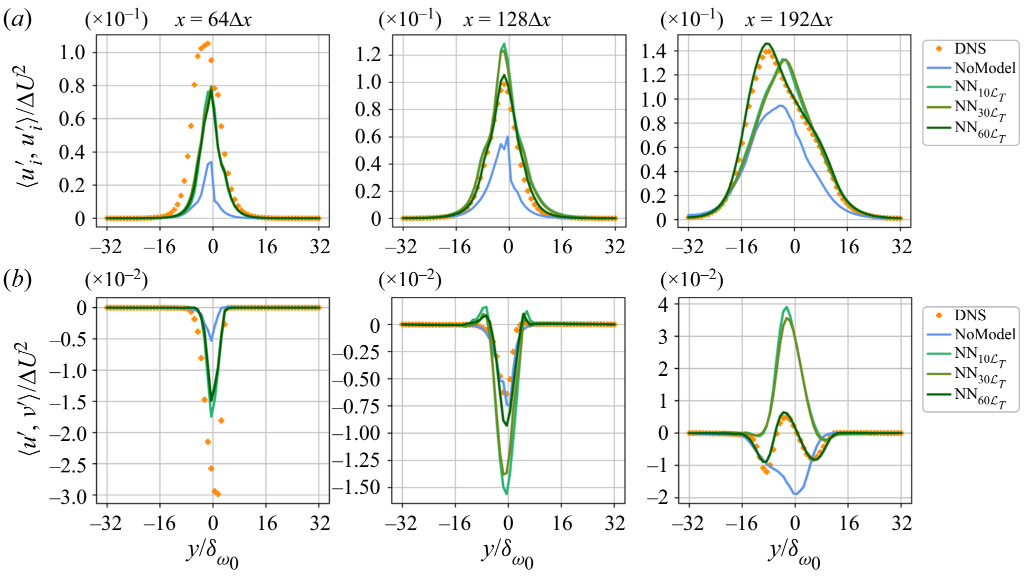


Figure 12. Comparison of downsampled DNS, no-model and learned model simulations with respect to Reynolds-averaged resolved turbulence kinetic energy (*a*) and Reynolds stresses (*b*).

but precisely match the DNS on all turbulent scales when comparing the centreline kinetic energy spectra in figure 13.

The evaluations of vorticity and momentum thickness in figures 14(*a*) and 14(*b*) capture a delayed mixing layer development. Especially early stages of the mixing layer

Name	Loss	Steps	t_{f_ω}	MSE at t_e
NoModel	—	—	—	2.03×10^{-2}
$\mathrm{NN}_{10,\mathcal{L}_T}$	$\mathcal{L}_T$	10	0.14	5.22×10^{-3}
$\mathrm{NN}_{30,\mathcal{L}_T}$	$\mathcal{L}_T$	30	0.42	3.66×10^{-3}
$\mathrm{NN}_{60,\mathcal{L}_T}$	$\mathcal{L}_T$	60	0.85	2.98×10^{-3}

Table 5. Model details for unrollment study; MSE with respect to DNS from test data at $t_e = 1000\Delta t$.

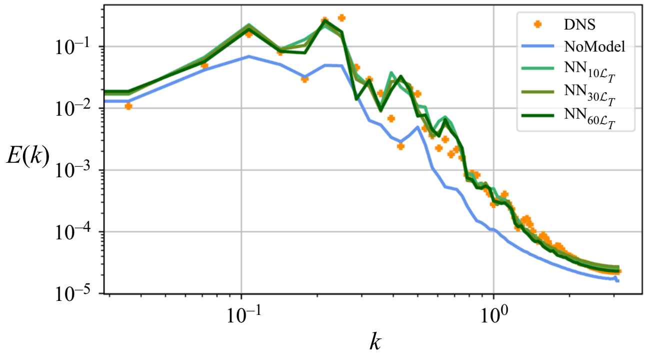


Figure 13. Centreline kinetic energy spectra for downsampled DNS, no-model and learned model simulations.

immediately after the first roll-up are modelled inaccurately. While all models show this behaviour, the delay in terms of momentum thickness is more pronounced for the long unrollment 60-step model. On the contrary, the roll-up inaccuracy results in a noticeable offset in the vorticity thickness around $x/\delta_{\omega_0} = 100$ for all models, but the 60-step model performs best further downstream by recovering the DNS behaviour. This recovery is lacking in the 10- and 30-step models, causing the evaluation of Reynolds stresses at $x = 192\Delta x$ (figure 12) to exhibit large discrepancies between DNS and learned model simulation for these models, with the notable exception of the 60-step model. Note, however, that, despite not being capable of exactly reproducing the entire mixing layer up to the finest detail, the learned models still greatly outperform a no-model simulation. Momentum thickness evaluations show beneficial results for the models trained with shorter unrollments. Due to the definition of momentum thickness as an integral quantity over the shear direction, an increase in this quantity is caused by strong deviations from the initial step profile of the mixing layer. While the integral values for the momentum thickness of the 10- and 30-step models are close to the DNS, the underlying turbulence fluctuations causing these values are not accurate compared with the DNS, which can be seen in turbulence kinetic energy and Reynolds stress evaluations in figure 12. Considering these results jointly, we draw the conclusion that the 60-step model yields the best performance.

Additionally, the evaluations show the benefits of training through multiple unrolled steps. The 10-step model develops instabilities after $500\Delta t$, which is equivalent to one flow-through time. From this time on, the learned model only sees self-exited instabilities in the mixing layer. This constitutes an extrapolation with respect to the temporal unrollment, as well as with respect to the inlet perturbation due to the use of a test dataset. This in turn can cause spurious oscillations and thus a deterioration of solution quality.

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

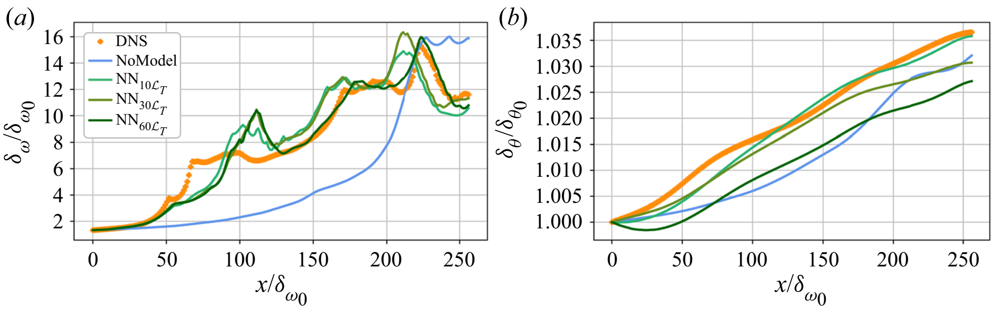


Figure 14. Vorticity and momentum thickness of the downsampled DNS, no-model and learned model simulations.

The 30-step model shows this behaviour to a lesser extent and generates a stable, statistically steady progression of the mixing layer for this case of temporal extrapolation. Even better behaviour is only achieved by the 60-step model. It practically eliminates the instabilities seen in other models.

While previous evaluations showcased the stability improvements gained by training through multiple solver steps, another benefit of this approach relates to the temporal fluctuations in DNS training data. As visualised in figure 15, only some of the interactions between CNN and these temporal oscillations are covered in a training iteration. Consequently, the training loop imposes a high-pass cutoff on the observed frequencies that directly depends on the number of unrolled solver steps. To extract the temporal features that our models learned from the training dataset, we calculate the power spectral density of the velocity fields at sampling point $(x, y) = (160, 0)$ on training data. The sampling timespan for the learned models starts after one flow-through time and stops after the next four flow-through times passed. The resulting power spectral densities are compared with a long-term evaluation of the DNS data, and a relative error between the spectra is computed. The results are shown in figure 15 and support the following observations. Firstly, all learned models can capture the discrete nature of the dominant frequencies quite well. In particular, the 60-step model shows good approximation to the DNS evaluation. In contrast, the no-model does not match the DNS characteristics. Secondly, the relative error of the power spectra generated by the 60-step model is substantially lower for all but the highest frequencies. Since the 30- and 10-step models only saw the interaction with fine scales during their training, these models perform worse at the lower frequencies, which results in higher relative errors for the relatively low vortex roll-up and vortex merging frequencies. These features operate at of the order of one integral time scale and are better resolved by 60 unrolled steps.

6. Gradient back-propagation

Our evaluations on temporally and spatially developing mixing layers show significant performance gain by longer unrollment times, with the best accuracy given by a 60-step model. However, long unrollments can cause stability problems. Repeated applications of neural networks are known to be problematic during training, where exploding or diminishing gradients can significantly deteriorate the quality of gradients (Pascanu, Mikolov & Bengio 2013). To avoid this, we utilise a custom version of the gradient stopping technique: instead of discarding gradients generated by some (earlier) simulation steps, we split the gradient back-propagation into individually evaluated subranges.

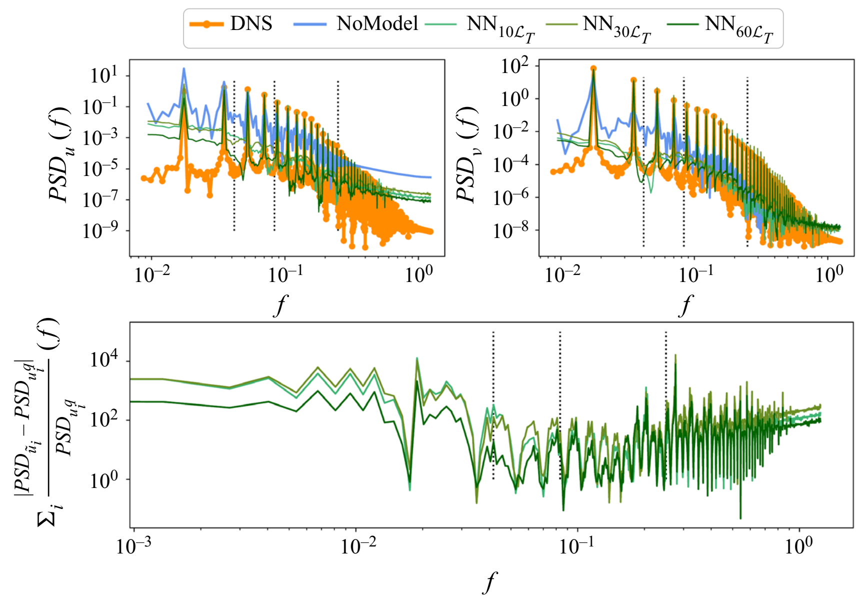


Figure 15. Power spectral density (PSD) of velocity fluctuations over time at sampling point $(x, y) = (192\Delta x, 0)$ based on the training dataset for DNS, no-model and learned model simulations at top; bottom figure displays the relative error of the power densities over frequencies, accumulated for both velocity components; frequencies to the right of a dotted vertical line are fully enclosed in a training iteration; vertical lines correspond to (60, 30, 10) unrolled steps from left to right.

In other words, the training still exposes long temporal unrollments and preserves the gradient influence of all steps, but does not propagate gradients back to the first application of the network model. We use 60-step models to study model accuracy with respect to the length of these back-propagation subranges on a range of 10, 20, 30 and 60 backward steps. We will use the notation $NN_{m\text{–}g}$ with two numbers m and g, where m describes the number of unrolled forward steps, and g represents the length of the subranges for which gradients are calculated individually. In practice, this means that gradients of a 60-20 model are only back-propagated through 3 non-overlapping sections of 20 steps each. The 60-60 model recovers the standard differentiable training procedure used for previous models.

This procedure was applied to temporally and spatially developing mixing layers. Details of the trained models are found in tables 6 and 7. Note that the training of the $NN_{60\text{–}60,\mathcal{L}_T}$ was not stable for the temporal mixing layer case, which we attribute to unstable gradients in the optimisation. In contrast, the subrange gradient models are stable during training. Additional evaluations of Reynolds stresses and turbulence kinetic energy for the temporal mixing layer indicate no performance differences between these models, as shown in figure 16. We thus conclude that the method of subrange back-propagation makes the training of the 60-step model possible, but also that the model performance on the temporal mixing layer was already saturated by the 30-step model, as previously mentioned in § 4. The $NN_{60\text{–}30,\mathcal{L}_T}$ model was used in the evaluation in § 4.

The spatial mixing layer models are evaluated on vorticity snapshots in figure 17, turbulence kinetic energy and Reynolds stresses in figure 18, as well as vorticity and momentum thickness in figures 19. These results indicate that there is a optimal number of consecutive back-propagation steps of around 20 to 30, where the optimisation gradients contain long-term information while still maintaining a good quality that is unaffected

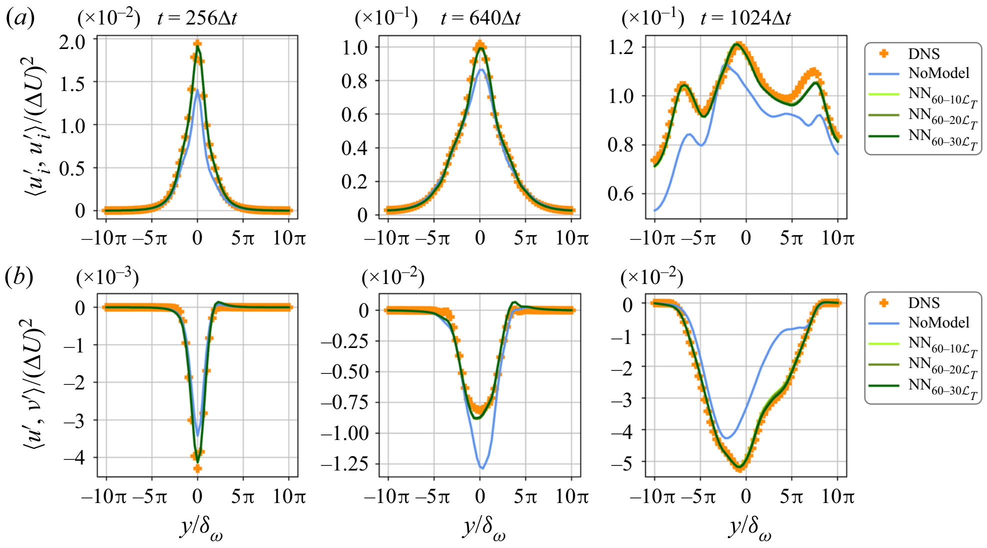


Figure 16. Comparison of DNS, no-model and 60-step model simulations with respect to resolved turbulence kinetic energy (*a*), and Reynolds stresses (*b*).

Name	Loss	Steps	Grad	MSE at t_e
NoModel	—	—	—	1.25×10^{-3}
$\mathrm{NN}_{60\text{–}10,\mathcal{L}_T}$	$\mathcal{L}_T$	60	10	2.36×10^{-5}
$\mathrm{NN}_{60\text{–}20,\mathcal{L}_T}$	$\mathcal{L}_T$	60	20	2.19×10^{-5}
$\mathrm{NN}_{60\text{–}30,\mathcal{L}_T}$	$\mathcal{L}_T$	60	30	1.93×10^{-5}
$\mathrm{NN}_{60\text{–}60,\mathcal{L}_T}$	$\mathcal{L}_T$	60	60	—

Table 6. Temporal mixing layer; model details for 60-*X* models; MSE with respect to DNS from test data at $t_e = 512\Delta t$; training of $\mathrm{NN}_{60\text{–}60,\mathcal{L}_T}$ is unstable.

Name	Loss	Steps	Grad	MSE at t_e
NoModel	—	—	—	2.03×10^{-2}
$\mathrm{NN}_{60\text{–}10,\mathcal{L}_T}$	$\mathcal{L}_T$	60	10	2.44×10^{-3}
$\mathrm{NN}_{60\text{–}20,\mathcal{L}_T}$	$\mathcal{L}_T$	60	20	2.73×10^{-3}
$\mathrm{NN}_{60\text{–}30,\mathcal{L}_T}$	$\mathcal{L}_T$	60	30	2.98×10^{-3}
$\mathrm{NN}_{60\text{–}60,\mathcal{L}_T}$	$\mathcal{L}_T$	60	60	1.19×10^{-2}

Table 7. Spatial mixing layer; model details for 60-*X* models; MSE with respect to DNS from test data at $t_e = 1000\Delta t$.

by risks of recurrent evaluation. The $\mathrm{NN}_{60\text{–}20,\mathcal{L}_T}$ and $\mathrm{NN}_{60\text{–}30,\mathcal{L}_T}$ models achieve best performance on all metrics except for the momentum thickness. We attribute the larger values of momentum thickness to some spurious oscillations exhibited by the $\mathrm{NN}_{60\text{–}10,\mathcal{L}_T}$ and $\mathrm{NN}_{60\text{–}60,\mathcal{L}_T}$ models. The $\mathrm{NN}_{60\text{–}30,\mathcal{L}_T}$ model was used in earlier unrollment evaluations in § 5.

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

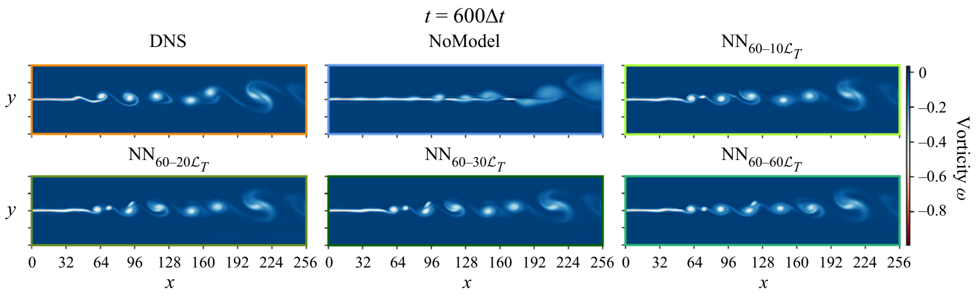


Figure 17. Vorticity comparison of 60-step models on spatial mixing layer simulations at $t = 700\Delta t$ on the test dataset.

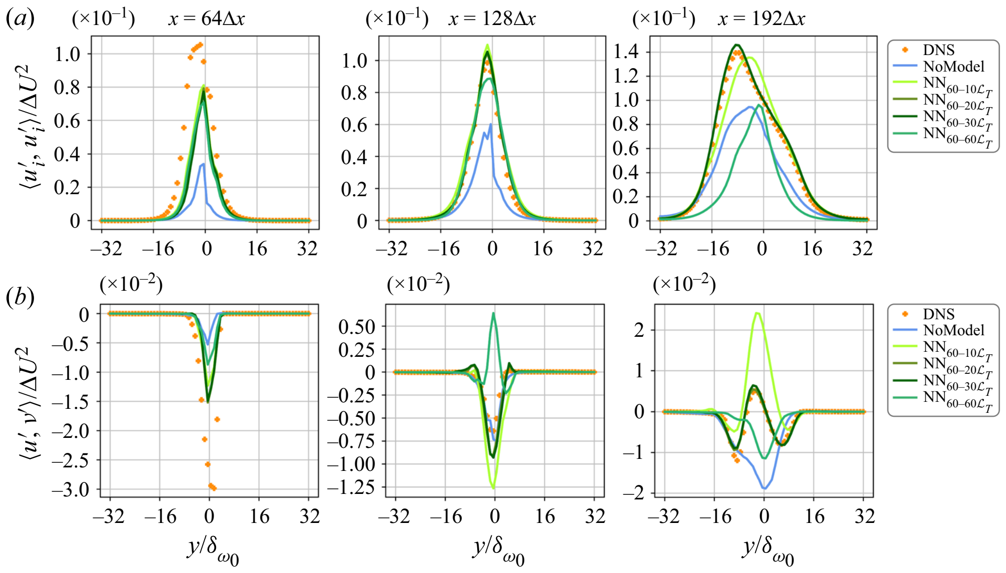


Figure 18. Comparison of downsampled DNS, no-model and learned model simulations with respect to Reynolds-averaged resolved turbulence kinetic energy (*a*) and Reynolds stresses (*b*).

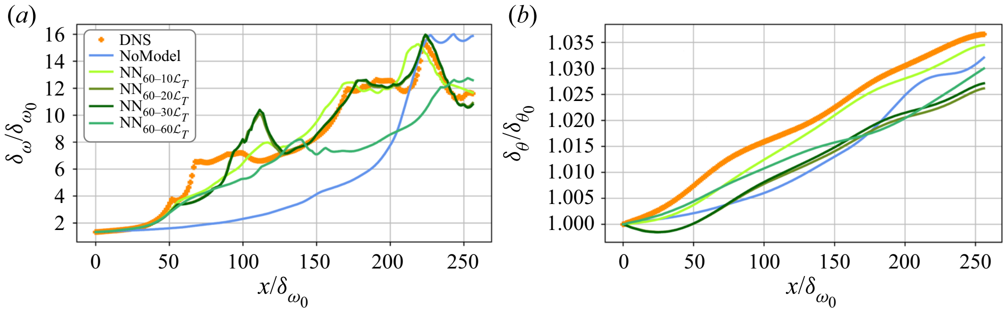


Figure 19. Vorticity and momentum thickness of the downsampled DNS, no-model and learned model simulations.

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

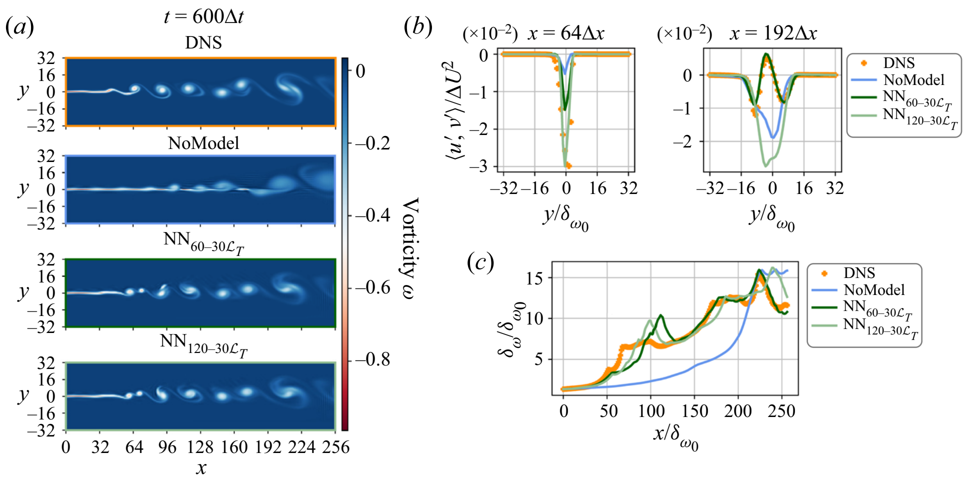


Figure 20. Selected evaluations of the 120-step model with vorticity snapshots in (*a*), Reynolds stresses in (*b*) and vorticity thickness in (*c*).

Another potential problem could be caused by training towards matching frames separated by long timespans. Turbulent flows could potentially loose correlation to the reference data over long temporal horizons, which would render this learning approach driven by simulated DNS data inapplicable. The unrollment times in this paper are, however, far from reaching an uncorrelated state. As shown in the previous evaluations, the 60-step models perform better than their 30-step counterparts, indicating that there is additional information provided by unrolling 60 steps. This shows that the unrolled temporal horizons are far from exhibiting flow decorrelation. Further experiments with even longer unrollments on the spatial mixing layer revealed that no improvement is achieved beyond 60 steps in this case. Figure 20 depicts selected evaluations of a 120-step model, which lack improvements over the 60-step counterpart. While the 120-step model gains accuracy in early upstream cross-sections, the mixing layer shift downstream of the first roll-up is worse in direct comparison.

We also investigated yet longer horizons (180 and 240 steps), but these runs saw a reduced accuracy with respect to some of the evaluations. One explanation is that the flow field is uncorrelated to the DNS data for these long horizons, leading to a diffused learning signal. If the loss was computed on late, uncorrelated frames, we would expect generated gradients to resemble random noise. While earlier frames would still provide valuable information, the random noise from these later frames could prevent the learning of precise corrections. In addition, the longer runs used the same set of hyperparameters as determined for the shorter unrollments, the long horizon runs could also profit from a broader hyperparameter search.

In this section, we have identified gradient instabilities as the main problem when unrolling long temporal horizons. We have introduced a gradient splitting technique that stabilised the training procedure. This is done by splitting the gradient calculation into non-overlapping subranges. For the studied set-ups and 60-step models, a split into two subranges of 30 steps each performed best. One can conclude that longer unrollments pay off in terms of modelling accuracy up to a certain saturation point. In our simulations this saturation point lies at approximately 60 steps, which coincides with the integral time scales of the respective scenarios. Unrolling beyond that saturation point is possible, but

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

	Name	Resolution	Time until t_e	No. Δt	Time per Δt	MSE	Training
IDT	DNS	1024×1024	$14497.7s$	8000	$1.812 \pm 0.432s$	0	—
	NN	128×128	$71.2s$	1000	$0.071 \pm 0.015s$	1.96×10^{-2}	61 h
	NoModel	128×128	$65.5s$	1000	$0.066 \pm 0.066s$	5.45×10^{-2}	—
	—	256×256	$240.6s$	2000	$0.120 \pm 0.027s$	1.07×10^{-2}	—
	—	512×512	$1348.8s$	4000	$0.337 \pm 0.049s$	1.03×10^{-3}	—
TML	DNS	1024×512	$12467.8s$	8000	$1.559 \pm 0.764s$	0	—
	NN	128×64	$155.6s$	1000	$0.156 \pm 0.049s$	1.66×10^{-5}	78 h
	NoModel	128×64	$144.5s$	1000	$0.145 \pm 0.052s$	1.23×10^{-3}	—
	—	256×128	$593.5s$	2000	$0.297 \pm 0.148s$	2.30×10^{-4}	—
	—	340×170	$1054.0s$	2656	$0.395 \pm 0.216s$	1.07×10^{-4}	—
	—	512×256	$2154.9s$	4000	$0.539 \pm 0.259s$	2.51×10^{-5}	—
SML	DNS	2048×512	$81925.3s$	8000	$10.242 \pm 1.144s$	0	—
	NN	256×64	$1813.6s$	1000	$1.815 \pm 0.254s$	2.20×10^{-3}	240 h
	NoModel	256×64	$1815.4s$	1000	$1.817 \pm 0.450s$	2.03×10^{-2}	—
	—	512×128	$3971.3s$	2000	$1.987 \pm 0.348s$	6.22×10^{-3}	—
	—	768×192	$6719.23s$	2667	$2.240 \pm 0.273s$	7.57×10^{-4}	—
	—	1024×256	$12071.5s$	4000	$3.019 \pm 0.245s$	3.13×10^{-4}	—

Table 8. Computational performance comparison over $t_e = 1000\Delta t$ for the used flow scenarios, isotropic decaying turbulence (IDT), temporal mixing layer (TML) and spatial mixing layer (SML); MSE values are evaluated on the velocity field at $500\Delta t$; training time on one GPU.

leads to increased computational effort and may require special treatment, such as a further tuning of the hyperparameters.

7. Computational performance

The development of turbulence models is ultimately motivated by a reduced computational cost, which facilitates numerical simulations in flow scenarios where a DNS is prohibitively expensive. Preceding sections have outlined the corrective capabilities of our learned models. We now seek to put these improvements into perspective by studying the computational cost of our learned models at the inference time. For all of our performance evaluations, an Intel Xeon E5-1650 CPU and a Nvidia GTX 1080Ti GPU are used. We use the computational set-ups from our model evaluation runs on test data in the isotropic turbulence, TML and SML cases in §§ 3, 4 and 5, respectively.

Exactly as before, an $8\times$ scaling factor is deployed on both the spatial resolution and timestep size. We then run the simulations until the time $t_e = 1000\Delta t$ is reached, while recording the required computational time for each timestep. The results are summarised in table 8, where the total simulation time as well as per-timestep values are listed. We also assess the computational cost of a no-model simulation that matches the performance of our models.

The resulting data show that the neural network incurs only a negligible cost of approximately 10 % in comparison with no-model simulations at the same resolution. The learned models clearly outperform the no-model variants in terms of MSEs, and incur only a fraction of the computational cost required for the DNS variants.

In addition, we provide the temporal evolution of the MSE evaluated on resolved turbulence kinetic energies for all three scenarios in figure 21. From this evaluation, we conclude that our method consistently outperforms simulations with a $2\times$ higher

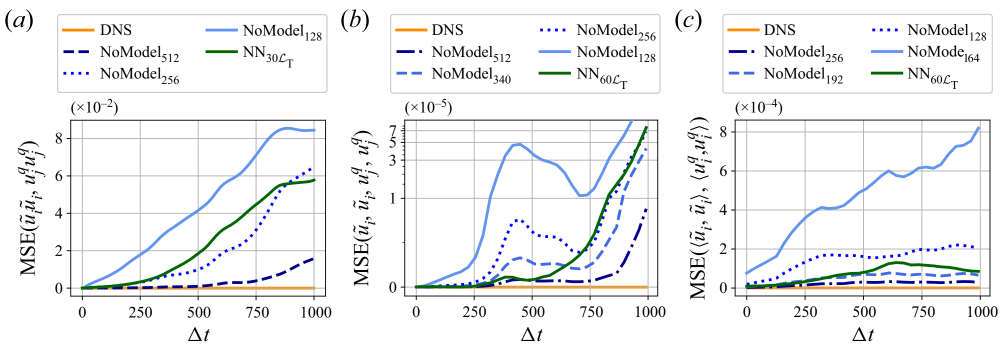


Figure 21. Similarity evolutions over time measured by the MSE on resolved turbulence kinetic energy for randomised turbulence simulations (*a*), TML simulations (*b*) and SML simulations (*c*).

resolution in spatial and temporal dimensions. Additionally, we found our learned models to often be on-par with 4× higher resolved simulations, e.g. in the first half of the temporal mixing layer case. On the basis of the clock times from table 8, this corresponds to a speed up of 3.3 over 2× isotropic turbulence simulations. For the mixing layer cases, the hybrid model on average resembles the performance of 3× reference simulations, which corresponds to a speed-up of 7.0 for the temporal and 3.7 for the spatial mixing layer. For the former, our model even closely matches the performance of a 4× simulation for several hundred timesteps, which represents a speed up of 14.4.

While other works have reported even larger performance improvements (Kochkov *et al.* 2021), we believe that our measurements are representative of real-world scenarios with higher-order solvers. Asymptotically, we also expect even larger payoffs for the high-resolution, three-dimensional simulations that are prevalent in real-world applications.

Naturally, the training of each neural network requires a substantial one-time cost. In our case, the network took 3 to 10 days of training, depending on the individual problem set-up. The required GPU hours for the best-performing models are listed in table 8. The longer unrolled temporal horizons and larger domain increase the required training time for the spatial mixing layer. For the three used set-ups, these training times are equivalent to [120, 118, 22] DNS solves of full length as used in the dataset calculation. However, under the assumption that the learned turbulence model can be employed by multiple users in a larger number of simulations to produce new outputs, this cost will quickly amortise. Especially the most complex spatial mixing layer case shows a favourable relation of training cost to simulation speed up. Additionally, a successful application of this approach to three-dimensional turbulence would make training cheaper in relation to DNS and speed ups larger, due to the scaling through an additional spatial dimension. It is worth noting that our comparisons are based on GPU solvers, and performance is likely to vary on CPU or mixed solvers, where parts of the computation are CPU-based and communication overheads could deteriorate gains.

8. Conclusion

In this paper, we studied adjoint training methods for hybrid solvers that integrate neural networks in a numerical solver. We targeted the modelling of the finest turbulent scales when these cannot be resolved by the simulation grid. The differentiable nature of our implementation of the PISO solver allows us to train the network through multiple

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

unrolled steps. We deem this feature crucial, since we found strong dependence of the model performance and long-term stability on the number of unrolled steps. Our results indicate that covering one integral time scale yields the best performance. Shorter unrollments generally suffer from accuracy and stability issues, while for longer ones the model accuracy saturates and training becomes less efficient. We showcased the application of our method to three different flow scenarios, the two-dimensional isotropic decaying turbulence, the temporally developing mixing layer and the spatially developing mixing layer, whilst keeping the network architecture identical. The optimisation of network parameters yielded good results when optimising towards the $\mathcal{L}_2$-loss, but could be substantially improved through our formulation of the turbulence loss $\mathcal{L}_T$.

When run in inference mode, the simulation based on the learned models trained with our method remained stable for long periods and allowed us to run simulations vastly surpassing the initial training horizon. Our models proved to be in very good agreement with the DNS test datasets when compared on the basis of *a posteriori* statistics. These agreements were obtained despite the fact that the evaluation metrics were not a target of the training optimisation, and that the test datasets constitute an extrapolation from training data. Furthermore, our hybrid approach achieved good results on a wide range of scales, with the Reynolds number varying from $Re = 126$ to $Re = 296$ in the isotropic turbulence case, and the vortex sizes ranging from $7\delta_{\omega 0}$ to $60\delta_{\omega 0}$ in the TML. Similarly, our approach yielded a learned model simulation that remained accurate and stable in a statistically steady test case of the SML. These SML models were trained with a range of perturbation parameters and demonstrated good extrapolation accuracy towards this quantity. In our test cases, the learned model simulation accurately reproduced the turbulence kinetic energy in its spectral distribution as well as its temporal evolution. Furthermore, the learned models captured the turbulent fluctuations, which led to a precise modelling of vortex roll-up and merging events. Our results also demonstrate the importance of unrolling simulator steps during training in achieving high accuracy and stability. Such models are effectively trained by our approach of optimising all subranges of a multi-step training loop divided by gradient stopping. This approach differs from the common practice in machine learning, where gradients of early evaluations of the neural network are usually discarded or re-scaled when gradient clipping is applied (Pascanu *et al.* 2013). Our learned models provide a significant increase in computational performance, where speed ups in terms of computation time of a factor of up to 14 are observed. The additional resources required for model inference are minor and can be justified with the gains in the solution accuracy.

Use of the turbulence loss and large unrollment numbers is motivated by physical and numerical considerations. As introduced in § 2, the components of the turbulence loss are derived from fundamental equations in turbulence theory. As described above, our experiments deem the solver unrollment imperative for training a long-term stable model. On a theoretical level, these principles apply to both two- and three-dimensional flows, which is why we believe that our findings are also of interest to the development of learned turbulence models for three-dimensional flows.

In its current form, our method has several limitations, such as the initial one time cost to train the neural network turbulence model. Also, our tests have focused on regular, Cartesian grids. However, more flexible convolutions (Ummenhofer *et al.* 2019; Sanchez-Gonzalez *et al.* 2020) could be employed to use the presented method on more flexible mesh structures with irregular discretisations. Moreover, even regular CNNs can be extended to take regular, non-uniform and stretched meshes into account (Chen & Thuerey 2021). For instance, this is highly important for wall-bounded flows and fluid–structure interactions. Similarly, further interesting extensions could work towards a differentiable solver that directly trains towards *a posteriori* statistics, or study the modelling capabilities of different network architectures with respect to the modelled turbulent scales.

To summarise, the improvements in accuracy and runtime of our approach render the proposed combination of neural network and numerical solver suitable for a variety of settings. As ground truth data are not restricted to originate from the same solver, they could stem from different numerical schemes such as higher-order spectral methods or even experiments. Furthermore, the learned models offer significant savings when a large quantity of turbulent simulations is required. This is especially important for inverse problems such as flow optimisation tasks. Due to the super-linear scaling of existing solvers, our method also could potentially provide even greater performance benefits when applied to three-dimensional flow fields.

Funding. This work was supported by European Research Council (ERC) Consolidator Grant CoG-2019-863850 (SpaTe).

Data availability statement. The data and code supporting this study are openly available in *disclosed upon publication*.

Declaration of interests. The authors report no conflict of interests.

Author ORCIDs.
Björn List https://orcid.org/0000-0002-1110-9517;
Li-Wei Chen https://orcid.org/0000-0002-0309-2284;
Nils Thuerey https://orcid.org/0000-0001-6647-8910.

Appendix A. The PISO solver details

The governing Navier–Stokes equations (2.1) were solved with a finite-volume approach, which naturally supports the staggered discretisation such that the velocity vector fields are stored at the cell faces, whereas the scalar pressure field is stored at the cell centres. All fluxes were computed to second-order accuracy using a central difference scheme.

A.1. *Governing equations*

The numerical solver follows the method introduced by Issa (1986). Our differentiable hybrid method includes a corrective network forcing $\boldsymbol{f}_{CNN}$ in the predictor step. In contrast, the supervised models cannot take advantage of any differentiable solver operations during training. The corrective forcing from a network trained with the supervised approach $\boldsymbol{f}_{CNN}^{sup}$ must thus be applied after a complete solver step. With the discrete velocity and pressure fields $(\boldsymbol{u}_n, p_n)$ at time t_n, the equations of the PISO solver for both cases read as

$$\boldsymbol{M}\boldsymbol{u}_n^* = \boldsymbol{u}_n - \nabla p_n[+\boldsymbol{f}_{CNN}(\boldsymbol{u}_n, \nabla p_n|\theta)], \tag{A1}$$

$$\nabla \cdot (\boldsymbol{A}^{-1}\nabla p_n^*) = \nabla \cdot \boldsymbol{u}_n^*, \tag{A2}$$

$$\boldsymbol{u}_n^{**} = \boldsymbol{u}_n^{*} - \boldsymbol{A}^{-1}\nabla p_n^{*}, \tag{A3}$$

$$\nabla \cdot (\boldsymbol{A}^{-1}\nabla p_n^{**}) = \nabla \cdot (\boldsymbol{H}\boldsymbol{u}_n^{**}), \tag{A4}$$

$$\boldsymbol{u}_n^{***} = \boldsymbol{u}_n^{**} + \boldsymbol{A}^{-1}(\boldsymbol{H}(\boldsymbol{u}_n^{**} - \boldsymbol{u}_n^{*}) - \nabla p_n^{**}), \tag{A5}$$

$$p_{n+1} = p_n + p^* + p^{**}, \tag{A6}$$

$$\boldsymbol{u}_{i+1} = \boldsymbol{u}_n^{***}[+\boldsymbol{f}_{CNN}^{sup}(\boldsymbol{u}_n^{***}, \nabla p_{n+1}|\theta^{sup})], \tag{A7}$$

where the corrective forcings $\boldsymbol{f}_{CNN}$ and $\boldsymbol{f}_{CNN}^{sup}$ are never applied at the same time, but share this set of equations for brevity. The matrix $\boldsymbol{M}$ represents the discretised advection, diffusion and temporal integration, and matrix $\boldsymbol{A}$ contains the diagonal entries of $\boldsymbol{M}$ such that $\boldsymbol{M} = \boldsymbol{A} + \boldsymbol{H}$. The network weights are represented by θ.

The optimisation loss is applied to the output of a solver step. Using the downsampling $(\tilde{\boldsymbol{u}}_n, \tilde{p}_n) = q(\boldsymbol{u}_n, p_n) = \tilde{q}_n$ as introduced in § 2, we can abbreviate a solver step by $\tilde{q}_{n+1} = \mathcal{S}_\tau(\tilde{q}_n, \tilde{\boldsymbol{f}}_{CNN,n})$ in the case of the differentiable model, and by $\tilde{q}_{n+1} = \mathcal{S}_\tau(\tilde{q}_n) + \tilde{\boldsymbol{f}}_{CNN,n}^{sup}$ in the case of the supervised model. The parameter τ describes the temporal increment of a solver step as $\Delta t = \tau \Delta t_{DNS}$. At this stage, it becomes obvious that optimising $\min_\theta[\mathcal{L}(\tilde{q}_{n+\tau}, \mathcal{S}_\tau(\tilde{q}_n, \boldsymbol{f}_{CNN,n}))]$ with the differentiable model, as introduced in (2.3), requires the computation of

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \tilde{q}_{n+1}} \frac{\partial \tilde{q}_{n+1}}{\partial \boldsymbol{f}_{CNN}} \frac{\partial \boldsymbol{f}_{CNN}}{\partial \theta}, \tag{A8}$$

which in turn requires the differentiation of a solver step. In contrast, optimising a supervised model with $\min_\theta[\mathcal{L}(\boldsymbol{u}_{n+\tau}, \mathcal{S}_\tau(\tilde{\boldsymbol{u}}_n, \tilde{p}_n) + \boldsymbol{f}_{CNN}^{sup}(\mathcal{S}_\tau(\tilde{\boldsymbol{u}}_n, \tilde{p}_n)))]$ has to compute

$$\frac{\partial \mathcal{L}}{\partial \theta^{sup}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_{CNN}^{sup}} \frac{\partial \boldsymbol{f}_{CNN}^{sup}}{\partial \theta^{sup}}, \tag{A9}$$

which can be achieved without a differentiable solver.

When n solver steps are unrolled during training of differentiable models, this yields the optimisation procedure as introduced in (2.4)

$$\min_\theta \left(\sum_{s=0}^{m} \mathcal{L}(\tilde{q}_{n+s\tau}, \mathcal{S}_\tau^s(\tilde{q}_n, \tilde{\boldsymbol{f}}_n)) \right). \tag{A10}$$

During back-propagation, the gradients based on the losses at all (intermediate) steps are calculated and propagated through all previously unrolled forward steps, accumulating gradients with respect to all network forces on the way back. For a loss on an (intermediate) solver step $\mathcal{L}^s = \mathcal{L}(\tilde{q}_{n+s\tau}, \mathcal{S}_\tau^s(\tilde{q}_n, \tilde{\boldsymbol{f}}_n))$, the following gradient calculation arises:

$$\frac{\partial \mathcal{L}^s}{\partial \theta} = \sum_{B=1}^{s} \left[\frac{\partial \mathcal{L}^s}{\partial \tilde{q}_{n+s}} \left(\prod_{b=s}^{B+1} \frac{\partial \tilde{q}_{n+b}}{\partial \tilde{q}_{n+b-1}} \right) \frac{\partial \tilde{q}_{n+B}}{\partial \boldsymbol{f}_{CNN}^{B-1}} \frac{\partial \boldsymbol{f}_{CNN}^{B-1}}{\partial \theta} \right], \tag{A11}$$

where $\boldsymbol{f}_{CNN}^B$ denotes the network forcing in the Bth step. As explained in § 6, we use a custom gradient splitting technique that splits the back-propagation into subranges. The gradients are only back-propagated within a subrange, and set to zero when they cross a subrange boundary. When using gradient subranges of length r, the gradient calculation

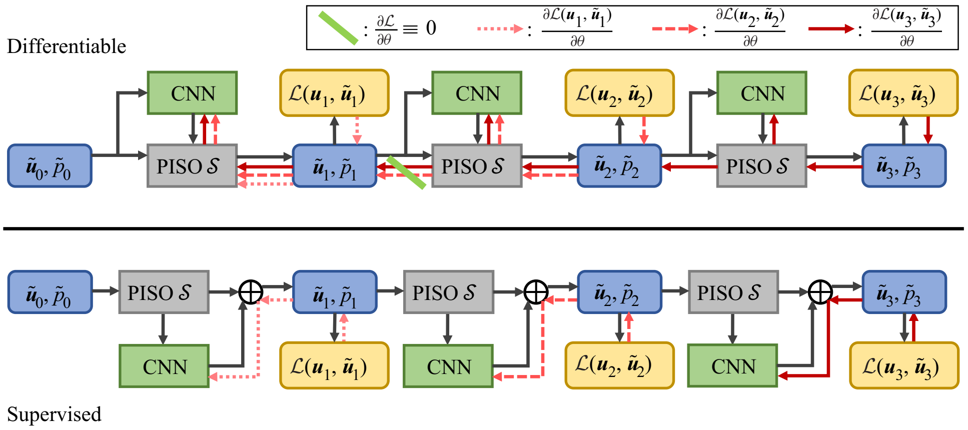

Differentiable
$\frac{\partial \mathcal{L}}{\partial \theta} \equiv 0$
$\frac{\partial \mathcal{L}(\boldsymbol{u}_1, \tilde{\boldsymbol{u}}_1)}{\partial \theta}$
$\frac{\partial \mathcal{L}(\boldsymbol{u}_2, \tilde{\boldsymbol{u}}_2)}{\partial \theta}$
$\frac{\partial \mathcal{L}(\boldsymbol{u}_3, \tilde{\boldsymbol{u}}_3)}{\partial \theta}$
CNN
PISO $\mathcal{S}$
$\tilde{\boldsymbol{u}}_0, \tilde{p}_0$
$\tilde{\boldsymbol{u}}_1, \tilde{p}_1$
$\tilde{\boldsymbol{u}}_2, \tilde{p}_2$
$\tilde{\boldsymbol{u}}_3, \tilde{p}_3$
$\mathcal{L}(\boldsymbol{u}_1, \tilde{\boldsymbol{u}}_1)$
$\mathcal{L}(\boldsymbol{u}_2, \tilde{\boldsymbol{u}}_2)$
$\mathcal{L}(\boldsymbol{u}_3, \tilde{\boldsymbol{u}}_3)$
Supervised

Figure 22. Visualisation of gradient back-propagation, comparing differentiable and supervised set-ups; displayed is a 3-step set-up; the loss gradients from the last step are propagated through all previous steps and towards all previous network outputs; if the back-propagation is split into subranges, the gradients of the simulation state are set to zero, visualised by '\'.

gives

$$\frac{\partial \mathcal{L}^s}{\partial \theta} = \sum_{B=\lfloor s \backslash r \rfloor * r}^{s} \left[\frac{\partial \mathcal{L}^s}{\partial \tilde{q}_{n+s}} \left(\prod_{b=s}^{B+1} \frac{\partial \tilde{q}_{n+b}}{\partial \tilde{q}_{n+b-1}} \right) \frac{\partial \tilde{q}_{n+B}}{\partial \boldsymbol{f}_{CNN}^{B-1}} \frac{\partial \boldsymbol{f}_{CNN}^{B-1}}{\partial \theta} \right], \tag{A12}$$

where $\lfloor s \backslash r \rfloor$ denotes the integer division of s by r. This formulation can be easily implemented by setting the gradient of the simulation state to zero at the subrange boundaries, as visualised in figure 22.

Supervised models train on the optimisation

$$\min_{\theta} \left(\sum_{s=0}^{m} \mathcal{L}(\tilde{q}_{n+s\tau}, [\mathcal{S}_\tau(\tilde{q}_n) + \tilde{\boldsymbol{f}}_{CNN,n}^{sup}]^s) \right), \tag{A13}$$

where the expression in $[\cdots]^s$ denotes the recurrent application of a solver step with a supervised model. We abbreviate for simplicity $\mathcal{L}^{sup,s} = \mathcal{L}(\tilde{q}_{i+s\tau}, [\mathcal{S}_\tau(\tilde{q}_i) + \tilde{\boldsymbol{f}}_{CNN,i}^{sup}]^s)$. The gradients of these losses are only calculated within the locality of an (intermediate) solution and are thus a trivial extension of (A9)

$$\frac{\partial \mathcal{L}^{sup,s}}{\partial \theta^{sup}} = \frac{\partial \mathcal{L}^{sup,s}}{\partial \boldsymbol{f}_{CNN}^{sup,s}} \frac{\partial \boldsymbol{f}_{CNN}^{sup,s}}{\partial \theta^{sup}}. \tag{A14}$$

The training unrollment and its gradient back-propagation for differentiable hybrid as well as supervised models are visualised in figure 22.

A.2. *Implementation*

The presented method was implemented using the tensor operation library *TensorFlow* (Abadi 2016). This framework supports the GPU-based execution of various linear algebra operations, however, it does not support sparse matrix data at the time of this project. Consequently, a series of custom operations surrounding the linear solves for advection–diffusion and pressure in the PISO scheme were added to facilitate an efficient GPU-based execution of the solver. The back-propagation gradients of the custom linear solves $Ax = b$ were linearised around their respective matrices and thus read as $A^{\mathrm{T}}\hat{b} = \hat{x}$, where $\hat{x}$ and $\hat{b}$ represent the incoming and outgoing back-propagation gradients of the linear solve operation. This yields a solver that can flexibly change the number of steps unrolled during training (only limited by GPU memory and computation time), and account for any loss functions or network architectures. Access to our code is provided through the following GitHub page: https://github.com/tum-pbs/differentiable-piso

A.3. *Solver verification*

Our implementation is verified on two standardised simulations. Firstly, we study the grid convergence properties on the two-dimensional Taylor–Green vortex decay. This flow scenario is simulated on a periodic, square domain of size $(L_x, L_y) = (2\pi, 2\pi)$ and initialised with the analytical solution

$$\left.\begin{aligned} \hat{u}(x, y, t) &= -\cos(x)\sin(y)\exp\left(\frac{-2t}{Re}\right), \\ \hat{v}(x, y, t) &= \sin(x)\cos(y)\exp\left(\frac{-2t}{Re}\right), \\ \hat{p}(x, y, t) &= -\frac{\cos(2x)+\cos(2y)}{4}\exp\left(\frac{-4t}{Re}\right), \end{aligned}\right\} \tag{A15}$$

where the Reynolds number is set to $Re = 10$. The grid resolution is varied as $N_x = N_y = [8, 16,\ 32,\ 64,\ 128]$. The governing equations (2.1) are integrated until $t = 2$ is reached, while a small timestep of $\Delta t = 10^{-3}$ is chosen for all resolutions. Figure 23 depicts the normalised error of the numerical solution $\boldsymbol{u} = (u,\ v)^{\mathrm{T}}$ with respect to the analytical solution from (A15), computed as $L_2 = (\sum_{i,j}(u_{i,j} - \hat{u}_{i,j})^2 + (v_{i,j} - \hat{v}_{i,j})^2)/N_xN_y$. This demonstrates second-order convergence of our implementation. Secondly, we verify the solver on numerical benchmark data for a lid-driven cavity flow. This case consists of a fluid domain of size $(L_x, L_y) = (1, 1)$ with no-slip wall boundaries enforcing $u(y=0) = 0$, $v(x=0) = 0$, $v(x=1) = 0$ and $u(y=1) = 1$ for the lid. Our simulations are performed at two different Reynolds numbers. For $Re = 100$, the steady state is approximated by running the simulation until $t = 10$ on a $(N_x, N_y) = 128, 128$ grid. We verify our solver by comparing the velocities at the domain-centre cross-sections with the benchmark solutions reported by Ghia, Ghia & Shin (1982). The results are shown in figure 24(*a*). Similarly, the evaluations for simulations at $Re = 1000$ on 128×128 and 256×256 grids are shown in figure 24(*b*). Both cases show good agreement with the benchmark data for sufficiently high resolutions.

Appendix B. Convolutional neural network

Our turbulence models are parameterised by a CNN, and thus formed by the kernel weights in each convolutional layer. Our set-up utilises seven layers with kernel sizes $[7,\ 5,\ 5,\ 3,\ 3,\ 1,\ 1]$ and leaky ReLU activation functions. The input to the network consists of the velocity and pressure gradient vector fields, yielding four channels in total. The layers then operate on $[8, 8,\ 16,\ 32,\ 32,\ 32]$ channels respectively and output a forcing vector field with two channels. Consequently, the network consist of $\sim 82 \times 10^3$ trainable weights contained in the kernels.

The structure of this network resembles an encoder network, where the larger kernel size in the first layers increases the receptive field of the convolution. The potential complexity

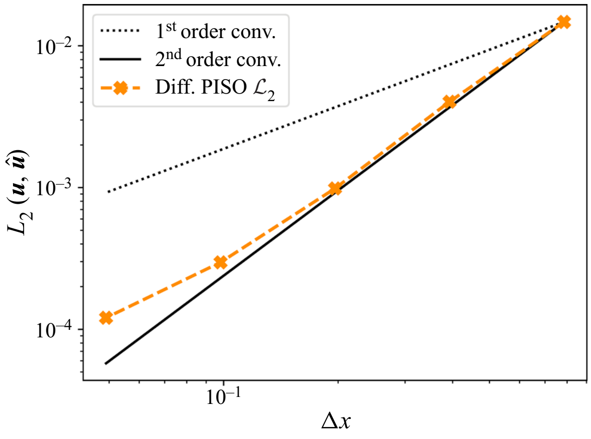


Figure 23. Grid convergence study, the numerical error on the Taylor–Green vortex with respect to analytical data converges with second order.

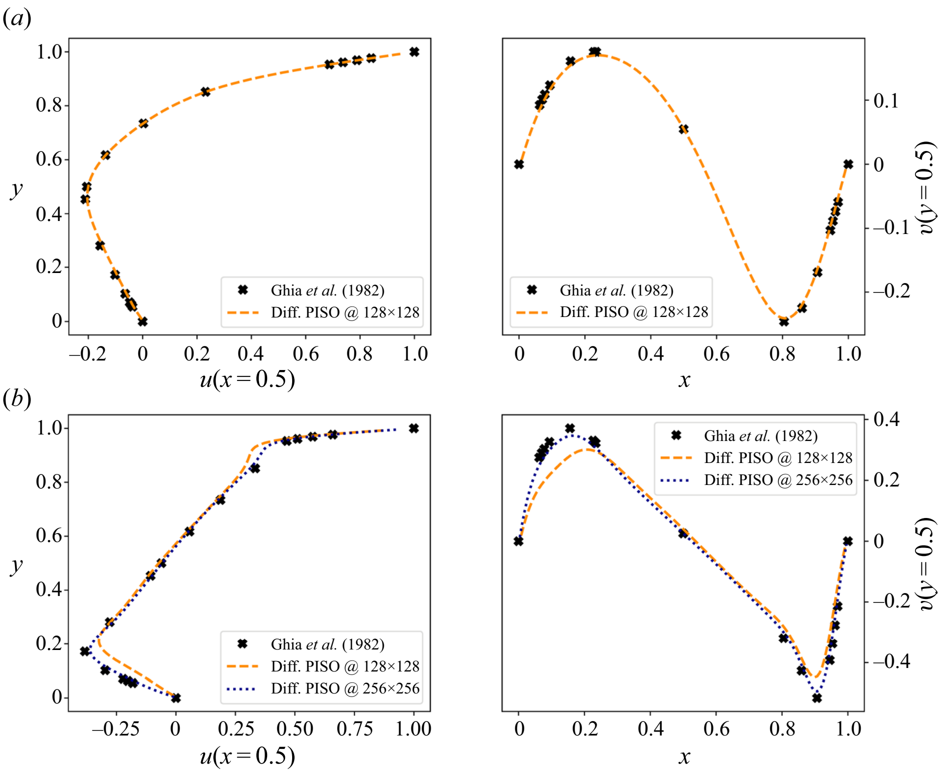


Figure 24. Lid-driven cavity verification case, figures show the domain-centre velocities for $Re = 100$ in (*a*), and $Re = 1000$ in (*b*), in comparison with numerical benchmark data by Ghia *et al.* (1982).

of the function parameterised by the network is largely dependent on the channel widths and layer count. We have found the described architecture to work well for turbulence modelling, without overfitting to training data, as larger models are more likely to do.

By the nature of the discrete convolution operation, the output size shrinks with each layer. At periodic boundaries this can be counteracted by padding the input with real data. At other boundaries, where no periodicity is enforced, no padding procedure is used on the input to avoid feeding unphysical data. In these cases, the output of the CNN does not coincide with the grid dimensions and is accordingly padded with zeros. Prior to training, the weights were initialised using the Glorot Normal initialisation.

Appendix C. Training procedure

Our method trains neural networks to model the effect of turbulent motion. These effects are implicitly learnt from high-resolution DNS simulations by reproducing their behaviour. Our training procedure uses the commonly chosen Adam optimiser (Kingma & Ba 2015). During one optimisation step o, Adam takes the loss gradient as specified in Appendix A.2 and applies a weight update according to

$$\begin{aligned} &g_o \leftarrow \frac{\partial \mathcal{L}_\wr}{\partial \theta_{o-1}} \\ &m_o \leftarrow \beta_1 m_{o-1} + (1-\beta_1) g_o \\ &v_o \leftarrow \beta_2 v_{o-1} + (1-\beta_2) g_o^2 \\ &\hat{m}_o \leftarrow m_o/(1-\beta_1^o) \\ &\hat{v}_o \leftarrow v_o/(1-\beta_2^o) \\ &\theta_o \leftarrow \theta_o - \alpha \frac{\hat{m}_o}{\sqrt{\hat{v}_o + \epsilon}}, \end{aligned}$$

where m_o and v_o are exponential moving averages approximating the mean and variance of the gradient. To account for the initialisation error in these approximates, the corrected variables $\hat{m}_o$ and $\hat{v}_o$ are introduced; see the original publication for further details. We set the bias corrections to the standard values $\beta_1 = 0.9$, $\beta_2 = 0.999$. The networks were trained with a learning rate of 1×10^{-5} and a learning-rate decay factor of 0.4. We found that the training procedure was stable for learning rates in the neighbourhood of that value, however, no extensive hyper-parameter tuning was performed. On the contrary, we found the unrollment number s (see (2.4)) to have great effect on the training procedure. Newly initialised models can cause the accumulation of erroneous structures and subsequently solver divergence in long unroll times. To mitigate this effect, the models trained on more than 10 steps were initialised with a pre-trained network from a 10-step model. The parameter optimisations were run until no further significant decrease in loss values is observed.

Appendix D. Large eddy simulation with the Smagorinsky model

A series of tests were conducted to select an appropriate value for the Smagorinsky coefficient used in the IDT simulation in § 3. We ran simulations with our usual downscaling of $8\times$ in space and time and coefficients from $C_s = [0.17, 0.08, 0.02, 0.008, 0.002]$. The velocity MSEs of these simulations with respect to the DNS test data after $100\Delta t$ were evaluated to $[12.21,\ 6.824,\ 4.320,\ 4.256,\ 4.364] \times 10^{-3}$. Based on that analysis, $C_s = 0.008$ was chosen for further consideration. This value is relatively low in comparison with other common choices, such as the default coefficient of $C_s = 0.17$ for three-dimensional turbulence (Pope 2000). Since two-dimensional isotropic turbulence is largely dependent on the backscatter effect that transfers energy from small to large scales, lower C_s are applicable (Smith *et al.* 1996). With the strictly dissipative behaviour of the Smagorinsky model, larger C_s leads to an overly powerful dampening of fine-scale motions that quickly decreases the turbulence kinetic energy. While backscatter is important to

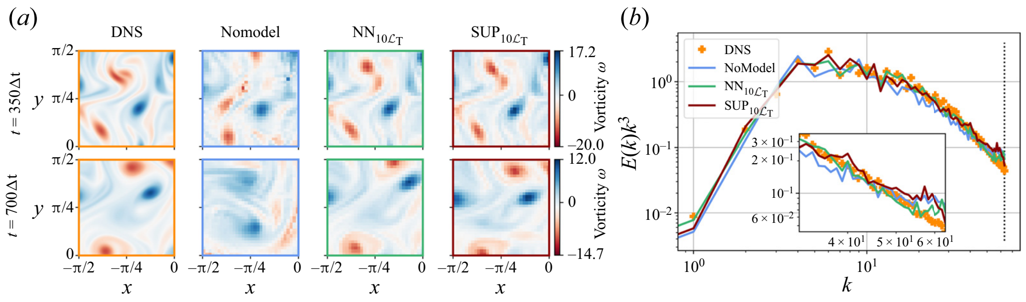


Figure 25. Comparison of DNS, no-model and learned model simulations trained with the adjoint-based method and with a supervised method on IDT; evaluation with respect to vorticity (*a*) and resolved turbulence kinetic energy spectra (*b*).

many flow scenarios (Biferale, Musacchio & Toschi 2012), especially three-dimensional turbulence scenarios may rather have significant forward diffusion, which would be more favourable towards dissipative models like the Smagorinsky model (Kraichnan 1967). Nevertheless, this showcases an inherent benefit of learned turbulence models, where no scenario-dependent modelling assumptions are necessary.

Appendix E. Supervised models

A core point of the experiments in the main section is the temporal unrollment during training, and substantial accuracy improvement of the differentiable models is achieved by this procedure. As illustrated in Appendix A.1, the temporal unrollment has less severe effects on the optimisation equations of supervised models. Despite this, considerable accuracy improvements are achieved by exposing the supervised training to multiple steps. Nevertheless, models trained with a differentiable approach outperform these improved supervised models, when all other parameters are kept constant, as revealed by our experiments on supervised models. For this, we trained 10-step supervised models for the IDT and temporal mixing layer cases. Figures 25 and 26 depict evaluations on the spectral energy for isotropic turbulence, Reynolds stresses and turbulence kinetic energy for the temporal mixing layer, as well as vorticity visualisations for both. For the isotropic case, the supervised model comes remarkably close to the differentiable counterpart, and only shows slight over-estimation of fine-scale energies. For more complex flow like TMLs, it is clearer that differentiable models outperform supervised ones.

Appendix F. Loss ablation

To test the effects of the loss terms introduced in § 2, we perform an ablation study on the loss term. A series of 10-step models are trained with identical initialisation, data shuffling and learning rate, but variations in loss composure. These tests are conducted on all three flow scenarios. The loss factors λ are identical to the ones used in the main sections, where the values are set to yield similar loss contributions for each loss term. An exception is λ_2, which was chosen to give a $10\times$ larger contribution in order to steer an initialised network into reproducing DNS structures. We then perform evaluations based on our out-of-sample test datasets. The results are summarised in table 9. Our evaluations include three metrics. The first is an instantaneous MSE on the velocity field. Secondly, we assess the performance with respect to the turbulence kinetic energy by using an instantaneous MSE for isotropic turbulence, an MSE on spatially averaged energy for the TML and the

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

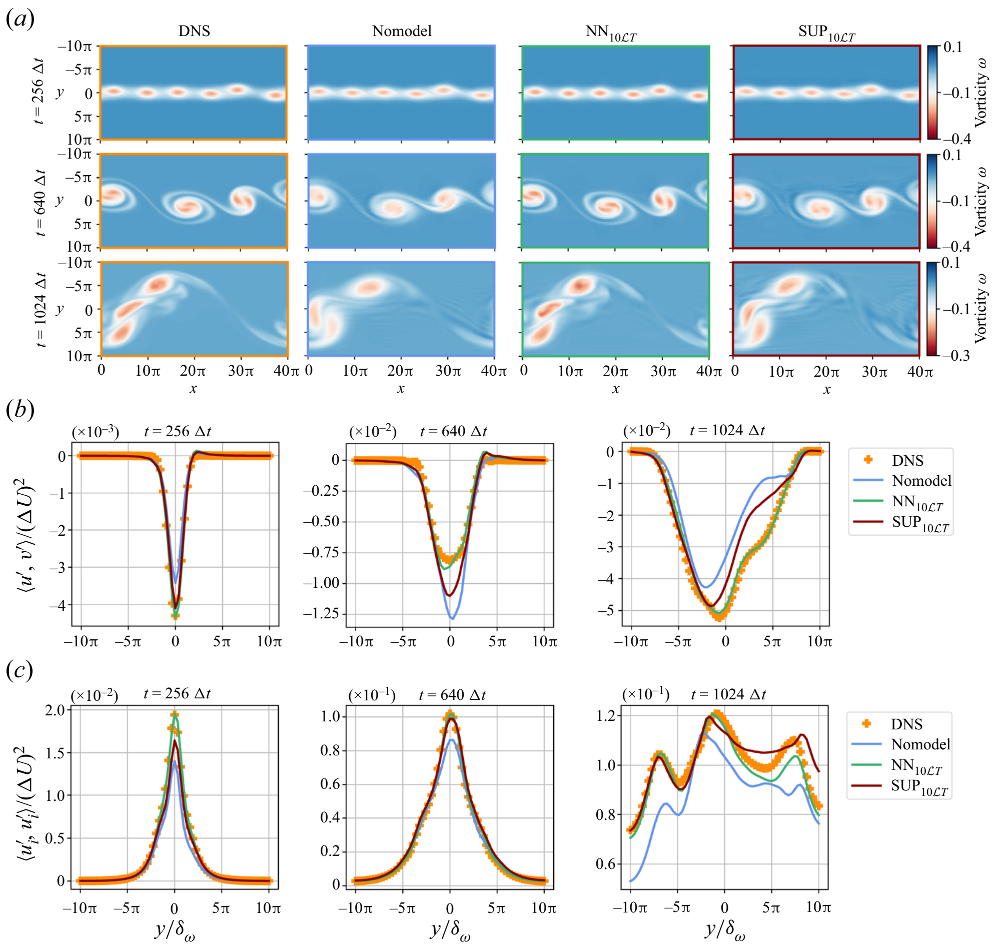


Figure 26. Comparison of DNS, no-model and learned model simulations trained with the adjoint-based method and with a supervised method on TMLs; evaluation with respect to vorticity (*a*), resolved Reynolds stresses (*b*) and resolved turbulence kinetic energy (*c*).

MSE on temporally averaged data for the SML. Lastly, we assess the energy distribution over spectral wavelengths, which is based on a two-dimensional evaluation for isotropic turbulence, a cross-section analysis for the TML and a centreline analysis for the SML. Additionally, two temporal snapshots were considered, a short $64\Delta t$ distance and a longer one, which was set to $1000\Delta t$ for all set-ups except for the SML, where stability concerns limited the horizon to $500\Delta t$.

The results indicate that the baseline $\mathcal{L}_2$ loss only performs well on short temporal horizons, while its performance deteriorates over longer evaluations. The tests on decaying turbulence and TMLs generally show best results with a combination of $\mathcal{L}_2$, $\mathcal{L}_E$ and $\mathcal{L}_S$ over longer temporal horizons. The only exception is the spectral energy analysis in the TML, where an addition of $\mathcal{L}_{MS}$ outperforms this combination by a small margin. Due to the fact that this margin is minor compared with the improvements of the $\mathcal{L}_2$, $\mathcal{L}_E$, $\mathcal{L}_S$ combination on the long horizons, we conclude that including the temporal averaging loss is not beneficial in the flow scenarios that are not statistically steady. In contrast, the evaluations of the SML reveals that incremental additions of the turbulence loss terms $\mathcal{L}_E$,

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

		Time t_1			Time t_2		
	Loss	MSE($\boldsymbol{u}$)	MSE($\boldsymbol{k}$)	$\sum(E(k)_{\boldsymbol{u}}/E(k)_{\tilde{\boldsymbol{u}}})-1$	MSE($\boldsymbol{u}$)	MSE($\boldsymbol{k}$)	$\sum(E(k)_{\boldsymbol{u}}/E(k)_{\tilde{\boldsymbol{u}}})-1$
IDT	$\mathcal{L}_2$	$\mathbf{5.45 \times 10^{-4}}$	$\mathbf{1.96 \times 10^{-4}}$	7.256	0.182	0.0317	−19.27
	$\mathcal{L}_2, \mathcal{L}_E$	5.80×10^{-4}	2.05×10^{-4}	**3.598**	0.168	0.0276	−19.14
	$\mathcal{L}_2, \mathcal{L}_E, \mathcal{L}_\mathcal{S}$	5.79×10^{-4}	2.04×10^{-4}	3.682	**0.166**	**0.0271**	**−18.92**
	$\mathcal{L}_2, \mathcal{L}_E, \mathcal{L}_\mathcal{S}, \mathcal{L}_{MS}$	5.70×10^{-4}	2.01×10^{-4}	4.373	0.182	0.0332	−21.20
TML	$\mathcal{L}_2$	9.92×10^{-7}	$\mathbf{1.22 \times 10^{-8}}$	**−0.307**	1.39×10^{-3}	4.53×10^{-5}	5.109
	$\mathcal{L}_2, \mathcal{L}_E$	1.44×10^{-6}	2.28×10^{-8}	−0.613	1.10×10^{-3}	5.44×10^{-5}	4.951
	$\mathcal{L}_2, \mathcal{L}_E, \mathcal{L}_\mathcal{S}$	8.59×10^{-7}	1.97×10^{-8}	−1.261	$\mathbf{2.15 \times 10^{-4}}$	$\mathbf{1.78 \times 10^{-5}}$	4.155
	$\mathcal{L}_2, \mathcal{L}_E, \mathcal{L}_\mathcal{S}, \mathcal{L}_{MS}$	$\mathbf{4.83 \times 10^{-7}}$	1.57×10^{-8}	−1.572	9.11×10^{-4}	2.85×10^{-5}	**4.142**
SML	$\mathcal{L}_2$	2.29×10^{-4}	4.11×10^{-6}	62.01	0.0243	2.67×10^{-4}	500.7
	$\mathcal{L}_2, \mathcal{L}_E$	1.60×10^{-4}	4.14×10^{-6}	52.15	0.0260	2.21×10^{-4}	454.5
	$\mathcal{L}_2, \mathcal{L}_E, \mathcal{L}_\mathcal{S}$	1.07×10^{-4}	3.46×10^{-6}	50.85	0.0127	6.72×10^{-5}	457.7
	$\mathcal{L}_2, \mathcal{L}_E, \mathcal{L}_\mathcal{S}, \mathcal{L}_{MS}$	$\mathbf{3.86 \times 10^{-5}}$	$\mathbf{1.34 \times 10^{-6}}$	**27.65**	**0.0025**	$\mathbf{2.74 \times 10^{-5}}$	**216.0**

Table 9. Loss ablation study for the used flow scenarios, IDT, TML and SML; $t_1 = 64\Delta t = 512\Delta t_{DNS}$ and $t_2 = [1000,\ 1000,\ 500]\Delta t$ for IDT, TML, SML, respectively; MSE(k) is evaluated on instantaneous turbulent kinetic energy fields for IDT, and on spatially/temporally averaged fields for TML and SML; $\sum(E(k)_{\boldsymbol{u}}/E(k)_{\tilde{\boldsymbol{u}}})-1$ is evaluated on two-dimensional spectral analysis for IDT, cross-sectional spectra for TML, and centreline spectra for SML.

$\mathcal{L}_S$ and $\mathcal{L}_{MS}$ yield better performance for each addition. Thus, we conclude that using all loss terms is beneficial in this case.

References

Abadi, M. *et al.* 2016 Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283. USENIX Association.

Aizinger, V., Korn, P., Giorgetta, M. & Reich, S. 2015 Large-scale turbulence modelling via α-regularisation for atmospheric simulations. *J. Turbul.* **16** (4), 367–391.

Albawi, S., Mohammed, T.A. & Al-Zawi, S. 2017 Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6. IEEE.

Argyropoulos, C.D. & Markatos, N.C. 2015 Recent advances on the numerical modelling of turbulent flows. *Appl. Math. Model.* **39** (2), 693–732.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J. & Kolter, J.Z. 2018 End-to-end differentiable physics for learning and control. *Adv. Neural Inform. Proc. Syst.* **31**, 7178–7189.

Barton, I.E. 1998 Comparison of simple-and piso-type algorithms for transient flows. *Intl J. Numer. Meth. Fluids* **26** (4), 459–483.

Beck, A., Flad, D. & Munz, C.-D. 2019 Deep neural networks for data-driven les closure models. *J. Comput. Phys.* **398**, 108910.

Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K. & Kaushik, S. 2019 Prediction of aerodynamic flow fields using convolutional neural networks. *Comput. Mech.* **64** (2), 525–545.

Biferale, L., Musacchio, S. & Toschi, F. 2012 Inverse energy cascade in three-dimensional isotropic turbulence. *Phys. Rev. Lett.* **108**, 164501.

Bozzi, S., Dominissini, D., Redaelli, A. & Passoni, G. 2021 The effect of turbulence modelling on the assessment of platelet activation. *J. Biomech.* **128**, 110704.

Chasnov, J.R. 1997 On the decay of two-dimensional homogeneous turbulence. *Phys. Fluids* **9** (1), 171–180.

Chen, L.-W. & Thuerey, N. 2021 Towards high-accuracy deep learning inference of compressible turbulent flows over aerofoils. Preprint, arXiv:2109.02183.

Cheng, Y., Giometto, M., Kauffmann, P., Lin, L., Cao, C., Zupnick, C., Li, H., Li, Q., Abernathey, R. & Gentine, P. 2019 Deep learning for subgrid-scale turbulence modeling in large-eddy simulations of the atmospheric boundary layer. Preprint, arXiv:1910.12125.

Choi, H. & Moin, P. 2012 Grid-point requirements for large eddy simulation: Chapman's estimates revisited. *Phys. Fluids* **24** (1), 011702.

Duraisamy, K., Iaccarino, G. & Xiao, H. 2019 Turbulence modeling in the age of data. *Annu. Rev. Fluid Mech.* **51**, 357–377.

Eivazi, H., Guastoni, L., Schlatter, P., Azizpour, H. & Vinuesa, R. 2021 Recurrent neural networks and Koopman-based frameworks for temporal predictions in a low-order model of turbulence. *Intl J. Heat Fluid Flow* **90**, 108816.

Ghia, U., Ghia, K.N. & Shin, C.T. 1982 High-re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method. *J. Comput. Phys.* **48** (3), 387–411.

Giles, M.B., Duta, M.C., Muller, J. -D. & Pierce, N.A. 2003 Algorithm developments for discrete adjoint methods. *AIAA J.* **41** (2), 198–205.

Guastoni, L., Güemes, A., Ianiro, A., Discetti, S., Schlatter, P., Azizpour, H. & Vinuesa, R. 2021 Convolutional-network models to predict wall-bounded turbulence from wall quantities. *J. Fluid Mech.* **928**, A27.

Ho, C.-M. & Huang, L.-S. 1982 Subharmonics and vortex merging in mixing layers. *J. Fluid Mech.* **119**, 443–473.

Holl, P., Thuerey, N. & Koltun, V. 2020 Learning to control PDEs with differentiable physics. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyeSin4FPB.

Issa, R.I. 1986 Solution of the implicitly discretised fluid flow equations by operator-splitting. *J. Comput. Phys.* **62** (1), 40–65.

Kim, S.-W. & Benson, T.J. 1992 Comparison of the smac, piso and iterative time-advancing schemes for unsteady flows. *Comput. Fluids* **21** (3), 435–454.

Kingma, D.P. & Ba, J. 2015 Adam: a method for stochastic optimization. In *International Conference on Learning Representations*. arXiv:1412.6980.

Ko, J., Lucor, D. & Sagaut, P. 2008 Sensitivity of two-dimensional spatially developing mixing layers with respect to uncertain inflow conditions. *Phys. Fluids* **20** (7), 077102.

Kochkov, D., Smith, J.A., Alieva, A., Wang, Q., Brenner, M.P. & Hoyer, S. 2021 Machine learning–accelerated computational fluid dynamics. *Proc. Natl Acad. Sci.* **118** (21), e2101784118.

https://doi.org/10.1017/jfm.2022.738 Published online by Cambridge University Press

KRAICHNAN, R.H. 1967 Inertial ranges in two-dimensional turbulence. *Phys. Fluids* **10** (7), 1417–1423.

LAPEYRE, C.J., MISDARIIS, A., CAZARD, N., VEYNANTE, D. & POINSOT, T. 2019 Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates. *Combust. Flame* **203**, 255–264.

LI, Z., KOVACHKI, N., AZIZZADENESHELI, K., LIU, B., BHATTACHARYA, K., STUART, A. & ANANDKUMAR, A. 2020 Fourier neural operator for parametric partial differential equations. Preprint, arXiv:2010.08895.

LILLY, D.K. 1971 Numerical simulation of developing and decaying two-dimensional turbulence. *J. Fluid Mech.* **45** (2), 395–415.

LING, J., KURZAWSKI, A. & TEMPLETON, J. 2016 Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *J. Fluid Mech.* **807**, 155–166.

LUO, W., LI, Y., URTASUN, R. & ZEMEL, R. 2016 Understanding the effective receptive field in deep convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 4905–4913.

LUSCH, B., KUTZ, J.N. & BRUNTON, S.L. 2018 Deep learning for universal linear embeddings of nonlinear dynamics. *Nat. Commun.* **9** (1), 4950.

MACART, J.F., SIRIGNANO, J. & FREUND, J.B. 2021 Embedded training of neural-network subgrid-scale turbulence models. *Phys. Rev. Fluids* **6** (5), 050502.

MAULIK, R., SAN, O., RASHEED, A. & VEDULA, P. 2019 Subgrid modelling for two-dimensional turbulence using neural networks. *J. Fluid Mech.* **858**, 122–144.

MICHALKE, A. 1964 On the inviscid instability of the hyperbolictangent velocity profile. *J. Fluid Mech.* **19** (4), 543–556.

NOVATI, G., DE LAROUSSILHE, H.L. & KOUMOUTSAKOS, P. 2021 Automating turbulence modelling by multi-agent reinforcement learning. *Nat. Mach. Intell.* **3** (1), 87–96.

PARK, J. & CHOI, H. 2021 Toward neural-network-based large eddy simulation: application to turbulent channel flow. *J. Fluid Mech.* **914**, A16.

PASCANU, R., MIKOLOV, T. & BENGIO, Y. 2013 On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, Proceedings of Machine Learning Research*, vol. 28, pp. 1310–1318. PMLR.

POPE, S.B. 2000 *Turbulent Flows*. Cambridge University Press.

POPE, S.B. 2004 Ten questions concerning the large-eddy simulation of turbulent flows. *New J. Phys.* **6** (1), 35.

ROGERS, M.M. & MOSER, R.D. 1994 Direct simulation of a self-similar turbulent mixing layer. *Phys. Fluids* **6** (2), 903–923.

SAN, O. 2014 A dynamic eddy-viscosity closure model for large eddy simulations of two-dimensional decaying turbulence. *Intl J. Comput. Fluid Dyn.* **28** (6-10), 363–382.

SAN, O. & STAPLES, A.E. 2012 High-order methods for decaying two-dimensional homogeneous isotropic turbulence. *Comput. Fluids* **63**, 105–127.

SANCHEZ-GONZALEZ, A., GODWIN, J., PFAFF, T., YING, R., LESKOVEC, J. & BATTAGLIA, P. 2020 Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research, vol. 119, pp. 8459–8468. PMLR.

SARGHINI, F., DE FELICE, G. & SANTINI, S. 2003 Neural networks based subgrid scale modeling in large eddy simulations. *Comput. Fluids* **32** (1), 97–108.

SIRIGNANO, J., MACART, J.F. & FREUND, J.B. 2020 Dpm: A deep learning pde augmentation method with application to large-eddy simulation. *J. Comput. Phys.* **423**, 109811.

SLOTNICK, J.P., KHODADOUST, A., ALONSO, J., DARMOFAL, D., GROPP, W., LURIE, E. & MAVRIPLIS, D.J. 2014 CFD vision 2030 study: a path to revolutionary computational aerosciences. *NASA Tech. Rep.* NASA/CR-2014-218178.

SMITH, L.M., CHASNOV, J.R. & WALEFFE, F. 1996 Crossover from two- to three-dimensional turbulence. *Phys. Rev. Lett.* **77**, 2467–2470.

STACHENFELD, K., FIELDING, D.B., KOCHKOV, D., CRANMER, M., PFAFF, T., GODWIN, J., CUI, C., HO, S., BATTAGLIA, P. & SANCHEZ-GONZALEZ, A. 2021 Learned coarse models for efficient turbulence simulation. Preprint, arXiv:2112.15275.

THUEREY, N., HOLL, P., MUELLER, M., SCHNELL, P., TROST, F. & UM, K. 2021 Physics-based deep learning. Preprint, arXiv:2109.05237.

THUEREY, N., WEISSENOW, K., PRANTL, L. & HU, X. 2020 Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA J.* **58** (1), 25–36.

Um, K., Brand, R., Fei, Y., Holl, P. & Thuerey, N. 2020 Solver-in-the-loop: learning from differentiable physics to interact with iterative pde-solvers. In *Advances in Neural Information Processing Systems*, pp. 6111–6122. Curran Associates, Inc.

Ummenhofer, B., Prantl, L., Thuerey, N. & Koltun, V. 2019 Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1lDoJSYDH.

Xie, C., Li, K., Ma, C. & Wang, J. 2019 Modeling subgrid-scale force and divergence of heat flux of compressible isotropic turbulence by artificial neural network. *Phys. Rev. Fluids* **4** (10), 104605.

Xie, C., Wang, J., Li, H., Wan, M. & Chen, S. 2020 Spatially multi-scale artificial neural network model for large eddy simulation of compressible isotropic turbulence. *AIP Adv.* **10** (1), 015044.

Yang, Z. 2015 Large-eddy simulation: past, present and the future. *Chin. J. Aeronaut.* **28** (1), 11–24.

Yu, X., Fernando, B., Hartley, R. & Porikli, F. 2018 Super-resolving very low-resolution face images with supplementary attributes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 908–917. Computer Vision Foundation.