

Spatio-temporal deep learning models of 3D turbulence with physics informed diagnostics

Arvind T. Mohan , Dima Tretiak , Misha Chertkov & Daniel Livescu

To cite this article: Arvind T. Mohan , Dima Tretiak , Misha Chertkov & Daniel Livescu (2020) Spatio-temporal deep learning models of 3D turbulence with physics informed diagnostics, *Journal of Turbulence*, 21:9-10, 484-524, DOI: [10.1080/14685248.2020.1832230](https://doi.org/10.1080/14685248.2020.1832230)

To link to this article: <https://doi.org/10.1080/14685248.2020.1832230>



Published online: 15 Oct 2020.



Submit your article to this journal



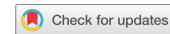
Article views: 75



View related articles



View Crossmark data



Spatio-temporal deep learning models of 3D turbulence with physics informed diagnostics

Arvind T. Mohan ^{a,d}, Dima Tretiak  ^{b,d}, Misha Chertkov  ^c and Daniel Livescu  ^d

^aCenter for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, NM, USA; ^bDepartment of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, USA; ^cProgram in Applied Mathematics, University of Arizona, Tucson, AZ, USA; ^dComputational Physics and Methods Group, Los Alamos National Laboratory, Los Alamos, NM, USA

ABSTRACT

Direct Numerical Simulations (DNSs) of high Reynolds number turbulent flows, encountered in engineering, earth sciences, and astrophysics, are not tractable because of the curse of dimensionality associated with the number of degrees of freedom required to resolve all the dynamically significant spatio-temporal scales. Designing efficient and accurate Machine Learning (ML)-based reduced models of fluid turbulence has emerged recently as a promising approach to overcoming the curse of dimensionality challenge. However, to make the ML approaches reliable one needs to test their efficiency and accuracy, which is recognised as important but so far incomplete task. Aiming to improve this missing component of the promising approach, we design and evaluate two reduced models of 3D homogeneous isotropic turbulence and scalar turbulence based on state-of-the-art ML algorithms of the Deep Learning (DL) type: Convolutional Generative Adversarial Network (C-GAN) and Compressed Convolutional Long-Short-Term-Memory (CC-LSTM) Network. Quality and computational efficiency of the emulated velocity and scalar distributions is juxtaposed to the ground-truth DNS via physics-rich statistical tests. The reported results allow to uncover and classify weak and strong aspects of C-GAN and CC-LSTM. The reported results, as well as the physics-informed methodology developed to test the ML-based solutions, are expected to play a significant role in the future for making the DL schemes trustworthy through injecting and controlling missing physical information in computationally tractable ways.

ARTICLE HISTORY

Received 2 March 2020
Accepted 15 September 2020

KEYWORDS

3D turbulence; deep learning; neural networks; convolutional LSTM; autoencoders; generative adversarial networks

1. Introduction

Several research problems in seemingly disparate fields such as socio-economics, infrastructure networks, physical and natural sciences etc., have a common thread. The data from these systems consist of multiple features varying in both space and time exhibiting

strong *spatio-temporal* dynamics. In addition, many of these systems are high dimensional with millions or billions of degrees of freedom, making them exceptionally complex to study theoretically by means of mathematical and statistical analysis. Such systems are often modelled through numerical computations producing vast amounts of data. However, many practical high dimensional cases arising in engineering, earth sciences and climate modelling, make reliable numerical computations virtually impossible because of the sheer amount of the spatio-temporal resolution required to simulate the governing fluid mechanics equations with high fidelity. One naturally asks if data science approaches, improved dramatically in recent years, can help to resolve the challenge.

Deep Learning (DL), and specifically Deep Neural Networks (NNs), have established themselves as the state-of-the art for data driven models, with successes in myriad applications. Not surprisingly, there has also been a surge of interest in DL applications to fluid mechanics, specifically to computational fluid dynamics (CFD) of turbulent flows. Several recent advancements in applications of DL and classical machine learning techniques to CFD have focused on improving Reynolds Averaged Navier–Stokes (RANS) and Large Eddy Simulation (LES) techniques. In these approaches, the turbulent scales are intelligently parameterised for a class of flows through learning from the ground truth provided by Direct Numerical Simulations (DNS). Some of these approaches have augmented existing turbulence models with traditional ML approaches [1–4] while others have utilised NNs to learn Reynolds-stress closures [5,6], thereby reducing computational costs of RANS/LES and increasing accuracy.

While we acknowledge that physics parameterisation of turbulence is a valuable body of work in itself, we also remark that there are many applications of interest sensitive to accurate resolution of boundary/initial conditions as in [7,8] and/or generating complex synthetic flows [9,10], that require much deeper insight into modelling underlying spatio-temporal phenomena. Efforts in these areas are focused on constructing physics-trustworthy and implementation efficient ROMs. The challenge then boils down to learning the *behaviour* of the underlying dynamical system (turbulence) in an ROM which can then be used to generate spatio-temporal samples which are consistent with spatio-temporal correlations expected in turbulence. Traditional approaches to resolving the challenges rely on computationally efficient projection methods of the Galerkin type – see, e.g. [11–14]. Other methods, such as based on sparse coding [15,16], cluster expansion [17] and also networks of vortices [18], were also utilised to construct ROMs.

Simultaneously, several advances in the recent decade have occurred in using DL, which have made spectacular progress in extracting valuable patterns from large datasets. Most of these successes have been in areas of image classification (i.e. spatial complexity) and language modelling (sequential complexity), which are associated with the focused problems of high-priority in the information industry. As a result, DL for modelling *spatio-temporal complexity* in high dimensions has not yet progressed at the same rate. However, ubiquity of the complex spatio-temporal structures in advanced technological applications has caught attention of the DL community in recent years, and significant advances have been made in developing generative models [19,20] for image generation and video classification/generation. Much of this interest had originated from the computer graphics and animation community, with several recent works focusing on realistic fluid animations [21], splash animation [22], and droplet dynamics [23]. These recent efforts have demonstrated that DL methods, such as Generative Adversarial Networks (GANs), have

a tremendous potential to handle large and spatio-temporally complex datasets. However, these impressive recent results – primarily from the animation and graphics communities – remain rudimentary in exploring and understanding the underlying multi-scale physical phenomena. In a parallel development, driven largely by the dynamical-systems oriented physics community, interesting advances were made in reservoir computing [24–26], allowing, in particular, an advanced modelling of relatively simple but chaotic systems, such as governed by one-dimensional Kuramoto–Sivashinsky equation [27]. However, this line of work has not yet progressed sufficiently far to describe truly multi-scale complex systems. In particular, we are yet to properly understand capability of DL methods to represent complex turbulence phenomena, and specifically for the methods based on GANS architectures [28–31].

This manuscript advances the cause and focuses on exploring capability of various DL algorithms to model the dynamics of turbulence. Specifically, we focus on analysis of algorithms/methods which are split into the following two categories – *Dynamic-map* and *Static-map*. *Dynamic-map* methods model the dynamics of the flow in time, whereas *Static-map* methods model samples of the flow dynamics, without accounting for any temporal dynamics that might be present. Essentially, *Dynamic-map* methods are formulated as an input-output (supervised) learning problem, such that given a sequence of flow realisations, NN is tasked to predict realisations at the subsequent time instants. We adapt algorithms from the DL literature which include both dynamic and static maps, for the aforementioned turbulence application. The core focus of this work is on physically sound analysis of predictions provided by the NNs. We hope that the insights provided by this analysis will help to demystify reported successes of these black boxes and pave a road for their further use as ‘gray’ (if not completely transparent) boxes for multi-scale and physics-rich applications, in particular by incorporating more physics into the NN design.

The remainder of the manuscript is organised as follows. Section 2 outlines the static and dynamic map neural network architectures studied in this work. In Section 3, we present the DNS dataset which is utilised as ground truth throughout this work. Section 4 outlines the turbulence diagnostic metrics we employ to assess the validity of the machine learning models. The results for the static-map architecture are presented in Section 5. Our approach for the dynamic-map architecture via dimensionality reduction is presented in Section 2.2, followed by results in Section 7. Finally, we discuss our findings and scope for future efforts in Section 8.

2. Deep learning algorithms

In this section, we describe two DL approaches discussed in the manuscript: *Static-map* and *Dynamic-map*.

2.1. Static-map: generative adversarial networks (GANs)

Generative Adversarial Networks (GANs), proposed by Goodfellow [19,32], are built from two NNs, called generator and discriminator, respectively. In this architecture, the layers within the generator typically consist of transpose convolution, batch normalisation, and ReLU activation, respectively. The discriminator mirrors the generator’s architecture by using standard convolutional layers, and notably, contains no fully connected layers.

The discriminator's final activation function is sigmoid; therefore, its output is a probability of the sample being real or fake. In summary, the generator up-samples a latent vector z to generate snapshots while the discriminator down-samples to classify. Batch normalisation modules are present in both networks and address the issue of changing data distributions between layers in the models. Ioffe and Szegedy [33] named this issue internal covariate shift and addressed it in detail in their paper. Our architecture differs from this vanilla GANs, by employing convolutional layers in GANs (CGANs) to account for the high dimensional data, and several modifications to improve training stability and performance, as detailed below.

Figure 1 illustrates the CGAN architecture, including the input and output sizes. There are two phases to a training cycle, one for each network. First, the discriminator is trained to differentiate between real samples and generated samples. The real images are labelled as a 0 and the generated images as a 1. Predicted labels are compared to the target labels, and then the loss gradients are propagated through the network. The generator is trained using the following dynamic loss function.

$$\text{Loss}_G = \text{BCE}(D(G(z)), 0)$$

We take the Binary Cross Entropy between the discriminator's label of the generated snapshots, notated as $D(G(z))$, and the target label. The target for the generator's loss function is 0, i.e. it tries to produce samples indistinguishable from the real samples, from the perspective of the discriminator. While one network trains, the other's weights are frozen and are not updated. As the two networks train together, the gradients from the discriminator allows the generator to learn the distribution of the training data as it tries to replicate it. Another key consideration when training CGANs is the balance between the discriminator and the generator. In general, the discriminator should perform better than the generator and correctly identify whether the samples it receives are fake or real. If the discriminator is too weak, it will not be able to process the details of each sample and differentiate between DNS and generated data. However, if the discriminator is far stronger than the generator, it no longer provides meaningful gradients to the generator,

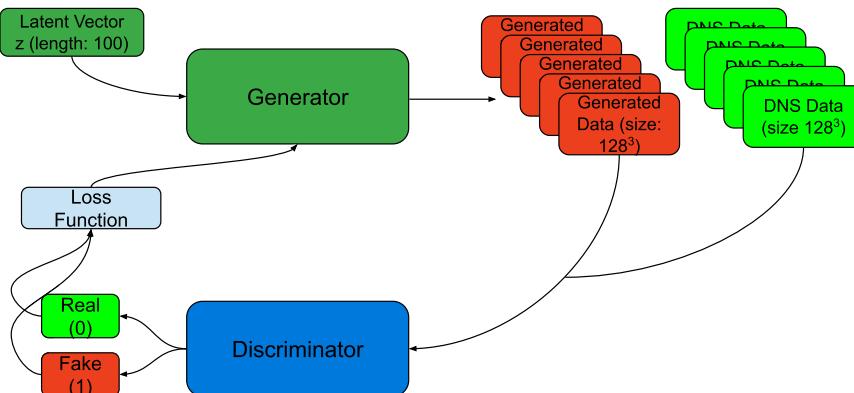


Figure 1. Schematic of generative adversarial networks (GANs) with convolutional generators and discriminators for 3D turbulence.

preventing further training. We combat this through a combination of label smoothing, architecture changes, variable learning rates, and variable optimisers over the vanilla GANs architecture.

The CGANs employed in this work consist of an eight layer discriminator and a five layer generator. The generator takes a uniform vector z as an input and produces a cubic snapshot (of the same dimensions as the input) as an output. For a discussion of sampling z , please see Appendix A.4. The discriminator and generator do not have the mirrored structure typically found in vanilla GANs literature. Instead, we found that adding further depth to the discriminator allows it to better discern between the generated data and the real data. Essentially, our deeper discriminator serves as a thorough accuracy check. A larger kernel size (7^3) was found to perform well, and we observe that the kernel size is especially important in the generator's transpose convolutional layers and is less important in the discriminator, for the accuracy of the predictions. Intuitively, we can see that the generator performs regression compared to the discriminator, which has a possibly less challenging objective of binary classification. For this reason, we decided to break convention in designing the CGAN architecture, with non-symmetrical generator and discriminator networks. We did not use the loss functions for either network as a metric to determine training progress. Instead, we used the physical diagnostics detailed in Section 4. We determined that the model converged when our diagnostics stopped improving. Furthermore, we noticed that the trained discriminator became a useful tool to sort the generated snapshots by those which the discriminator labelled real, and therefore fared better on our diagnostic tests.

An important issue observed during training was the tendency of the network to 'memorise' a subset of samples rather than learning the entire data distribution. This is known as mode collapse [34,35]. An analogous, illustrative example of the same problem with the popular MNIST [36] dataset would be if the CGANs were only able to reproduce the number 4 and nothing else. This occurs when the generator reproduces a sample which 'fools' the discriminator. After doing so, it learns to continue reproducing similar samples until it converges on what it presumes is an optimal output which – in reality – is a collapsed output containing only a small set of classes. This subset of classes is generally determined by the initialised weights of both networks. Since the output minimises the loss function, there is nothing to push the generator into creating any other sample. As this happens, changing the latent vector z no longer induces a change in the output image y . In the case of our CGANs, mode collapse was readily apparent every time it occurred: the discriminator's loss quickly approached 0 while the generator's loss exploded. Furthermore, different snapshots were visually indistinguishable when cut into slices and shown as an image (accomplished through a method similar to Figure A4). To rectify this issue, we included multiple dropout layers in both networks. By using random dropout to nullify some of the networks' nodes, we force the generator into creating different types of samples by introducing extra noise into the network [35]. Hence, even after training, the same input z will still produce a different outputs y due to the dropout layers present in the generator network. This prevents it from converging to a single sample subset. While mode collapse happens to be one of the more important aspects of training CGANs, there are other practical considerations crucial to training CGANs for complex datasets such as turbulence, and these are outlined in Appendix A.2.



2.2. Dimensionality reduction of large datasets with convolutional autoencoder neural networks

The major challenge of data-driven modelling of large complex systems is that time varying dynamics are fundamentally high dimensional in nature. Over the years, several strong arguments have been made that in spite of its high dimensional nature, the practically relevant large scale dynamics of many systems of interest are typically low dimensional [37]. Thereby, it is argued that one can study the system reliably by modelling its low dimensional representation (LDR), while ignoring other features.

Another important idea in dynamical systems theory is that the spatio-temporal realisations of the system state contain information about the LDR, in form of its *observables* [38–40]. Therefore, several studies have focused on estimating/approximating the LDR, directly from observations of the actual system. This is a popular strategy since there are several cases where it is difficult to analytically derive a model for the LDR from the governing equations. Common examples are turbulence (due to the complexity of the Navier–Stokes operator) and various earth sciences problems where a theoretical description of the system is itself an area of active research.

The LDR is often only the first step in building ROMs for system modelling, with the next step being to model the *temporal evolution* of the LDR dynamics. For turbulent flows, a popular strategy is to compute the LDR with Proper Orthogonal Decomposition (POD) of the flow, whose modes contain dominant dynamics in a smaller, low-dimensional subspace compared to that of the entire flow. These dominant modes are then evolved via Galerkin projection [12], which projects the modal dynamics on the Navier–Stokes equations, with the goal of approximating the evolution of the flow's intrinsic low dimensional attractor. A more recent innovation has been to utilise Koopman operator theory to model the LDR by directly learning the eigenpairs of the system [41,42]. However, Galerkin projection based approaches require that the projected dynamics be analytically represented and maintaining temporal stability is a topic of research [43]. Deep learning approaches demonstrated in Ref. [44] use POD modes that were evolved with LSTM neural networks instead of Galerkin projection. The results showed promise in the ability of LSTM networks to capture non-linear, non-stationary dynamics in temporal evolution. However, much like the POD-Galerkin approach, the efforts in [44] did not account for variations in the spatial POD modes of the LDR, and hence were limited in application. The CC-LSTM deep learning architecture proposed in the present work significantly extends that capability to include 3D spatio-temporal dynamics in a compute efficient manner, thereby opening up the idea to larger datasets.

As mentioned previously, we construct an LDR with a *Convolutional Autoencoder* NN that has been increasingly popular in the deep learning community [45]. A Convolutional Autoencoder (CAE) consists of multi-layered deep CNNs, which utilise the convolutional operators to successively reduce the dimensionality of the data. The CAE learns compressed, low dimensional '*latent space*' representations for each snapshot of the flow. The CAE has two main components – the *encoder* and the *decoder*. The representational information to transform the snapshot from its original high-dimensional state to the latent space is stored in the encoder. Similarly, the reconstruction from the latent to original state is learned by the decoder. Both the encoder and decoder are tensors which are learned by standard neural network backpropagation and optimisation techniques. It is important to

note that this is a *convolutional* autoencoder, such that the spatial information is learned by translating filters throughout the domain, as in a convolutional neural network. These convolving filters capture various spatial correlations and drastically reduce the number of weights we need to learn due to parameter-sharing [46]. This makes the training considerably cost effective and faster than using a standard fully-connected autoencoder. The reader is referred to Ref. [46] for more details.

2.3. Dynamic-map: compressed convolutional LSTM (CC-LSTM)

2.3.1. Convolutional LSTM: potential and challenges

Since turbulence datasets exhibit strong spatio-temporal dynamics, *dynamic-map* networks can be a viable choice to learn these variations. The Convolutional Neural Network (CNN) architecture is ideal for learning patterns in spatial datasets, like images or volumetric datasets [47]. More details on the CNN architecture can be found in Appendix A.1. On the other hand, Long Short Term Memory (LSTM) NNs have been found to be powerful for sequence modelling, in applications ranging from language translation [48] to financial forecasting applications [49]. The details of the LSTM architecture are presented in Appendix A.2. In a complementary fashion, vanilla LSTMs are generally restricted to one-dimensional datasets and not cases where the data also exhibits spatial dynamics in addition to temporal. In this architecture, an LSTM cell consists of input and hidden states that are one-dimensional vectors. Therefore a two- or three-dimensional input (such as an image or a volumetric data field) has to be resized to a single dimension. The ‘removal’ of this dimensional information fails to capture spatial correlations that may exist in such data, leading to increased prediction errors, as reported by Xinjian [20].

While deep learning literature on addressing this dual spatial/temporal modelling challenge is scarce, a notable algorithm by Xinjian [20] is the Convolutional LSTM (ConvLSTM). ConvLSTM consists of a simple but powerful idea – to embed Convolutional kernels (used in CNNs) in a LSTM to learn both spatial and sequential dynamics *simultaneously*. As a direct consequence of this embedding, the LSTM cell can now process hidden and input states in higher dimensions, as opposed to strictly one-dimensional sequences in traditional LSTM. With this abstraction, the same equations for LSTM in Appendix A.2 can be used for ConvLSTM cell, with the only difference being that the input vector and the cell gates *have the same dimensionality*. This enables us to provide a 2D/3D input and obtain 2D/3D vectors C_t and h_t as outputs from the ConvLSTM cell, thereby retaining spatial information in the data. ConvLSTM has been successfully demonstrated for several sequential image prediction/classification tasks [50–52].

In spite of its strengths, a major limitation of using ConvLSTM for large 2D and 3D datasets has been its huge memory cost. The primary reason is the complexity of embedding a convolutional kernel in an LSTM and unrolling the network, which drastically increases the number of trainable parameters for even moderate sized datasets. Consequently, existing literature on ConvLSTM has primarily focused on 2D datasets, instead of 3D and higher dimensional datasets, which are ubiquitous in scientific problems. As a result, there is a clear need to adapt and rigorously evaluate ConvLSTMs for high dimensional datasets like those encountered in turbulent flows and compare the results with popular methods like GANs. This is the focus of this paper.

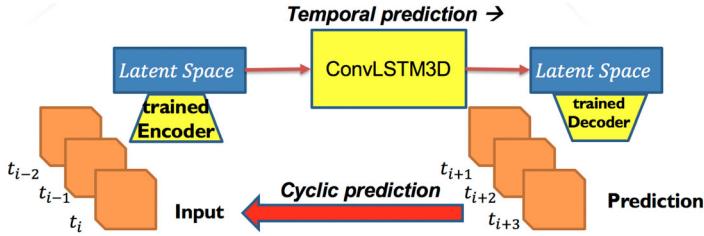


Figure 2. Schematic of the compressed ConvLSTM (CC-LSTM) architecture with pre-trained convolutional autoencoder layers for dimensionality reduction of spatio-temporal 3D flow dataset.

2.3.2. Compressed convolutional LSTMs

In order to reduce the computational/memory costs while also leveraging the strengths of ConvLSTM, we propose a modified architecture where the high dimensional flow snapshot (i.e. at any given time instant) is first ‘compressed’ to a low dimensional *latent space*, which is then used as a training data for the ConvLSTM. The trained ConvLSTM predicts future instances of the flow also in latent space, which is subsequently ‘decompressed’ to recover the original dimensions of the flow. This compression and decompression are accomplished using a Convolutional Autoencoder neural network (CAE), and we call the combined architecture of CAE + ConvLSTM as *Compressed Convolutional LSTM* (CC-LSTM). This approach makes the ConvLSTM approach more computationally tractable. A schematic detailing this architecture is shown in Figure 2. Further information about CAE is presented in Section 2.2.

3. Dataset

The dataset consists of a 3D Direct Numerical Simulation (DNS) of homogeneous, isotropic turbulence with passive scalars advected with the flow, in a box of size 128^3 . Two passive scalars with different Probability Density Functions (PDF) are considered here in order to provide more complexity to the test cases, as explained below. We denote this dataset as *ScalarHIT* for the remainder of this work. We provide a brief overview of the simulation and its physics in this section. See [53] for details. The ScalarHIT dataset is obtained using the pseudo-spectral version of the CFDNS code, as described in [53]. We solve the incompressible Navier–Stokes equations:

$$\partial_{x_i} v_i = 0, \quad \partial_t v_i + v_j \partial_{x_j} v_i = -\frac{1}{\rho} \partial_{x_i} p + \nu \Delta v_i + f_i^\nu, \quad (1)$$

where f^ν is a low band forcing, restricted to small wavenumbers $k < 1.5$. The 128^3 pseudo-spectral simulations are dealiased using a combination of phase-shifting and truncation to achieve a maximum resolved wave-number of $k_{\max} = \sqrt{2}/3 \times 128 \sim 60$. Spectral resolution used is $\eta k_{\max} \sim 1.5$ (Figure 3).

The scalar field ϕ evolves according to

$$\partial_t \phi + v_j \partial_{x_j} \phi = \mathcal{D} \Delta \phi + f^\phi, \quad (2)$$

where the form of f^ϕ is designed such that the scalar PDF at stationarity can be controlled. ν and \mathcal{D} in Equations (1)–(2) are viscosity and diffusion coefficients respectively.

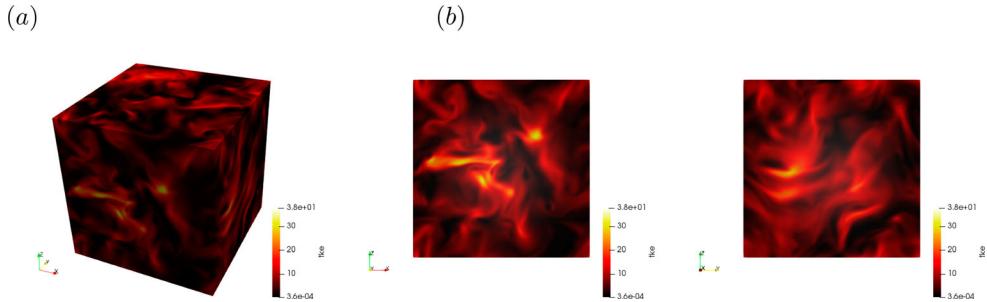


Figure 3. Instantaneous turbulent kinetic energy from the homogeneous isotropic turbulence with passive scalars (ScalarHIT) dataset: (a) 3D view and (b) cross-sectional views.

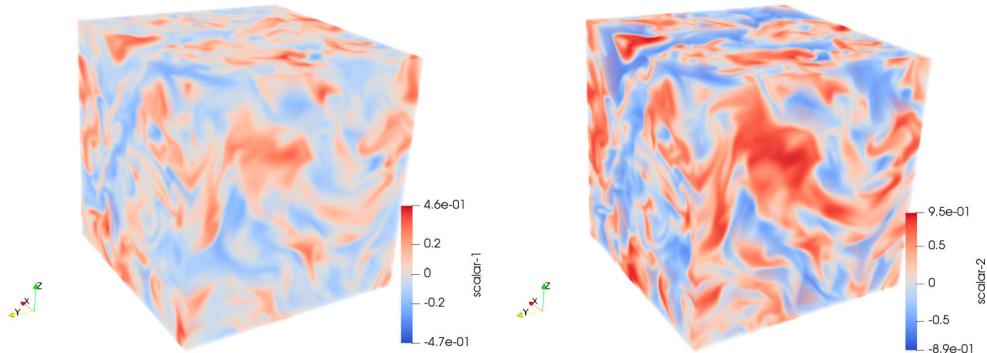


Figure 4. 3D snapshots of the two passive scalar fields, with quasi-Gaussian (left) and flat (right) PDFs [53].

Two relevant parameters of the flow are the Schmidt number (ν/\mathcal{D}) and Reynolds number (Re). Simulations considered here are performed for a constant $Sc = 1$. In homogeneous isotropic turbulence, it is standard to associate Re with the Taylor microscale, as

$$Re_\lambda = \sqrt{\frac{20}{3} \frac{\text{TKE}^2}{\nu \epsilon}}, \quad (3)$$

where TKE is the turbulent kinetic energy (Figure 4).

In this work, we use the novel scalar forcing approach based on a chemical reaction analogy (RA), proposed in [53]. This method can produce more general scalar PDFs, for instance quasi-double- δ PDF, compared to forcing methods that are limited to producing Gaussian or near-Gaussian scalar PDFs. It also ensures the boundedness of the scalar field, in contrast to previous methods that can violate naturally existing bounds. For completeness, here we briefly describe the method and refer the reader to Ref. [53] for details. The RA method uses a hypothetical chemical reaction to convert the mixed fluid back into unmixed pure states. Reactants are identified based on a RA similar to that proposed in [54] to quantify the width of the Rayleigh–Taylor mixing layer and further generalised in [55]. Thus any partially mixed fluid state can be considered as being composed of fully mixed fluid, M, where the scalar has the value of its average, and excess pure fluid, E, i.e.

fluid where the scalar has the value of one of its bounds. Using standard reaction kinetics formulas between M and E, Ref. [53] arrived at a formula for the forcing term, f^ϕ in Equation (2). If the scalar bounds are $\phi_l = -1$ and $\phi_u = +1$, then f^ϕ can be written in a compact form as

$$f^\phi = \text{sign}(\phi) f_c |\phi|^n (1 - |\phi|)^m \quad (4)$$

where m, n are the stoichiometric coefficients and f_c , which is related to the reaction rate constant, defines the strength of the forcing (Figure 4). All three parameters influence the shape of the scalar PDF at stationarity.

The forcing terms ensure that velocity and scalar fields attain stationary states. The level of turbulence attained in the simulation translates to $Re_\lambda \sim 91$ in the statistically steady regime. The scalar forcing parameters are chosen such that scalar ϕ_1 exhibits quasi-Gaussian characteristics with kurtosis value of approximately 3, while scalar ϕ_2 has a much lower kurtosis value of 2.2. In both cases, $m = n = 1$, but f_c has different values. We expect that the two NNs considered here would be able to capture the quasi-Gaussian scalar PDF. The ability to capture the scalar bounds is a novel test for both the static and dynamics maps.

Both networks studied in this work are trained on the ScalarHIT dataset. A static-map network is agnostic to the sequential order in the snapshots and only seeks to learn the statistics of the flow in individual snapshots. We use DNS training snapshots from $\tau = 0 - 3$. Here, τ is the normalised large eddy turnover time, corresponding to a single cycle in the statistically stationary flow. The test data to validate the trained model predictions consists of snapshots from $\tau = 3 - 4.5$. Dynamic-map networks can also use the same train/test data split as above. However, since the model aims to capture the temporal dynamics of the flow, the sequential information in the train/test data is retained throughout training.

4. Diagnostic tests for turbulence

In this section, we review basic statistical concepts commonly used in the modern literature to analyse results of theoretical, computational and experimental studies of homogeneous isotropic incompressible turbulence in three dimensions. Combination of these concepts are used in the main part of the manuscript as a metric to juxtapose results of the two (static-map and dynamic-map) DL methods.

We assume that a $3d$ snapshot, or its $2d$ slice, or a temporal sequence of snapshots of the velocity field, $\mathbf{v} = (\mathbf{v}^{(i)}(\mathbf{r}) | i = 1, 2, 3)$, is investigated. Here, we focus on analyses of static correlations within the snapshots. The remainder of this section contains classical material described in many books on the theory of turbulence (see, e.g. [56]). We describe the main turbulence concepts mentioned in the results section one by one, starting from simpler ones and advancing towards more complex concepts. A key expectation from any generative machine learning models would be the ability to predict non-Gaussian statistics.

4.1. 4/5 Kolmogorov law and the energy spectra

A main statement of the Kolmogorov theory of turbulence is that asymptotically in the inertial range, i.e. at $L \gg r \gg \eta$, where L is the largest (so-called energy-containing) scale of turbulence and η is the smallest (so-called Kolmogorov or viscous) scale of turbulence,

the statistics of motion have a universal form that is uniquely dependent on the kinetic energy dissipation, $\varepsilon = \nu \langle (\nabla^{(i)} v^{(j)}) (\nabla^{(i)} v^{(j)}) \rangle / 2$, and does not depend on viscosity, ν . A consequence of the existence of the inertial range is that, within this range, the transfer term,

$$F(r) \doteq \langle v^{(j)}(\mathbf{0}) v^{(i)}(\mathbf{r}) \nabla^{(i)} v^{(j)}(\mathbf{r}) \rangle,$$

does not depend on r . Moreover, (in fact, the only formally proven statement of the theory) the so-called 4/5-law states that for the third-order moment of the longitudinal velocity increment, $S_3^{(i,j,k)}(\mathbf{r}) \doteq \langle (v^{(i)}(\mathbf{r}) - v^{(i)}(\mathbf{0})) (v^{(j)}(\mathbf{r}) - v^{(j)}(\mathbf{0})) (v^{(k)}(\mathbf{r}) - v^{(k)}(\mathbf{0})) \rangle$:

$$L \gg r \gg \eta : \quad S_3^{(i,j,k)} \frac{r^i r^j r^k}{r^3} = -\frac{4}{5} \varepsilon r. \quad (5)$$

The Kolmogorov self-similarity hypothesis applied to the second moment velocity increment results in the expectation that within the inertial range, this scales as $S_2(r) \sim C_2(\varepsilon r)^{2/3}$. This feature is typically tested by plotting the energy spectrum of turbulence in the wave vector domain, where it is restated as a $-5/3$ power law dependence of the energy spectrum with respect to the wavenumber, and will be addressed in the forthcoming sections.

4.2. PDF of longitudinal velocity gradient

Consistently with Equation (5), the estimation of the moments of order n of the longitudinal velocity gradient results in

$$D_n \doteq \left\langle \left| (\nabla^{(i)} v^{(j)}) (\nabla^{(i)} v^{(j)}) \right|^{n/2} \right\rangle \sim \frac{S_n(\eta)}{\eta^n}, \quad (6)$$

where $S_n(r) \doteq \langle \prod_{i=1}^n (v^{(i)}(\mathbf{r}) - v^{(i)}(\mathbf{0})) \mathbf{r}^{(i)} / |\mathbf{r}^{(i)}| \rangle$. Intermittency (extreme non-Gaussianity) of turbulence is stronger expressed at larger n in Equation (6).

4.3. Statistics of coarse-grained velocity gradients: Q–R plane.

The properties of the velocity gradient tensor are related to a wide variety of turbulence characteristics, such as the flow topology, deformation of material volume, energy cascade, and intermittency. One of the hallmarks of 3D turbulence is the tear-drop shape of the joint-PDF of the second (usually denoted by Q) and third (usually denoted by R) invariants of the velocity gradient tensor. This form can be related to the vortex stretching mechanism and shows that certain local flow configurations are preferred in 3D turbulence. A useful extension of this analysis was proposed in [57] to velocity gradient coarse-grained over an inertial-range scale. Following the notations from [57], the coarse-grained velocity gradient tensor \mathbf{M} is constructed by interpolating the velocity at Lagrangian points, i , at the centre of mass of the associated tetrahedron of volume Γ as

$$M_{ab} = (\rho^{-1})_i^a v_i^b - \frac{\delta_{ab}}{3} \text{tr}(\rho_i^{-1} \mathbf{v}_i), \quad (7)$$

where a and b are spatial coordinates and ρ_i^a is the vector resulting from the vertex positions after the elimination of the centre of mass. The invariants Q and R are then defined such that



$Q = -(1/2)tr\mathbf{M}^2$ and $R = -(1/3)tr\mathbf{M}^3$. Note that the trace of \mathbf{M} (i.e. the first invariant) is zero due to incompressibility. Then the $Q-R$ joint-PDF indicates the turbulence structure at scale $r = |\rho|$. Different parts of the $Q-R$ plane are associated with different structures of the flow. Thus lower right corner (negative Q and R), which has higher probability than other regions, corresponds to a pancake type of structure (two expanding directions, one contracting) with the direction of rotation (vorticity) aligned with the second eigenvector of the stress. This tear-drop shape of the probability isoline becomes more prominent with decrease of the coarse-graining scale.

5. Results using convolutional generative adversarial networks (CGANs) for 3D turbulence

The network is trained as described in Section 2.1, with the objective to model the statistics of the ScalarHIT data. We now present the results where we attempt to generate samples of the ScalarHIT flow and compare it with the real flow. It is important to note in the CGAN architecture, the predicted samples are not temporal and they are *static*, i.e. the predictions are not correlated, unlike its exotic variants like RNN-GANs [58]. Figure 5 shows three randomly chosen samples out of the hundreds generated by the CGANs, followed by its average. The diagnostic metrics used are those described in Section 4. The first is the energy spectra, on the top left in Figure 5. We can see that the spectra captured by the CGANs match very closely with the low and mid range wavenumbers, which correspond to large and inertial scales of turbulence. Discrepancies occur at higher wavenumbers in the inertial scales and all the viscous scales. The next metric (in the top right panel of Figure 5) is the probability density function (PDF) of the velocity gradient. The objective is testing how the network captures intermittent events in the flow, which are associated with the tails of the PDF. The intermittent events are seen in strongly non-Gaussian shape of the PDF characterised by extended tails and the CGANs come close to reproducing this trend well, with discrepancies occurring at the tail. This behaviour is seen in all samples that CGANs generate as it has learned the statistics of the stationary flow dataset, and 3 samples are shown in Figure 5 for example. Finally, the most stringent test on the bottom panel is the $Q-R$ joint PDF, since it captures the 3D morphology of the flow. The $Q-R$ joint PDF at $r = 0$ corresponds to small scale behaviour, $r = 8$ for inertial range scales and $r = 32$ for large scale behaviour, as explained in Chertkov et al. [57]. Even though the kernel sizes for all networks in CGANs were ≤ 7 , i.e. significantly larger than the kernels of size 3, we notice that it does not improve large scale resolution. A clearer picture emerges from the $Q-R$ joint PDF, where we notice that CGANs neglect the smaller scales as seen in the energy spectra, while the inertial range scales are modelled reasonably well. Finally, the CGANs seem to model the qualitative statistics of the stretching and compression of the large scale flow morphology, with discrepancies occurring in some of the quadrants. This finding also illustrates the value of $Q-R$ joint PDF in assessing any ML turbulence model, since such subtle deviations in large scale structures are not noticed in the widely used Kolmogorov spectrum and PDFs of velocity gradient magnitude.

We now turn our attention to the two passive scalars ϕ_1 and ϕ_2 which are advected with the velocity field. Figures 6 and 7 compare the CGANs scalar PDFs predictions against the DNS results for ϕ_1 and ϕ_2 , respectively. The passive scalars were introduced with specific, hard physical bounds with ϕ_1 in range $(-0.5, +0.5)$ and ϕ_2 in $(-1.0, +1.0)$;

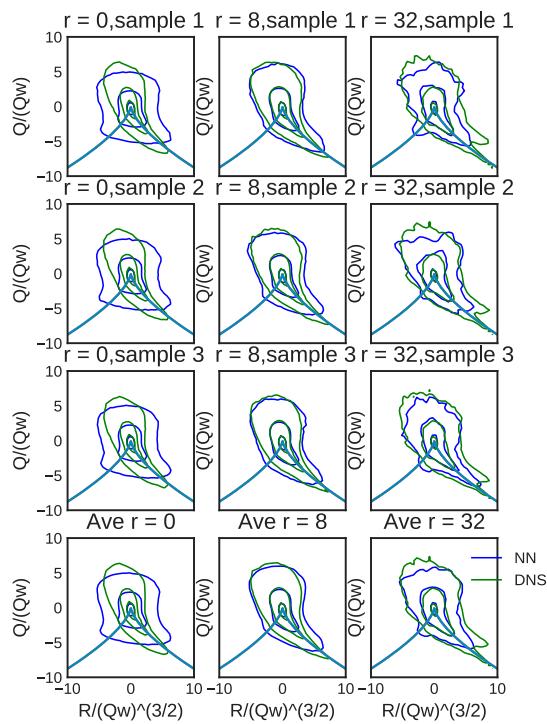
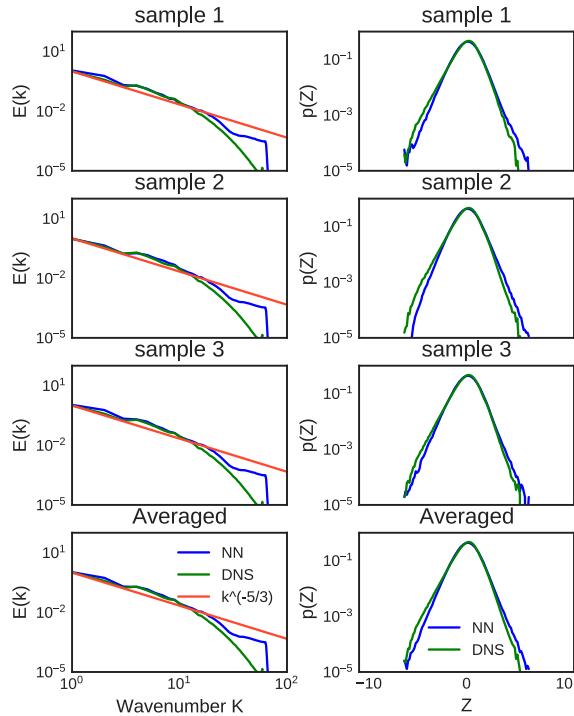


Figure 5. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen static snapshot predictions produced by CGANs

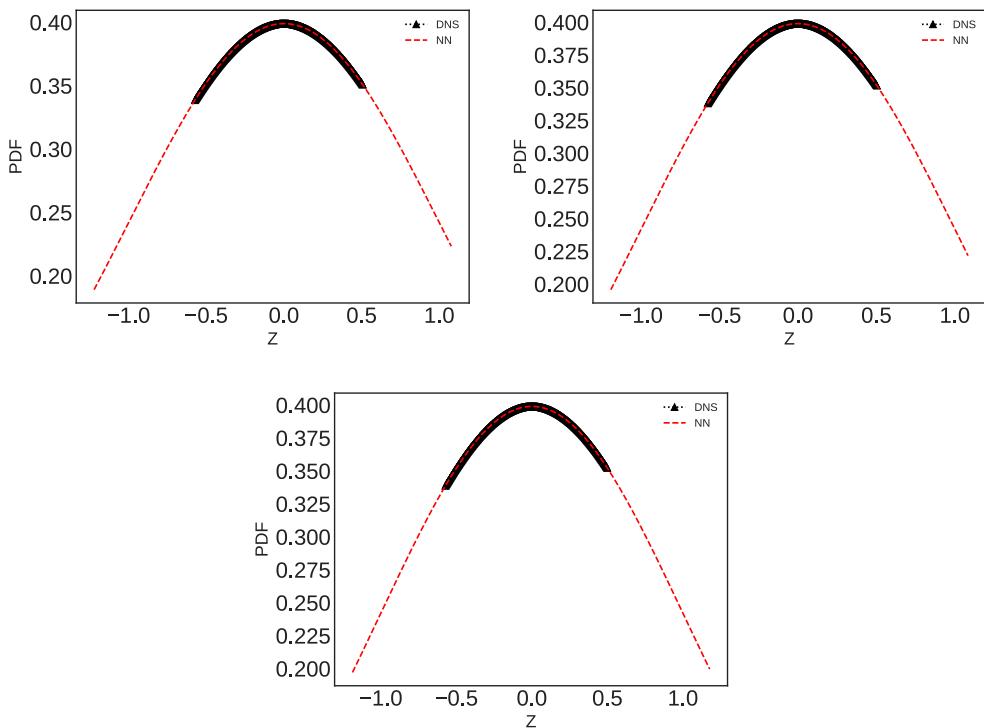


Figure 6. Normalised PDFs of passive scalars ϕ_1 predicted by CGANs and comparison with DNS, where CGANs fail to capture accurate scalar PDF bounds.

which is encountered in many physical scalars (e.g. mass fractions). We see that while CGANs predictions for both scalars appear to capture the PDF profile, they severely overshoot the scalar bounds, with the predicted ϕ_1 and ϕ_2 bounded between $(-1.0, +1.0)$ and $(-2.0, +2.0)$ respectively. Therefore, even though the convolutional generator can sufficiently learn trends of large-scale behaviour in the velocity fields, it appears that it has severe difficulties learning the *advection* quantities by the same velocity fields, especially for highly non-gaussian PDFs (as seen in Figure 7). This points to a topic worthy of further research due to the popularity of GANs in modelling turbulent velocity fields, with passive scalars having not been previously explored.

6. Analysis of 3D turbulence dimensionality reduction with convolutional autoencoders

The schematic of the CAE architecture used for the ScalarHIT dataset is shown in Figure 9. The CAE greatly reduces the memory utilisation since the same n weights in a convolutional kernel are translated throughout the domain of size $m \times m$, where $m \gg n$. These n weights are global, hence learned for all regions of the domain. In contrast, the standard, fully-connected autoencoder architecture would need m^2 weights which are local, leading to prohibitive memory consumption and extremely high training cost. In addition to computational benefits, the design of the convolutional kernel offers flexibility in

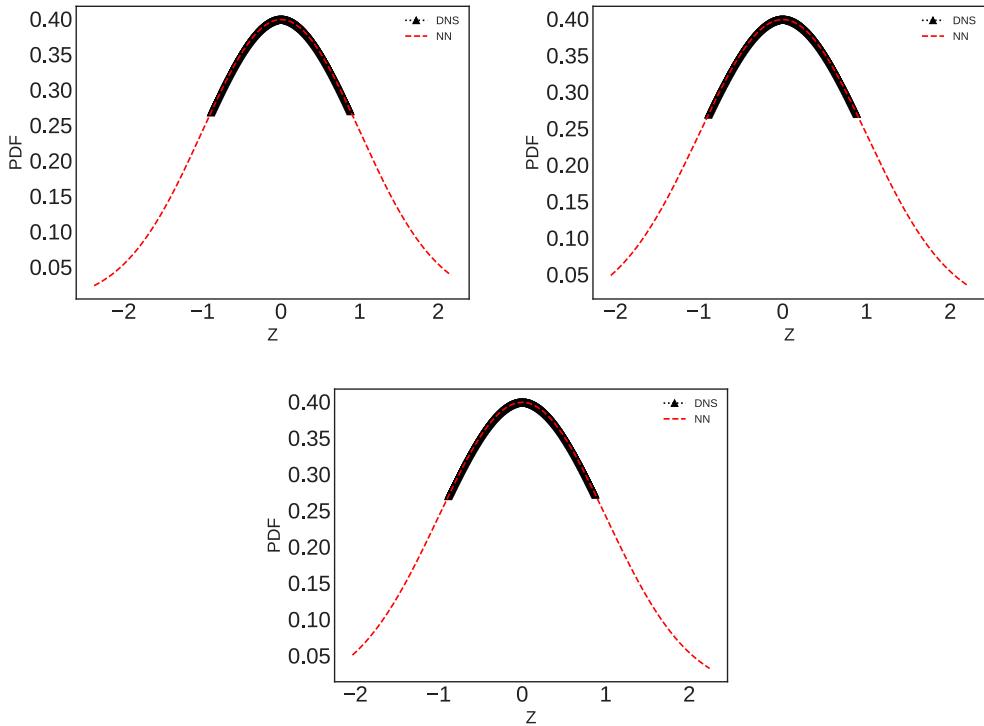


Figure 7. Normalised PDFs of passive scalar ϕ_2 predicted by CGANs and comparison with DNS, where CGANs fail to capture accurate scalar PDF bounds.

tuning the number of shared weights and mode of translation through the domain, as will be explained in this section.

Another important aspect is the number of features, i.e. trainable parameters in the CAE. In the case of ScalarHIT dataset, there are five features corresponding to the three components of velocity and two passive scalars. Increasing the number of features in the latent space allows it to encode more information at a minimal increase in computing cost, while also compressing the high dimensional dataset. We thus define the compression ratio z as

$$z = \frac{(\text{original dimensions} \times \text{number of input features})}{(\text{latent dimensions} \times \text{number of latent features})} \quad (8)$$

From Equation (8), it follows that for input dimensions of size 128^3 with 5 features; and a latent space of dimensions 15^3 with 25 features, there is considerably lesser impact on z with increase in latent space features. As such, the most significant impact comes from the latent space dimensions, giving us the liberty to increase the feature space. In fact, the CAE in this work uses 25 features to obtain a compression ratio of ≈ 125 , i.e. a 125-fold decrease in size for every snapshot of the flow. This leads to tremendous gains in efficiency and makes a ROM computationally efficient, since the original dimensions were prohibitive from a memory standpoint. Mathematically, we can say that the subspace spanned by the input features is mapped by a neural network onto a latent subspace spanned by a different set of learned features. Typically, an increased number of features in the latent space has a direct effect on accuracy of compression, but with a decrease in the compression ratio

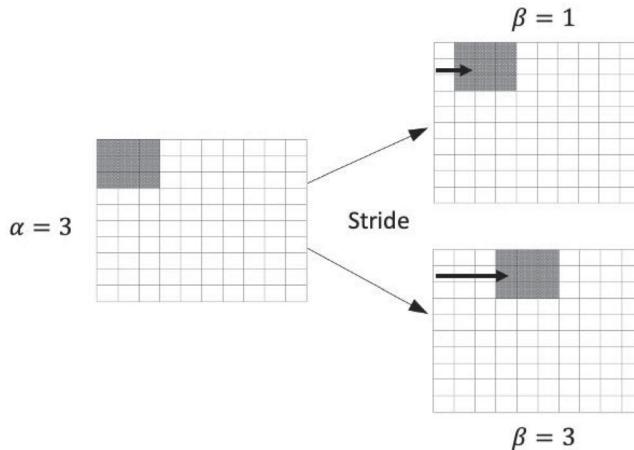


Figure 8. Variable striding in convolutional kernels with kernel size α and stride length β : $\beta = 1$ corresponds to cell by cell striding, while $\beta = 3$ skips over 2 cells for every stride, thereby producing a convolved domain of lower dimension.

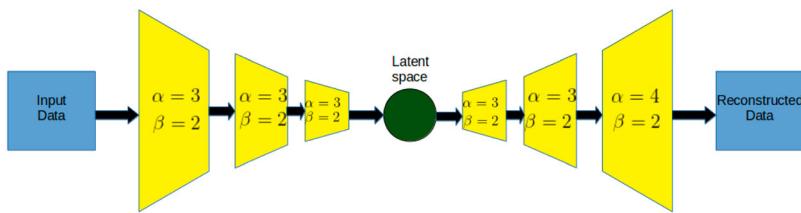


Figure 9. Schematic of a convolutional autoencoder NN Architecture with kernel size α and kernel stride β for dimensionality reduction of input data to latent space, and reconstruction from reduced latent space to original dimensions.

and increase in the computational cost. The optimal number of features is therefore a user choice, based on compute resources and level of compression required. In any CAE, two key design choices have to be made: the *kernel size*, α and *kernel stride*, β . The kernel size indicates the spatial extent of a single kernel. For instance, a kernel size of $3 \times 3 \times 3$ contains 27 shared, trainable weights. The next choice is to decide how the shared weights (i.e. the kernel) translate across the domain. An illustration is shown in Figure 8, where a kernel $\alpha = 3$ can be translated by a distance β of our choosing, known as stride. The figure shows kernel positions after $\beta = 1$ and $\beta = 3$ strides on the domain, and the strides are repeated until the entire domain has been traversed by the kernel. By increasing the stride, the kernel needs fewer convolutions to cover the entire domain and results in much smaller domain, as will be explained in the following section.

At the core of any CNN (and therefore a CAE) is the convolution operation. In the CAE encoder network (i.e. layers to the left of the latent space in the schematics), the kernel convolves with the data to reduce its dimensionality for every time instant t_i . As a result, a $\alpha = 3$ kernel had dimensions $(3 \times 3 \times 3)$ and therefore downsamples a spatial field of size 3^3 – known as the *receptive field* (Ref. [46]) – to a single point. The decoder network kernel (i.e. layers to the right of the latent space in the schematics) then

upsamples each point in the latent space back to the size of the receptive field through a deconvolution operation. Downsampling in the case of NNs can be explained as a weighted averaging operation, where the averaging weights are learned. Similarly, the upsampling kernel weights are also learned to perform the inverse operation. By stacking multiple CNN layers in the encoder, the input is downsampled in every layer and the resulting domain – the latent space – can be extremely low dimensional. Likewise, upsampling can be performed by suitable number of decoding layers to recover the original dimension.

It is important to note that dimensionality reduction to obtain an LDR is accompanied by loss of some information. For instance, popular approaches like POD can represent dominant energetic dynamics in the first few eigenpairs. These eigenpairs can be used for further analysis or modelling tasks, such as Galerkin projection, while the eigenpairs having very low energy contribution to the overall dataset are truncated, thereby leading to information loss. While the CAE is no exception, it distinguishes itself from the POD in two major ways: first, the POD bases compress the dataset as a *linear* map, whereas autoencoders with multiple layers and non-linear activation functions are inherently *non-linear* maps [59]. Consequently, autoencoders can provide very high compression ratios for the same dataset. Second, POD computation results in several global modes with the same dimensionality of the datasets, with ROMs primarily emulating only the temporal coefficients of the modes, i.e. the spatial structures captured by the POD modes are still high dimensional. In contrast, CAE can directly learn local LDRs for each snapshot that have degrees of freedom several orders of magnitude lower than the training dataset. From a computing standpoint, this leads to significant reduction in memory resources and ROMs can now emulate both spatial and temporal dynamics with the low dimensional latent space.

Since it is derived from a CNN, the information content learned by a CAE is dominated by α and β . For a fixed kernel size α , the striding of the convolutional kernel has a direct effect on the dimensionality of the convolved output after each layer. From Figure 8, it is clear that increasing the stride diminishes the coverage of kernel over the domain, making the convolved output sparser. For a fixed $\alpha = 3$, $\beta = 3$ leads to an overlap with receptive field at the previous stride, while $\beta = 3$ removes any overlap. Higher values of β create gaps in the domain which are not seen by the kernel, and hence can traverse the entire domain in fewer steps than using $\beta = 1$. These choices significantly influence the accuracy, degree of compression and computational cost of the ROM. We now present some physical insight about α and β in the next section.

6.1. Physical interpretation of α and β

It is now worthwhile to discuss implications of these choices in dimensionality reduction of complex, spatio-temporal and multiscale datasets like turbulence. From the discussion above, it is apparent that there are two competing strategies for dimensionality reduction in a CAE. The first strategy relies on a large α to increase the receptive field. A larger receptive field would decompose several adjacent data-points into a single data-point. Therefore, for a desired dimension of the latent space, a suitable value of α can be computed. The second strategy is to retain a constant, small α , but increase β to traverse the domain in as few steps as possible. The optimum β can be estimated from the desired

latent space dimension and the number of stacked layers we are willing to allow, due to computational cost involved in training deep networks.

There are caveats to both these strategies: A larger receptive field; in the limit of $\alpha \rightarrow \infty$ (where ∞ refers to the dimensionality of the dataset) increases the number of trainable weights, with their number approaching the number of data-points in the domain. As mentioned before, this is computationally prohibitive for 3D datasets of even small sizes and is hence not feasible. This leads to the second strategy of increasing β , while retaining a relatively small α . This also has pitfalls due to large discontinuities created between adjacent receptive fields. A $\beta = 1$ leads to smooth transitions in convolution operations between subsequent layers, but requires large number of layers to achieve any meaningful dimensionality reduction. In contrast, $\beta > 1$ skips over some features in the domain, leading to some information loss in the smaller scales. However, it also leads to significant dimensionality reduction with fewer layers, which reduces computational costs. Figure 8 illustrates the effect of these parameters on the convolution kernel. A $\beta = 3$ for $\alpha = 3$ can quickly traverse the domain in fewer steps, while a $\beta = 1$ for $\alpha = 3$ ensures maximum overlap between adjacent receptive fields, at the cost of more traversal steps.

At this juncture, it is useful to develop some intuition on α and β in terms of numerical solution of partial differential equations in CFD. The convolutional kernel used in CAE has direct connections to the numerical stencils used in finite difference/finite volume approaches [60,61]. Consider the standard second-order central difference scheme in 1D for a quantity ϕ

$$\frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{\delta^2} \quad (9)$$

This can be represented as a 1D convolutional kernel of $\alpha = 3$ with three constant weights $\frac{1}{\delta^2}$, $-\frac{2}{\delta^2}$ and $\frac{1}{\delta^2}$. In the CAE, the kernel has the *same structure*, but all the constant weights in the convolutional kernels are replaced with learnable weights. Therefore, the output of trained kernel is analogous to a weighted combination of adjacent points, akin to numerical solution of PDEs. In fact, there are deeper connections between convolutional kernels and stencils of numerical schemes that have been uncovered recently for developing efficient neural network based PDE solvers, and the reader is directed to the Long et al. [61] and Dong et al. [60].

In numerical solutions of PDEs, the kernel size corresponds to the order of numerical scheme, which is typically constant and computed at *every point in the domain*. By analogy, larger stencils may represent higher order numerical schemes, as seen by an increased number of trainable weights in networks. Extending the CNN terminology to PDE solvers for comparison, a PDE solver has a constant α and $\beta = 1$ which completes its operation in a single ‘layer’. In contrast, the CAE has multiple layers with flexibility to have different α , β in each layer. Thus each layer of the CAE encoder consists of a customised numerical stencil specific to the dataset. In lieu of these close connections, the practical differences between PDE solvers and CAEs boil down to the treatment of boundaries, stride and the mapping of input features into a different subspace. In CAE, only the first layer in the deep neural network encoder treats the boundaries, while the increasing β at successive layers decreases dimensionality of the data. In summary, CAE encoders map the high dimensional input features into a low dimensional latent space with an intelligent choice of kernel weights, kernel sizes and stride lengths. The CAE decoder is essentially an inverse operation of

the encoder, but not in an explicit, mathematically exact fashion [62]. Instead the decoder weights and strides are trained with the encoder to estimate the inverse map from latent space to original data. These connections can be exploited to build CNNs with hard physics constraints based on numerical methods, and the reader is referred to Mohan et al. [63] for details.

6.2. Convolutional autoencoders: influence of kernel size and sequence length

The discussion thus far has emphasised the role of *kernel size* α , in a CNN (and therefore, a CAE) as a hyper-parameter with important consequences on the accuracy of our learned model. In this work, each batch trained consists of snapshots that retain their temporal order and are not shuffled. This means the CAE has to extract a low dimensional latent space from the dynamics of a temporal sequence, as opposed to learning from each snapshot as an independent sample. Consequently, the temporal gap between subsequent snapshots, i.e. *sampling-rate* ω becomes a factor building a DL-based ROM.

We now seek to study if the accuracy of the learned latent space is sensitive to this relationship. To understand the sensitivity of our model to ω , we increase the sampling rate for a constant batch size, to account for many real-world applications where data collection frequency is not ideal. Since the kernel performs a convolution operation over a numerical grid, its receptive field is intimately connected to the turbulence scales it captures. Intuitively, we would expect larger kernel sizes to capture spatial correlations of larger scales. Likewise, it follows that these kernels would also capture dynamics over longer time scales, due to the relationship between length and time scales in turbulence. By decreasing the sampling rate of the snapshots, we can account for these longer temporal scales, while keeping the batch size same. This ensures that all the differences we observe in model accuracy is not from the batch size, but rather its sampling frequency.

We intend to experimentally quantify the influence of α on the accuracy of the compression, for future applications in 3D turbulence. The goal is to observe if turbulent features over a range of scales orders of magnitudes apart, show any preferential dependence to learning by various kernel sizes and sampling rates. We choose ω to be 3, 6 and 9 samples apart, which corresponds to $\omega = 0.09\tau$, 0.18τ and 0.27τ , where τ is the eddy turnover time for this flow. Finally, such a hyper-parameter sweep would seek to establish that the results are consistent, and not due to chance numerical artefacts that may have occurred during optimisation. To this end, several experiments are performed with two families of parameters:

- (1) With a small kernel size $\alpha = 3$, vary sampling rate as $\omega = 0.09\tau, 0.18\tau, 0.27\tau$.
- (2) With large kernel size $\alpha = 9$, vary sampling rate as $\omega = 0.09\tau, 0.18\tau, 0.27\tau$.

For consistency, we ensure that the number of layers and the striding β in the encoder and decoder are constant for all experiments. The $\alpha = 9$ kernel creates a higher compression ratio than $\alpha = 3$, for the same number of layers in encoder and decoder. As a result, the only variables in the experiments are α and ω . All experiments above are also trained with three commonly used optimisers – Adam [64], Adadelta [65] and RMSProp [66] and the best model is used for analysis, to ensure the final trends are not a consequence of an arbitrary choice of optimiser, but instead an outcome of the α and ω choices. We now

present the results, and the trained model is assessed using the same diagnostic metrics mentioned in Section 4. All the diagnostics compare the statistics generated from (a) DNS snapshots and (b) CAE reconstructed models of their corresponding latent states. The previous discussion in Section 6.1 points to information loss in small scale behaviour due to $\beta > 1$. Quantifying the accuracy of the latent space with these physics based diagnostics will shed light if this indeed holds true.

6.2.1. Convolutional autoencoder: $\alpha = 3$

The statistical diagnostics for the small kernel $\alpha = 3$ with $\omega = 0.09\tau$ is shown in Figure 10. To indicate the quality of the model, we show diagnostics at three randomly chosen samples, followed by the averaged diagnostics for several samples. We adopt this style for all CAE results in this work. From the energy spectra, it is clear that large and inertial range frequencies are retained accurately, while there is a marked discrepancy in the small scale frequencies. This behaviour is also observed in the velocity gradient PDFs, where the large scale events around $Z = 0$ are well resolved, while discrepancies corresponding to small scales exist at the tails. Finally, the most stringent test is the $Q-R$ plane PDF, since it captures the 3D morphology of the flow. The $Q-R$ spectra at $r = 0$ corresponds to small scale behaviour, $r = 8$ for inertial range scales and $r = 32$ for large scale behaviour. The spectra shows excellent agreement at large scales, thereby corroborating the results from the energy spectra. The structure of the inertial range is also accurately captured, with very minor discrepancies in stretching behaviour. Finally, we see that the small scale behaviour is almost entirely neglected by the kernel. The symmetric nature of PDFs indicates that the network may be generating some random noise to compensate for information loss in the small scales. Interestingly, the discussion about β and its relationship to turbulence scales in the previous section indicates this outcome, which we have now verified. We now discuss the sensitivity of training with ω . The sampling rate is progressively decreased to $\omega = 0.09\tau, 0.18\tau$ and the diagnostics are shown in Figures 11 and 12 respectively. The diagnostics show that the quality of results are extremely robust despite a decrease in temporal sampling rate of the data. We caution that this may likely be true only for cases like stationary, homogeneous turbulence, whereas accuracy for flows with strong transients and non-stationarity can be affected by ω .

6.2.2. Convolutional autoencoder: $\alpha = 9$

We now turn our attention to the large kernel $\alpha = 9$ with $\omega = 0.09\tau$. The diagnostics in Figure 13 paint a somewhat different picture in comparison with the small kernel. The energy spectra shows good agreement in the low wavenumbers, but gets progressively worse with increasing wavenumber. Finally, the high wavenumbers show major discrepancies with oscillatory behaviour not present in the DNS dataset. On the other hand, the velocity gradient PDFs show a much better agreement with the DNS than the small kernel. This seemingly counter-intuitive behaviour likely happens due to the random high wavenumber oscillations (seen in the energy spectra) fortuitously replicating averaged small scale intermittent fluctuations in DNS. Finally, we get a clear understanding of the large kernel performance looking at the $Q-R$ PDF statistics. The statistics show good large scale reconstruction, but significant discrepancies in the inertial range, with the somewhat symmetric stretching in the R axis implying addition of random noise to the lower quadrant of the $Q-R$ plane. The noise effect is further accentuated in the small scale statistics, with

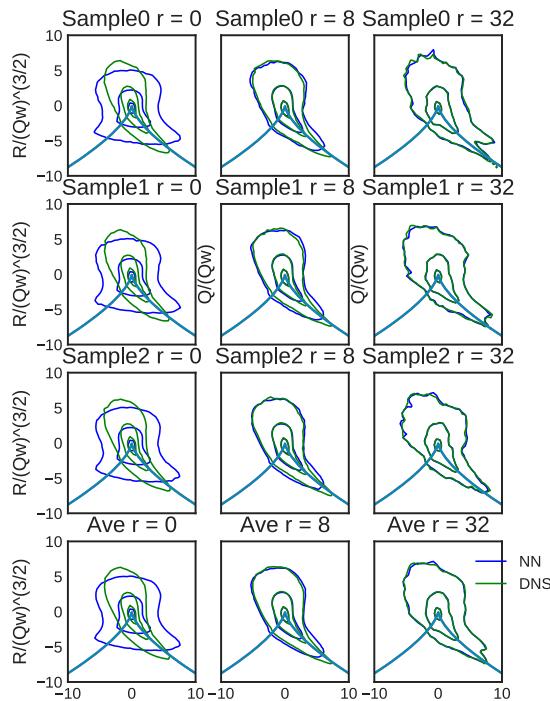
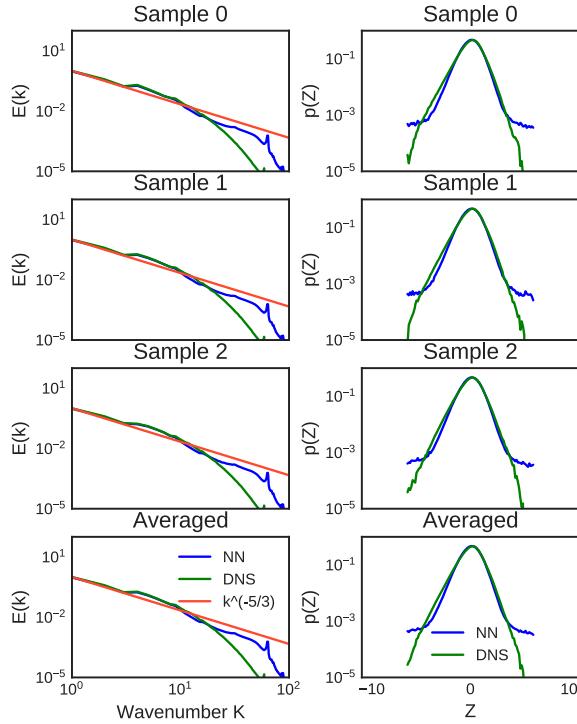


Figure 10. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen samples from CAE-NN dimensionality reduction with $\alpha = 3$ and $\omega = 0.09\tau$.

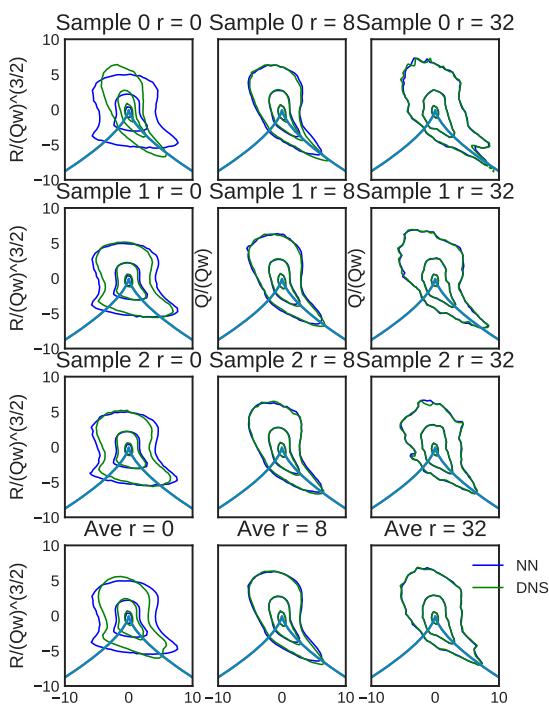
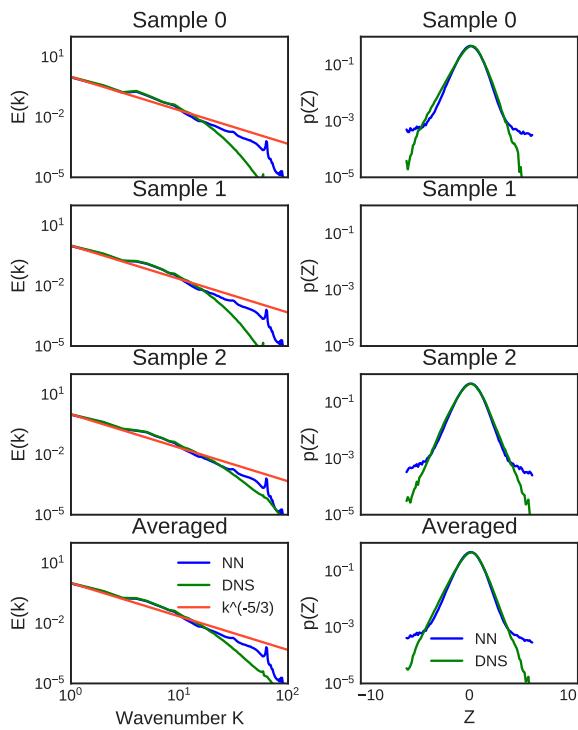


Figure 11. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen samples from CAE-NN dimensionality reduction with $\alpha = 3$ and $\omega = 0.18\tau$.

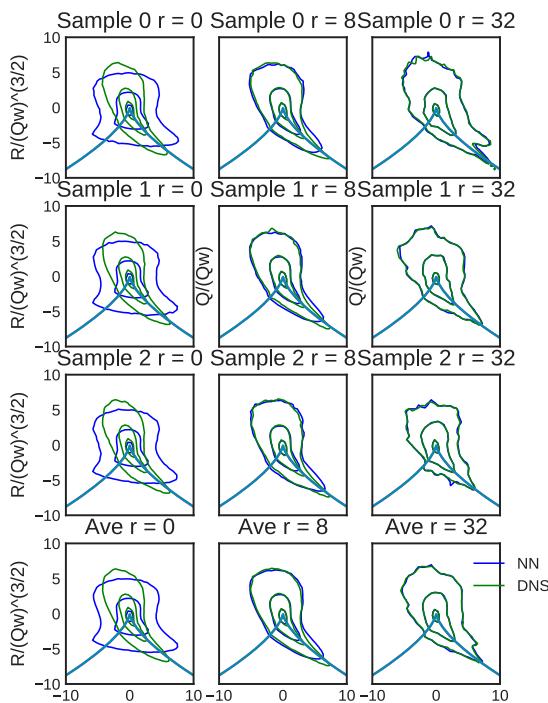
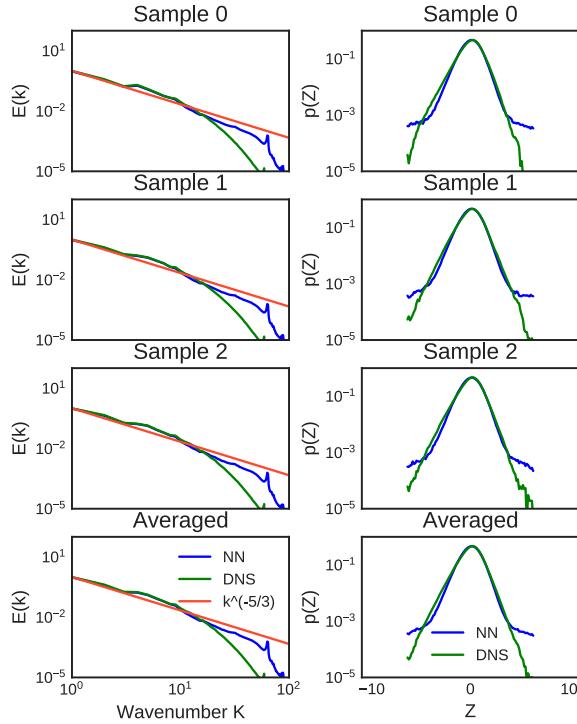


Figure 12. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen samples from CAE-NN dimensionality reduction with $\alpha = 3$ and $\omega = 0.27\tau$.

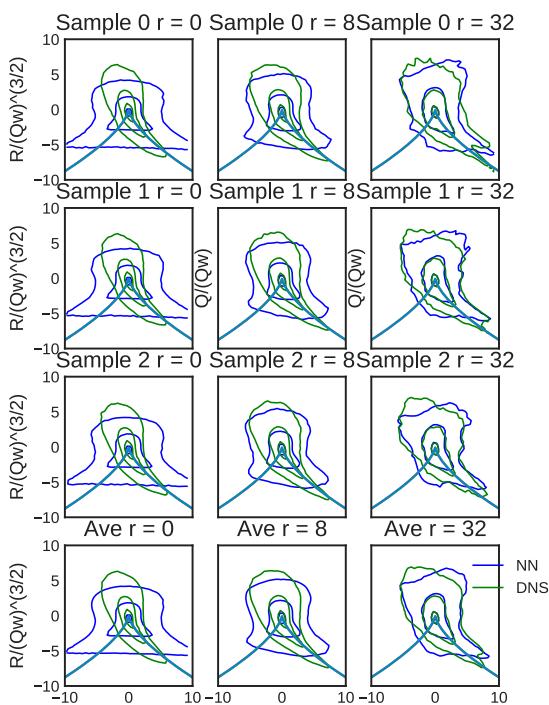
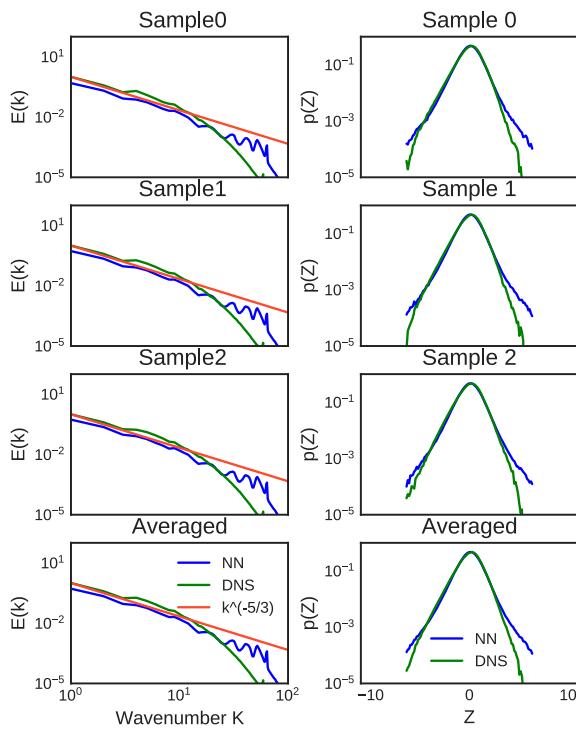


Figure 13. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen samples from CAE-NN dimensionality reduction with $\alpha = 9$ and $\omega = 0.09\tau$.

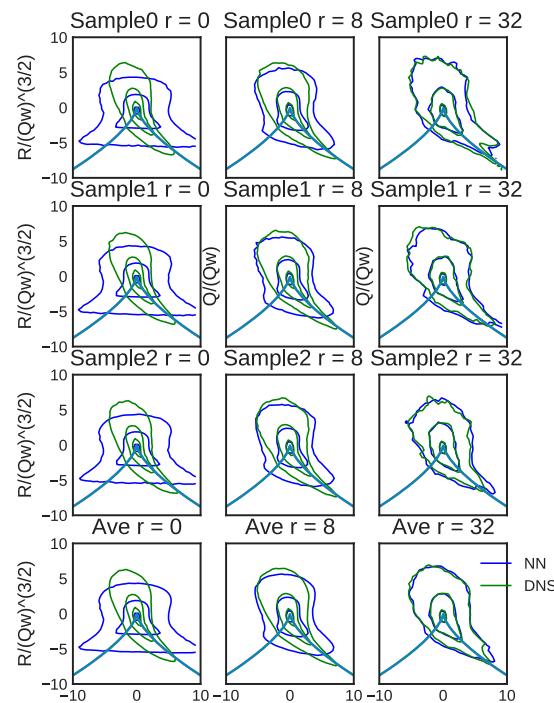
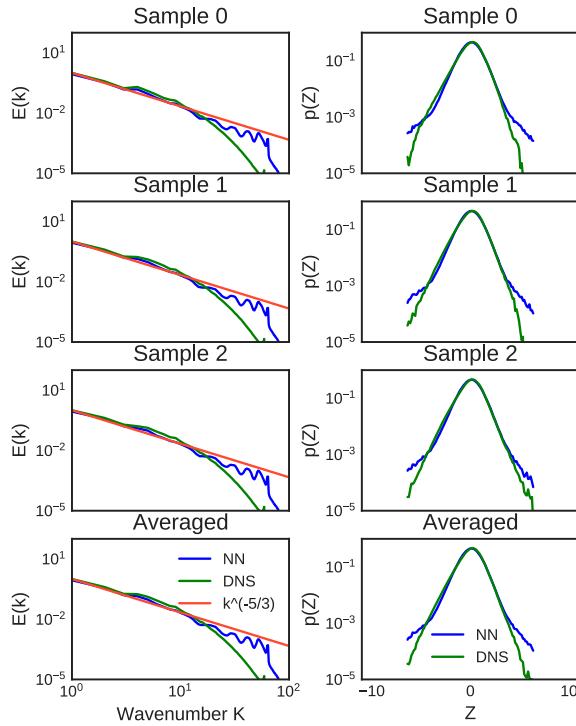


Figure 14. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen samples from CAE-NN dimensionality reduction with $\alpha = 9$ and $\omega = 0.18\tau$.

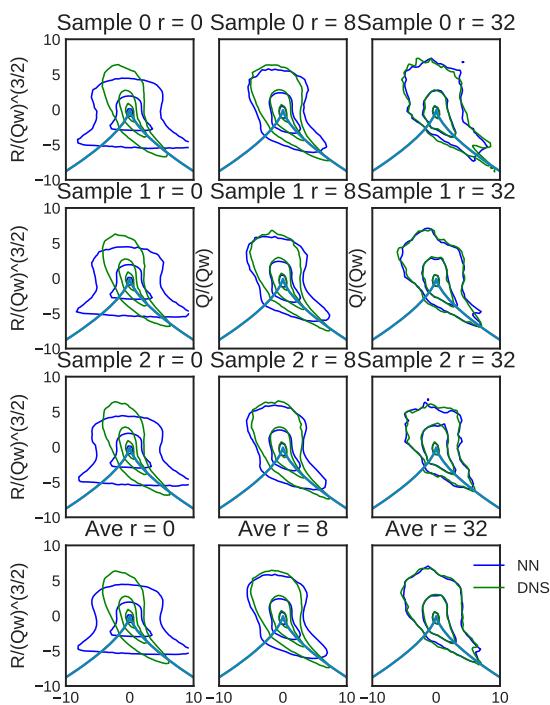
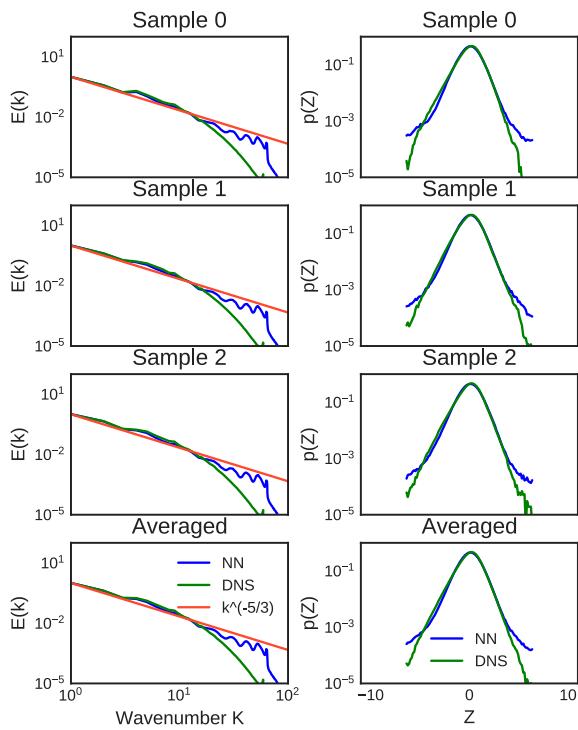


Figure 15. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom) for randomly chosen samples from CAE-NN dimensionality reduction with $\alpha = 9$ and $\omega = 0.27\tau$.

appreciable deviations from the DNS statistics. Similar to the small kernel, experiments are also performed for $\omega = 0.09\tau, 0.18\tau$ and the diagnostics are shown in Figures 14 and 15 respectively. For $\omega = 0.18\tau$ we see similar behaviour as $\omega = 0.09\tau$, except for minor discrepancies in the large scales. These trends are repeated in $\omega = 0.27\tau$. Overall, the quality of reconstruction does not seem to change with decreasing sampling frequency, as seen for $\alpha = 3$. Furthermore, all the results show consistent addition of random noise to high wavenumbers and several inertial-range wavenumbers. The presence of noise in the large kernel happens to be the most significant difference from the small kernel, which consequently leads to deterioration in reconstruction. It bears mentioning that the large kernel contains more parameters than the small kernel, and as such needs significantly longer training time to obtain convergence. In this work, the training time for $\alpha = 9$ was twice that of $\alpha = 3$, and the memory requirements were considerably higher.

From these experiments we can conclude that, at least for the case of isotropic turbulence, the kernel size appears to be a more important parameter affecting model accuracy than the sampling rate of the data. We note that while a large kernel is capable of higher compression ratios than a small kernel for the same layers, it comes at the price of accuracy, computational time and memory. While both large and small kernels capture large scale behaviour well, the small kernel also reconstructs the inertial scales reasonably well.

7. Results using compressed convolutional LSTM (CC-LSTM)

As discussed previously, the ConvLSTM network (Figure 2) necessitates some form of data compression to efficiently learn the spatio-temporal dynamics of the flow with tractable computational effort. The CAEs described above are seen to learn efficient latent space representations of the flow with excellent compression in data size, and we denote the combined approach as CC-LSTM. As mentioned in Section 3, we use as training data the time-varying latent space for $\tau = 3$ snapshots. After the parametric study with different α and ω , it is observed that $\alpha = 3$ and $\omega = 0.09\tau$ learn sufficiently accurate models with the lowest computational cost. Therefore, we use the latent space models from this configuration as the ConvLSTM training data.

Since a ConvLSTM network can model spatio-temporal dynamics, we evaluate it by making continuous predictions in time. We give a batch of temporal flow snapshots compressed into CAE latent spaces as input and the network predicts the next batch of latent spaces evolved in time. These predicted latent spaces are then used to recover the true dimensions of the flow thru the CAE decoder. The model is autoregressive, since the predictions are fed back into the network as a new input. We repeat this autoregressive process for several time instants, to study both the accuracy of the predicted snapshots, and how far in time the network is able to generate stable snapshots without significant deterioration in accuracy. The diagnostic tests outlined in Section 4 are used to evaluate CC-LSTM generated snapshots. The velocity diagnostics are shown in Figure 16 for predicted snapshots at 1.5 eddy times from $\tau = 3 - 4.5$ in the DNS dataset. We make autoregressive predictions in $\tau^* = \tau - 3 = 0 \rightarrow 1.5$, and the diagnostics are shown for $\tau^* = 0.1, 1.0, 1.5$ such that we are evaluating temporally correlated snapshots across the predicted range. The ConvLSTM network has 3 layers with constant kernel size $\alpha = 3$, with each hidden cell having 40 features and RMSProp optimiser used to train the network. The approach was implemented using the Pytorch [67] framework and trained in a distributed multi-GPU batch-parallel fashion.

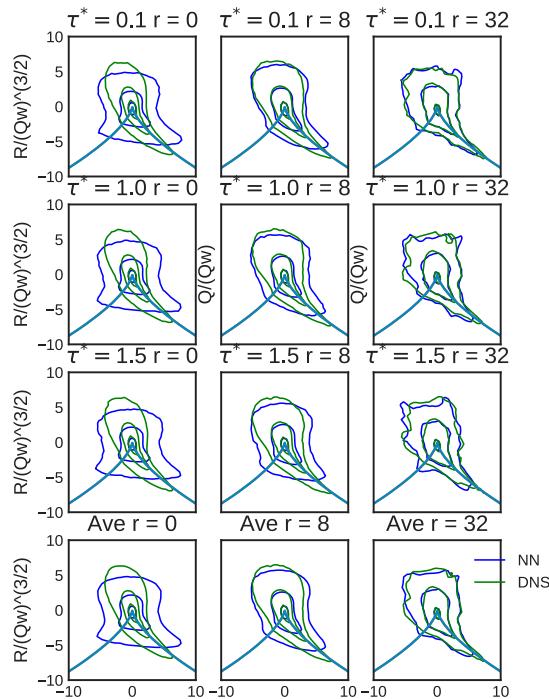
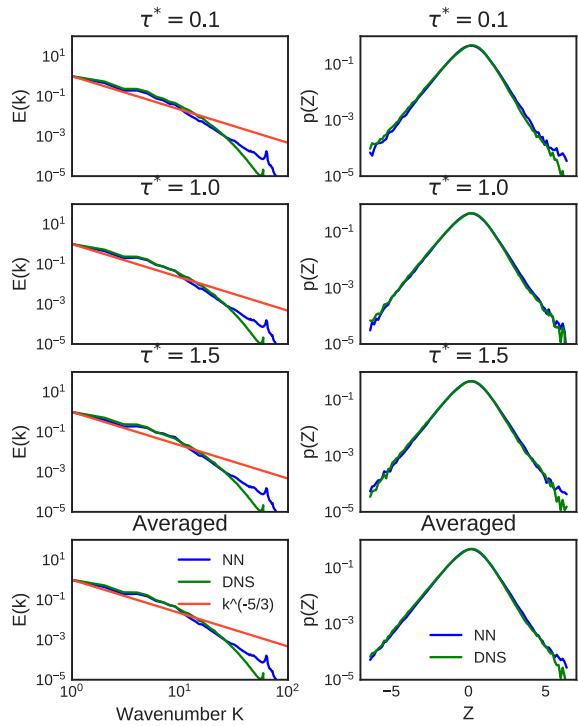


Figure 16. Energy spectra (top left), PDF of the longitudinal velocity gradient magnitude (top right), and joint PDFs of the Q and R invariants of the coarse-grained velocity gradient tensor (bottom). CC-LSTM predictions are more accurate than CGANs and temporally stable, with errors concentrated in the small scales.

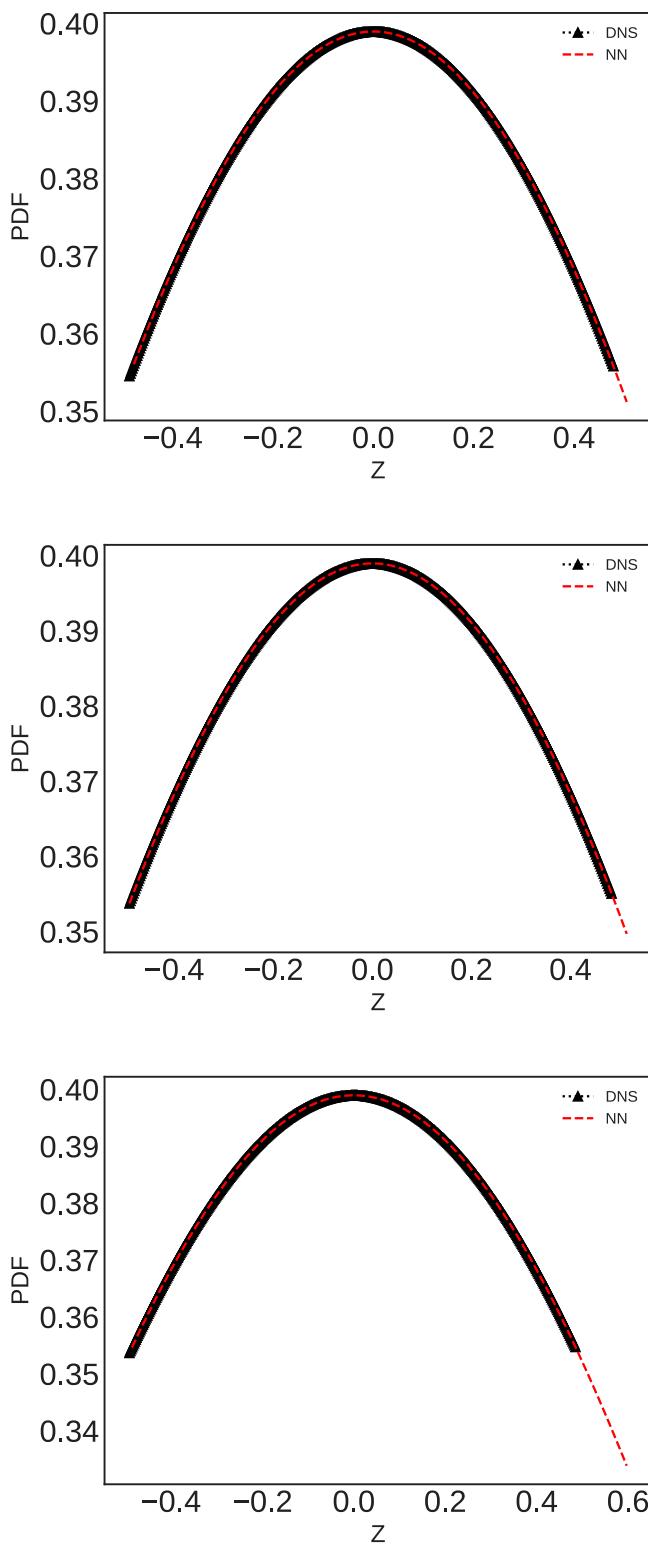


Figure 17. Instantaneous ϕ_1 scalar PDFs bounded $[-0.5, +0.5]$ from DNS and NN at different time instances: $\tau^* = 0.1$ (top), $\tau^* = 1.0$ (middle), $\tau^* = 1.5$ (bottom). CC-LSTM predictions are bounded and more accurate than CGANs with minor discrepancies at tails.

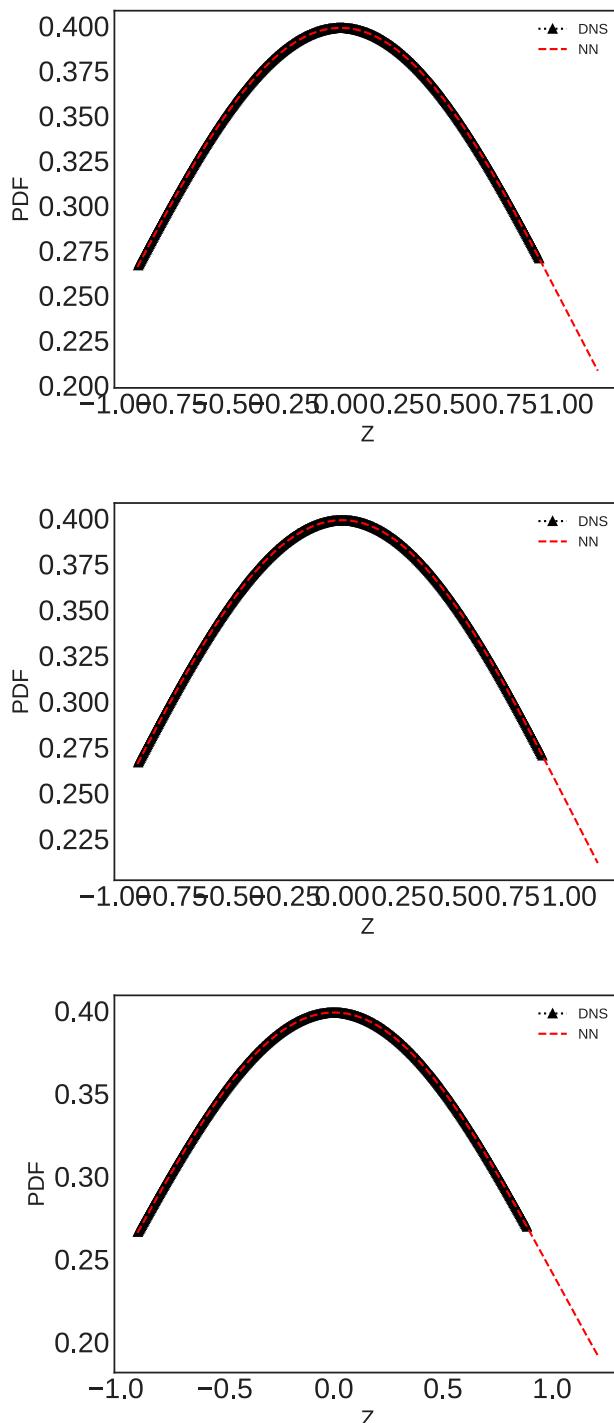


Figure 18. Instantaneous ϕ_2 scalar PDFs bounded $[-1.0, +1.0]$ from DNS and NN at different time instances: $\tau^* = 0.1$ (top), $\tau^* = 1.0$ (middle), $\tau^* = 1.5$ (bottom). CC-LSTM predictions are bounded and more accurate than CGANs with minor discrepancies at tails.

We see from the energy spectra that the large scale and inertial range spectra are predicted extremely well, with discrepancies only in the small scale range. Interestingly, the velocity gradient PDFs show near-perfect resolution across all the scales, including the small scale behaviour at the tails. This likely indicates the ConvLSTM network is adding some artefacts to the predictions which accurately mimics the tail behaviour of the PDF, since this was not a condition we enforced on the network. A more rigorous evaluation is performed with the $Q-R$ PDFs, where we see that the statistical trends of the small scales are neglected by the network as expected. Furthermore, we see that the large scale trends are predicted quite well, followed by inertial range scales with some discrepancies. Typically, most temporal modelling techniques are accompanied by a significant loss in accuracy as the prediction horizon τ^* increases. In this case, we see only marginal deterioration in large scale statistics at $\tau^* > 1$. The loss of accuracy is somewhat more significant in the inertial scales, while the small scales do not see much change. From these diagnostics, it is apparent that the CC-LSTM is able to consistently model large scale velocity dynamics of ScalarHIT over extended time ranges, even while the accuracy in other scales might suffer. This is quite promising, since modelling large scale dynamics at high fidelity is a requirement for several practical applications.

We now turn our attention to the passive scalars. Figure 17 compares PDFs of the DNS and CC-LSTM predictions at $\tau^* = 0.1, 1.0, 1.5$ for the scalar ϕ_1 . We observe that the predicted scalar is generally accurate and well bounded in $(-0.5, 0.5)$ with only a minor overshoot. This is also observed in the predictions of scalar ϕ_2 in Figure 18, where the network not only captures the PDF, but also tries to retain boundedness. These results show that the CC-LSTM are able to outperform CGANs in predicting both velocity vectors and passive scalars despite loss of information arising from the autoencoders.

Finally, a key factor to evaluate these architectures is the computational resources required to train an ROM. Since the CC-LSTM primarily learns dynamics on latent space rather than the high dimensional raw data, it requires orders of magnitude fewer parameters than CGANs, which has generally been the more popular approach in the turbulence community. The details of the computational costs are outlined in Appendix A.5, and show significant advantages of CC-LSTM over CGANs when scaling these approaches to large, realistic flows. Furthermore, we note that training GANs/CGANs in a stable manner involves several modifications over hyper-parameters in both the network design and optimisation, and has been well documented elsewhere in the broader machine learning literature [32,68–70]. In this work, the authors had to implement several strategies outlined in Appendix A.2 to obtain reliable predictions using CGANs. In contrast, CC-LSTM training was markedly more stable and resilient to variations in hyper-parameter choices across different kernel sizes and sequence lengths, further reducing compute cost.

8. Conclusion

In this work, we report a first systematic study of deep learning strategies for the generation of fully developed three-dimensional turbulence. We evaluate neural network architectures representing two different approaches to high-dimensional data modelling. The quality of the deep learning predictive models is tested with physics-based metrics which identify the statistical characteristics of 3D turbulence. The first architecture is a 3D convolutional variant of popular approach known as Generative Adversarial Networks (GANs). In this work,

Convolutional GANs (CGANs) are demonstrated to have acceptable accuracy in modelling large and inertial scale velocity features of individual snapshots of the flow, albeit without capability for temporal predictions. However, we also notice CGANs difficulties in modelling the probability density functions (PDFs) of the passive scalars advected with the velocity, with the predictions being frequently unbounded. Since CGANs lack temporal dynamics, we propose an alternative neural network approach to perform spatio-temporal prediction. This novel strategy utilises a convolutional autoencoder (CAE) neural network followed by a Convolutional LSTM (ConvLSTM) network. The CAE learns a projection of the high-dimensional spatial data to a low dimensional latent space, such that the latent space can be used as an input for temporal predictions. We then employ the ConvLSTM network to predict the latent space at future time instants. This two-tier prediction model, coined Compressed Convolutional LSTM (CC-LSTM), is able to predict dynamics of the flow. Furthermore, the CC-LSTM allows accurate reproduction of the large and inertial scale statistics making it very attractive for many practical/engineering applications. In case of the passive scalars, the CC-LSTM is able to capture the PDFs accurately, while bounding the scalar PDFs within its theoretical limits with only minor overshoot, as opposed to CGANs. From a practical standpoint, one of the major observations of this investigation is significant disparity between computational efficiency of CC-LSTM, when compared with popular, state-of-the art approaches like CGANs, in the context of 3D turbulence. Due to large number of parameters that ConvLSTM networks need even for modestly sized datasets, we show that performing model reduction with CAEs is a valuable first step in computationally efficient learning models of 3D turbulence. This modified CC-LSTM approach needs orders of magnitude fewer trainable parameters than CGANs, while showing superior spatio-temporal predictive accuracy. While the networks shown in this work do not have explicit physics constraints, versions of autoencoders with hard constraints demonstrated by the authors in [63] can be easily adapted to the CC-LSTM framework, providing considerable flexibility in learning.

Acknowledgements

The authors thank Don Daniel (LANL) for the dataset and valuable discussions. This work has been authored by employees of Triad National Security, LLC, which operates Los Alamos National Laboratory (LANL) under Contract No. 89233218CNA000001 with the U.S. Department of Energy/National Nuclear Security Administration. A.T.M. and D.L. have been supported by LANL's LDRD program, project number 20190058DR. A.T.M. also thanks the Center for Nonlinear Studies at LANL for support and acknowledges the ASC/LANL Darwin cluster for GPU computing infrastructure.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by U.S. Department of Energy [LDRD Los Alamos National Laboratory].

ORCID

Arvind T. Mohan  <http://orcid.org/0000-0002-9434-7691>

References

- [1] Wu JL, Xiao H, Paterson E. Physics-informed machine learning approach for augmenting turbulence models: a comprehensive framework. *Phys Rev Fluids*. **2018**;3(7):074602.
- [2] Wang JX, Wu J, Ling J, et al. A comprehensive physics-informed machine learning framework for predictive turbulence modeling. Preprint arXiv:170107102. 2017.
- [3] Tracey BD, Duraisamy K, Alonso JJ. A machine learning strategy to assist turbulence model development. In: Proceedings of the 53rd AIAA Aerospace Sciences Meeting; Kissimmee, Florida, Publisher: AIAA (American Institute of Aeronautics and Astronautics), 2015. p. 1287.
- [4] Singh AP, Medida S, Duraisamy K. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA J*. **2017**;55(7):2215–2227.
- [5] Maulik R, San O, Rasheed A, et al. Subgrid modelling for two-dimensional turbulence using neural networks. *J Fluid Mech*. **2019**;858:122–144.
- [6] Ling J, Kurzawski A, Templeton J. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *J Fluid Mech*. **2016**;807:155–166.
- [7] Klein M, Sadiki A, Janicka J. A digital filter based generation of inflow data for spatially developing direct numerical or large eddy simulations. *J Comput Phys*. **2003**;186(2):652–665.
- [8] Di Mare L, Klein M, Jones W, et al. Synthetic turbulence inflow conditions for large-eddy simulation. *Phys Fluids*. **2006**;18(2):025107.
- [9] Juneja A, Lathrop D, Sreenivasan K, et al. Synthetic turbulence. *Phys Rev E*. **1994**;49(6):5179.
- [10] Jarrin N, Benhamadouche S, Laurence D, et al. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *Int J Heat Fluid Flow*. **2006**;27(4):585–593.
- [11] Rempfer D. On low-dimensional galerkin models for fluid flow. *Theoret Comput Fluid Dynam*. **2000**;14(2):75–88.
- [12] Noack BR, Papas P, Monkewitz PA. The need for a pressure-term representation in empirical Galerkin models of incompressible shear flows. *J Fluid Mech*. **2005**;523:339–365.
- [13] Carlberg K, Bou-Mosleh C, Farhat C. Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. *Int J Numer Methods Eng*. **2011**;86(2):155–181.
- [14] Qian E, Kramer B, Peherstorfer B, et al. Lift & learn: physics-informed machine learning for large-scale nonlinear dynamical systems. *Phys D Nonlinear Phenomena*. **2020**;406:132401.
- [15] Deshmukh R, McNamara JJ, Liang Z, et al. Model order reduction using sparse coding exemplified for the lid-driven cavity. *J Fluid Mech*. **2016**;808:189–223.
- [16] Sargsyan S, Brunton SL, Kutz JN. Nonlinear model reduction for dynamical systems using sparse sensor locations from learned libraries. *Phys Rev E*. **2015**;92(3):033304.
- [17] Kaiser E, Noack BR, Cordier L, et al. Cluster-based reduced-order modelling of a mixing layer. *J Fluid Mech*. **2014**;754:365–414.
- [18] Nair AG, Taira K. Network-theoretic approach to sparsified discrete vortex dynamics. *J Fluid Mech*. **2015**;768:549–571.
- [19] Goodfellow I, Pouget-Abadie J, Mirza M. Generative adversarial nets. In: Proceedings of the Advances in Neural Information Processing Systems; 2014, Montreal, Canada. Publisher: Neural Information Processing Systems, p. 2672–2680.
- [20] Xingjian S, Chen Z, Wang H. Convolutional lstm network: a machine learning approach for precipitation nowcasting. In: Proceedings of the Advances in Neural Information Processing Systems; Montreal, Canada. Publisher: Neural Information Processing Systems, 2015. p. 802–810.
- [21] Chu M, Thurey N. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Trans Graphics (TOG)*. **2017**;36(4):69.
- [22] Um K, Hu X, Thurey N. Liquid splash modeling with neural networks. In: Computer Graphics Forum, Vol. 37. Wiley Online Library; 2018. p. 171–182.
- [23] Mukherjee R, Li Q, Chen Z, et al. Neuraldrop Dnn-based simulation of small-scale liquid flows on solids. Preprint arXiv:181102517. 2018.
- [24] Zimmermann RS, Parlitz U. Observing spatio-temporal dynamics of excitable media using reservoir computing. *Chaos: An Interdisciplinary J Nonlinear Sci*. **2018**;28(4):043118.

- [25] Lu Z, Pathak J, Hunt B, et al. Reservoir observers: model-free inference of unmeasured variables in chaotic systems. *Chaos Interdisc J Nonlinear Sci.* **2017**;27(4):041102.
- [26] Pathak J, Wikner A, Fussell R, et al. Hybrid forecasting of chaotic processes: using machine learning in conjunction with a knowledge-based model. *Chaos Interdisc J Nonlinear Sci.* **2018**;28(4):041101.
- [27] Pathak J, Hunt B, Girvan M, et al. Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. *Phys Rev Lett.* **2018**;120(2):024102.
- [28] Wu JL, Kashinath K, Albert A, et al. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. Preprint arXiv:190506841. 2019.
- [29] King R, Hennigh O, Mohan A, et al. From deep to physics-informed learning of turbulence: diagnostics. Preprint arXiv:181007785. 2018.
- [30] Yang Z, Wu JL, Xiao H. Enforcing deterministic constraints on generative adversarial networks for emulating physical systems. Preprint arXiv:191106671. 2019.
- [31] Chen W, Chiu K, Fuge M. Aerodynamic design optimization and shape exploration using generative adversarial networks. In: Proceedings of the AIAA Scitech 2019 Forum. 2019, San Diego, California. Publisher: AIAA (American Institute of Aeronautics and Astronautics), p. 2351.
- [32] Radford A, Metz L, Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks. Preprint arXiv:151106434. 2015.
- [33] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. e-prints. arXiv:1502.03167. 2015 Feb.
- [34] Che T, Li Y, Jacob AP, et al. Mode regularized generative adversarial networks. Preprint arXiv:161202136. 2016.
- [35] Salimans T, Goodfellow IJ, Zaremba W, et al. Improved techniques for training gans. CoRR. 2016. abs/1606.03498. <http://arxiv.org/abs/1606.03498>.
- [36] Deng L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process Mag.* **2012**;29(6):141–142.
- [37] Holmes PJ, Lumley JL, Berkooz G, et al. Low-dimensional models of coherent structures in turbulence. *Phys Rep.* **1997**;287(4):337–384.
- [38] Mezić I. Analysis of fluid flows via spectral properties of the Koopman operator. *Annu Rev Fluid Mech.* **2013**;45:357–378.
- [39] Bagheri S. Koopman-mode decomposition of the cylinder wake. *J Fluid Mech.* **2013**;726:596–623.
- [40] Rowley CW, Mezić I, Bagheri S, et al. Spectral analysis of nonlinear flows. *J Fluid Mech.* **2009**;641:115–127.
- [41] Lusch B, Kutz JN, Brunton SL. Deep learning for universal linear embeddings of nonlinear dynamics. *Nat Commun.* **2018**;9(1):4950.
- [42] Yeung E, Kundu S, Hodas N. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. Preprint arXiv:170806850. 2017.
- [43] Sirisup S, Karniadakis GE. A spectral viscosity method for correcting the long-term behavior of pod models. *J Comput Phys.* **2004**;194(1):92–116.
- [44] Mohan AT, Gaitonde DV. A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks. Preprint arXiv:180409269. 2018.
- [45] Theis L, Shi W, Cunningham A, et al. Lossy image compression with compressive autoencoders. Preprint arXiv:170300395. 2017.
- [46] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; **2016**.
- [47] Qi CR, Su H, Nießner M. Volumetric and multi-view cnns for object classification on 3d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; Las Vegas, Nevada. Publisher: IEEE 2016, p. 5648–5656.
- [48] Luong MT, Manning CD. Stanford neural machine translation systems for spoken language domains. In: Proceedings of the International Workshop on Spoken Language Translation; 2015, Da Nang, Vietnam. Publisher: International Workshop on Spoken Language Translation. p. 76–79.

- [49] Nelson DM, Pereira AC, de Oliveira RA. Stock market's price movement prediction with lstm neural networks. In: Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN). IEEE; 2017. Anchorage, Alaska p. 1419–1426.
- [50] Wu H, Prasad S. Convolutional recurrent neural networks for hyperspectral data classification. *Remote Sens (Basel)*. 2017;9(3):298.
- [51] Zhu G, Zhang L, Shen P, et al. Multimodal gesture recognition using 3-d convolution and convolutional lstm. *IEEE Access*. 2017;5:4517–4524.
- [52] Zhao R, Yan R, Wang J, et al. Learning to monitor machine health with convolutional bi-directional lstm networks. *Sensors*. 2017;17(2):273.
- [53] Daniel D, Livescu D, Ryu J. Reaction analogy based forcing for incompressible scalar turbulence. *Phys Rev Fluids*. 2018;3(9):094602.
- [54] Cook AW, Dimotakis PE. Transition stages of Rayleigh–Taylor instability between miscible fluids. *J Fluid Mech*. 2001;443:69–99.
- [55] Livescu D, Ristorcelli J. Variable-density mixing in buoyancy-driven turbulence. *J Fluid Mech*. 2008;605:145–180.
- [56] Frisch U. Turbulence: the legacy of a. n. kolmogorov. Cambridge University Press; 1995.
- [57] Chertkov M, Pumir A, Shraiman BI. Lagrangian tetrad dynamics and the phenomenology of turbulence. *Phys Fluids*. 1999;11(8):2394–2410.
- [58] Mogren O. C-rnn-gan: continuous recurrent neural networks with adversarial training. Preprint arXiv:161109904. 2016.
- [59] Gonzalez FJ, Balajewicz M. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. Preprint arXiv:180801346. 2018.
- [60] Dong B, Jiang Q, Shen Z. Image restoration wavelet frame shrinkage, nonlinear evolution PDES, and beyond. *Multiscale Model Simul*. 2017;15(1):606–660.
- [61] Long Z, Lu Y, Ma X, et al. Pde-net: learning pdes from data. Preprint arXiv:171009668. 2017.
- [62] Ardizzone L, Kruse J, Wirkert S, et al. Analyzing inverse problems with invertible neural networks. Preprint arXiv:180804730. 2018.
- [63] Mohan AT, Lubbers N, Livescu D, et al. Embedding hard physical constraints in neural network coarse-graining of 3d turbulence. Preprint arXiv:200200021. 2020.
- [64] Kingma DP, Ba J. Adam: a method for stochastic optimization. Preprint arXiv:14126980. 2014.
- [65] Zeiler MD. Adadelta: an adaptive learning rate method. Preprint arXiv:12125701. 2012.
- [66] Tieleman T, Hinton G. Lecture 6.5-rmsprop divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012;4(2):26–31.
- [67] Paszke A, Gross S, Massa F. Pytorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems; 2019. Vancouver, Canada, Publisher: Neural Information Processing Systems p. 8024–8035.
- [68] Thanh-Tung H, Tran T, Venkatesh S. Improving generalization and stability of generative adversarial networks. Preprint arXiv:190203984. 2019.
- [69] Arora S, Ge R, Liang Y. Generalization and equilibrium in generative adversarial nets (gans). In: Proceedings of the 34th International Conference on Machine Learning, Vol 70; JMLR.org; Sydney, Australia. Publisher: International Machine Learning Society 2017. p. 224–232.
- [70] Arora S, Risteski A, Zhang Y. Do gans learn the distribution? some theory and empirics. ICLR; 2018 Vancouver, Canada. Publisher: International Conference on Representation Learning.
- [71] Hochreiter S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Internat J Uncertain Fuzziness and Knowledge-Based Syst*. 1998;6(02):107–116.
- [72] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput*. 1997;9(8):1735–1780.
- [73] Odena A, Dumoulin V, Olah C. Deconvolution and checkerboard artifacts. Distill; 2016. Available from: <http://distill.pub/2016/deconv-checkerboard>.
- [74] Goodfellow I. Nips 2016 tutorial: generative adversarial networks. 2016. Available from: <http://arxiv.org/abs/1701.00160>.
- [75] Smith LN. Cyclical learning rates for training neural networks. In: Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV); IEEE; 2017. p. 464–472.

Appendices

Appendix 1. Overview of neural network architectures

Artificial Neural networks (ANNs) can be broadly defined as a class of biologically inspired statistical representations which capture patterns and connections in a dataset. The elementary unit of an ANN is the *artificial neuron*, and a *layer* of an ANN consists of multiple neurons. Subsequently, ‘Deep’ ANNs are built by stacking multiple such layers one after the other. Mathematically, stacking numerous neurons in a connected manner (often ranging from hundreds to millions) is able to represent complex nonlinear functions, i.e. ANNs can universally approximate any function; which is the paradigm behind the rise of the modern ‘deep learning’ revolution. The mathematical representation of a neuron is shown in Equation (A1)

$$y = \phi \left(\sum_{j=0}^m w_{kj} x_j \right) \quad (\text{A1})$$

where x is the vector of inputs, with w being the series of ‘weights’ that produce the output y . The right side of the equation is operated upon by an activation function ϕ , which can be nonlinear. The key idea behind ANNs is that the weights w can be *learned* or estimated, given x and y . This is typically accomplished by the backpropagation algorithm which iteratively computes w for any $x-y$ pair via optimisation (usually gradient descent)-based methods. This process is broadly termed as *training* an ANN. While the core strategy of having learnable weights estimated with backpropagation and optimisation methods have been the mainstay of deep learning, the actual architecture of the ANN has greatly evolved. Specifically, these variations have focused on the structure and layout of w , such that they are tailor-made for specific applications. Most applications can be broadly grouped into two classes of prediction problems: (a) classification and (b) regression. Classification problems are often found in large image datasets; arising from satellite imagery, consumer devices and even scientific observations. Likewise, regression problems are ubiquitous in financial markets, consumer demand forecasting, weather forecasting, and numerous scientific applications. Thus most of the modern architectures in deep learning have adapted the standard ANN layout of w to account for these classes, which we will now outline.

A.1. Convolutional neural networks

Classification problems for image and other spatially varying datasets are difficult due to the large number of degrees of freedom. For instance, a 256×256 image has 65,536 datapoints. The ANN has all neurons connected to every other neuron, known as a fully connected NN or FCNN. It is immediately apparent that training an FCNN for 65,536 points requires *at least* as many parameters, if not more. Therefore, FCNNs for images can be computationally prohibitive. However, we know that images often have spatial correlations, so treating each datapoint in isolation may not be very accurate (or efficient). As such, *Convolutional Neural Networks* (CNNs) are a class of ANNs developed which exploits the fact that each point in an image is likely related to its neighbouring points. The w are structured as a *kernel* which translates throughout the image. The kernel is essentially a function that performs a convolution operation over the image data, and the result is trained using backpropagation. Therefore, instead of learning w for each individual point in the image, we learn w in a kernel with a predetermined size. This kernel can be used to extract patterns in other images. CNNs are often the driving force behind several state of the art image and pattern recognition problems, and the idea of training a single kernel instead of an FCNN leads to drastic reductions in computational cost.

A.2. Recurrent neural networks

Sequence prediction is different from other types of learning problems, since it imposes an order on the observations that must be preserved when training models and making predictions. Recurrent

Neural Networks (RNNs) are a class of neural networks specially designed for such problems, which preserve this sequential information in the function being learned. A key assertion behind Recurrent networks is that sequential processes have ‘memory’, i.e. the value of the process is a function of the value at previous instants. Recurrent networks attempt to capture this sequential relationship and learn the memory effects. The Long Short-Term Memory (LSTM) neural network is a special variant of RNN, which overcomes stability bottlenecks encountered in traditional RNNs (like the Vanishing Gradient problem [71]), enabling its practical application. LSTMs can also learn and harness sequential dependence from the data, such that the predictions are conditional on the recent context in the input sequence. For instance, to predict the realisation at time t_i , LSTMs can learn from the data at t_{i-1} and also at times t_{i-k} , where k can be any number signifying the length of the prior sequence. In effect, k represents the ‘memory’ in the system, i.e. the extent to which the outcome of the system depends on its previous realisations.

The basic architecture of the LSTM NN is now outlined. The LSTM networks are different from other deep learning architectures like convolutional neural networks (CNNs), in that the typical LSTM cell contains three *gates*: The **input** gate, **output** gate, and the **forget** gate. The LSTM regulates the flow of training information through these gates by selectively adding information (input gate), removing (forget gate) or letting it through to the next cell (output gate). A schematic of the cells connected in a recurrent form is shown in Figure A1.

The input gate is represented by i , output gate by o and forget gate by f . The cell state is represented as C and the cell output is given by h , while the cell input is denoted as x . Consider the equations of an LSTM cell to compute its gates and states in Equation (A2) and a schematic of its structure in Figure A2.

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \quad (\text{A2})$$

W are the weights for each of the gates and \tilde{C} is the updated cell state. These states are propagated ahead through the network, as shown in Figure A1 and weights are updated by backpropagation through time. The forget gate plays a crucial role in reducing over-fitting by not retaining all information from the previous time steps. This arrangement of gates and selective information control is also the key reason why LSTMs do not suffer from the vanishing gradient problem which plagued traditional RNNs [71]. As a result, LSTMs are a powerful tool to model sequential datasets. A more detailed and introduction to LSTM can be found in Ref. [72].

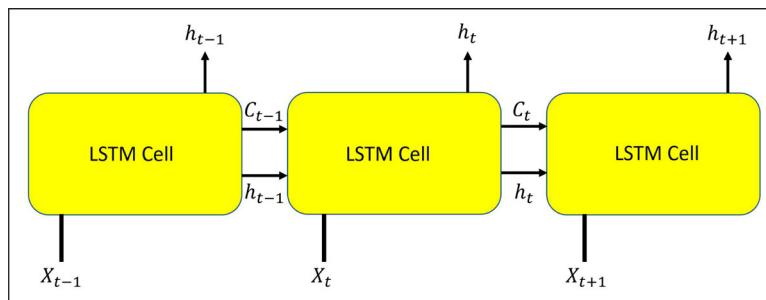


Figure A1. LSTM layout with cell connections.

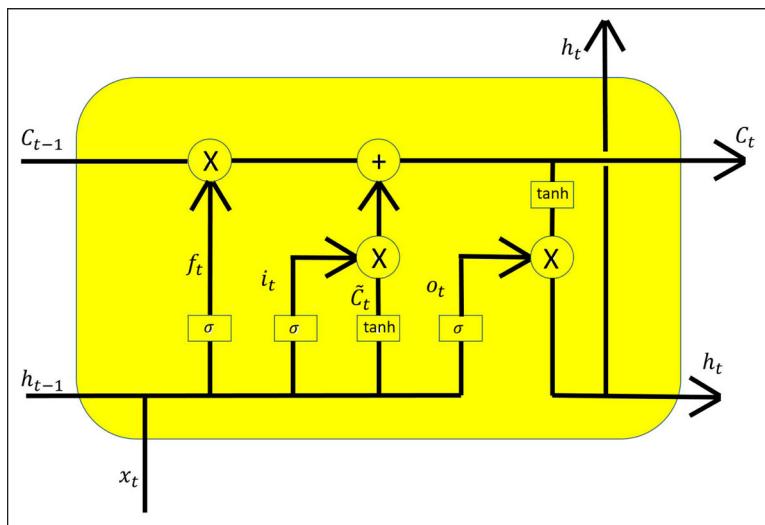


Figure A2. Architecture of an LSTM cell with various gates.

Appendix 2. CGANs: training and implementational details

The CGANs were trained with 96 feature maps each in both the generator and discriminator, with a batch size of 12. The noise vector to initialise the training was of size 100×1 . A binary cross entropy loss with ADAM optimiser was used for training both the networks in GAN. The learning rate was set at 2×10^{-5} , with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ being the optimisation parameters for ADAM.

A.3. Transpose convolution and resize convolution

Transpose convolutions are the traditional approach to upsampling used in CNN-based GANs. This operation can be thought of as the reverse of a standard convolution: Instead of sliding a kernel across a group of pixels to learn a mapping to fewer pixels, the kernel is trained to extrapolate individual pixels to a larger pixel group. The distance that the kernel slides each time is known as the stride. Deconvolution is sometimes mentioned as the same operation even though the two operations are not the same (deconvolution is, technically, the inverse of convolution). Since we are dealing with volumetric data, we utilise 3D transpose convolutions which use a cubic kernel. Furthermore, the stride defines how far the kernel translates each step, while the padding determines how many zero-value pixels are added to the input. See Figure A3 for a comparison of 3D and 2D transpose convolution. The use of transpose convolutions in the generator results in a common issue of GANs

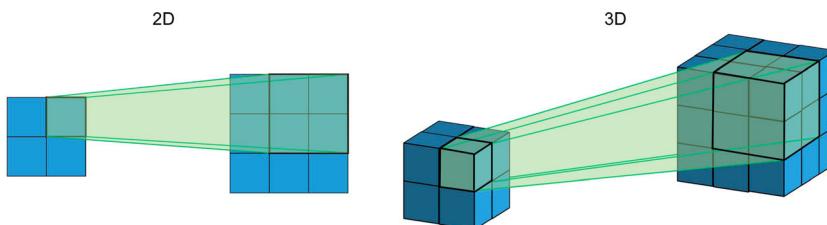


Figure A3. Representation of transpose convolution. In this case, an input size of 2 is up sampled to an output size of 3 with a kernel size of 2 (square kernel for 2D and cubic for 3D) and a stride of 1. 2D transpose convolution is shown on the left and volumetric transpose convolution is shown on the right.

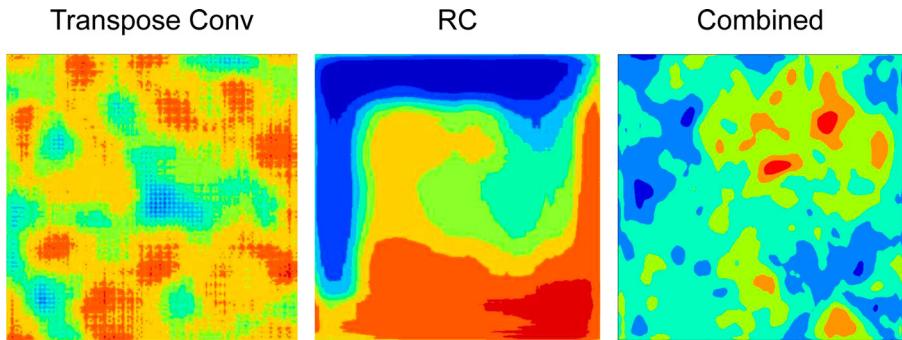


Figure A4. An illustration of using only transpose convolution (left), only resize convolution (middle), and the transpose-RC hybrid (right) on a 2D slice of the flow.

called checkerboard artefacting [73]. The artefacts are the result of overlapping transpose convolutions when upsampling the data. This typically occurs when the stride is less than the kernel size, especially when the kernel size is not divisible by the stride.

Odena et al. [73] provide an interesting solution to the checkerboard artefact problem, known as resize convolution (RC). RC involves interpolation followed by standard convolution. We found that trilinear interpolation worked best for our application whereas nearest-neighbour interpolation continued to result in some line artefacts. Trilinear interpolation consists of inserting zero padding in between values in the input tensor (to resize the sample to the desired dimensions) followed by averaging the values close to the padding to determine the new value of those indices. We also found that the generator does not learn when solely using RC. To determine if the generator network was learning, we suspend updates to the discriminator, and continue training the generator. With the discriminator no longer learning, the generator would have no competition and therefore should begin to learn to output samples that the discriminator will classify as ‘real’. However, if the discriminator still continued to identify the generated images as ‘fake’, then we can concur that the generator was not learning. This was indeed the case with the RC-only generator above.

Instead of choosing between the two approaches, we employ a hybrid strategy with transpose convolution in the first few layers of the generator and RC in the rest of the layers. This scheme proved successful as the transpose convolutional layers learned the underlying distribution of the data, while the RC layers learned to smooth out and eliminate the checkerboard artefacts. Figure A4 illustrates the results of using only one method of upsampling followed by combining both methods. The exact details of our implementation are described in the appendix.

A.4. Sampling from random distributions in GANS

GANs learn a probability distribution from a random noise vector that is provided as input. There are two instances where random sampling is important to consider in GANs. First, the input to the generator is a noise vector z of length 100. Traditionally, z would be sampled from the Gaussian distribution $N(1, 0)$. However Goodfellow’s GANs tutorial [74] mentions that if $z^{(2)}$ is Gaussian, prediction x is also conditionally Gaussian given $z^{(1)}$. Given that the training data is non-Gaussian, z is sampled from a uniform distribution in order to avoid Gaussian behaviour in the generated data. A uniform distribution has constant probability given by $P(x) = 1/(b - a)$ (the bounds, a and b were 0 and 1, respectively). Figure A5 graphically shows the differences in all three distributions.

In GANs, the discriminator has a tendency to become overconfident and outputs labels very close to 1 or 0 (as opposed to labels of 0.9 or 0.1). When this happens, the generator’s gradients begin to vanish and it can no longer meaningfully update its weights. One solution to this is to attempt to balance the networks by making the generator stronger or the discriminator weaker. Although this solution can be effective, we found it ultimately limited the potential of the GAN. Therefore, we implement a solution first mentioned by Salimans [35]: one sided label smoothing. Instead of using

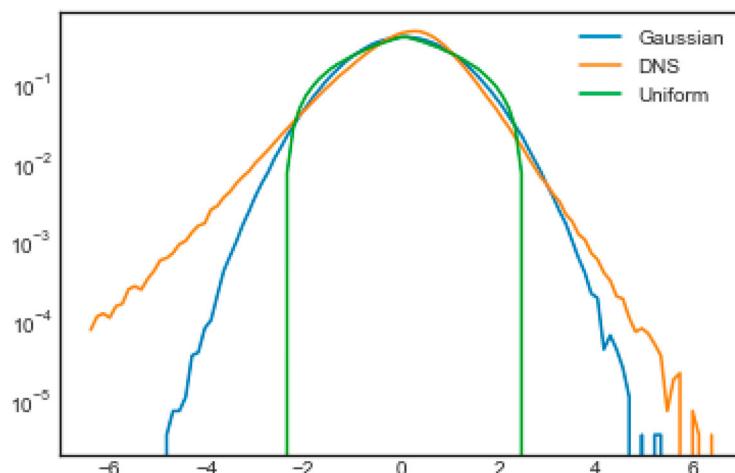


Figure A5. CGANs velocity gradient magnitude PDF comparison for different random vector initialisations.

1 and 0 as target labels, we added noise to the labels so that the real label would fall in the range [0.875, 1] and the fake label would fall in the range [0, 0.125]. This effectively decreased the discriminator's overconfidence so that it could still accurately classify the samples without compromising on the gradients it provides to the generator.

A.5. Cyclic learning rates

In order to improve training efficiency, we employed the technique of cyclic learning rates by Smith [75]. Varying the learning rate as the models train, allows them to converge faster and reach a lower loss value. Figure A6 shows how our learning rate varied as the models trained. We employed a triangular update policy, changing the learning rate in a piece-wise linear fashion. Hence, the learning rate fluctuates between minimum and maximum values at a rate determined by the number of

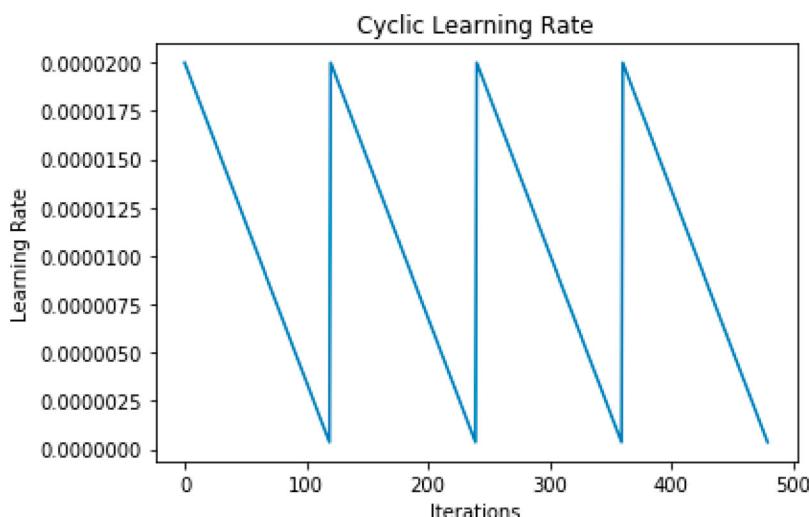


Figure A6. Cyclic learning rate: triangular update policy.

steps it takes to complete one full cycle. In this work, the min and max rates were set as 2×10^{-7} and 2×10^{-5} respectively. Changing the learning rate every iteration allowed us to forsake finding a perfect value for a constant learning rate. Although Ref. [75] detailed an excellent way to find the minimum and maximum values for a given classification model, GANs converge in a different manner that is not entirely clear from losses alone. Therefore, to select these values we found the minimum and maximum values at which the losses did not diverge, but also learned at an acceptable speed. Furthermore, the discriminator's learning rate is an order of magnitude less than the generator, in order to balance the networks' relative strength.

Appendix 3. Computational costs: CCLSTM vs. GANs

An import metric of comparison between the two approaches is their computational requirements of training the 3D turbulence models. The computational cost and memory requirements for neural networks are typically governed by the total number of trainable parameters required to 'learn' the dynamics. For GANs, the Generator network needed 130, 002, 821 parameters and Discriminator 108, 372, 769 parameters, i.e. a total of ≈ 238 million parameters. In contrast, the CCLSTM approach proposed in this work of two networks trained separately – the convolutional autoencoder (CAE) and the convolutional LSTM (CLSTM). The CAE needs a total of only 74,380 parameters to learn the compressed latent space for the flow, and the CLSTM network that trains on the latent space needs 307, 985 parameters, for a kernel size $\alpha = 3$ and sequence length $\omega = 3$. Therefore, the CC-LSTM approach only needs a combined 382, 365 parameters for spatial *and* temporal predictions, compared to GANs which does not account for temporal dynamics. We like to emphasise that the CC-LSTM needs ≈ 600 times fewer parameters than GANs for the same flow, while predicting large scale dynamics better. This superior increase in efficiency opens up CC-LSTM to larger datasets than GANs.