Embedding hard physical constraints in neural network coarse-graining of three-dimensional turbulence

Arvind T. Mohan*
Center for Nonlinear Studies Computational Physics and Methods Group, Los Alamos National Laboratory, Los Alamos 87545, United States

Nicholas Lubbers
Information Sciences Group, Los Alamos National Laboratory, Los Alamos 87545, United States

Misha Chertkov
Program in Applied Mathematics, University of Arizona, Tucson 85721, United States

Daniel Livescu
Computational Physics and Methods Group, Los Alamos National Laboratory, Los Alamos 87545, United States

(Received 17 February 2020; accepted 23 November 2022; published 31 January 2023)

In recent years, deep learning approaches have shown much promise in modeling complex systems in the physical sciences. A major challenge in deep learning of partial differential equations is enforcing physical constraints and boundary conditions. In this work, we propose a general framework to directly embed the notion of an incompressible fluid into convolutional neural networks, and apply this to coarse-graining of turbulent flow. These *physics-embedded neural networks* leverage interpretable strategies from numerical methods and computational fluid dynamics to enforce physical laws and boundary conditions by taking advantage the mathematical properties of the underlying equations. We demonstrate results on three-dimensional fully developed turbulence, showing that this technique drastically improves local conservation of mass, without sacrificing performance according to several other metrics characterizing the fluid flow.

DOI: 10.1103/PhysRevFluids.8.014604

I. INTRODUCTION

Several practical applications of computational fluid dynamics in engineering, earth, and aeronautical sciences are extremely high-dimensional and expensive to compute. Despite the increasing power of modern computing infrastructure, *cost-effective* and *rapid* prediction of fluid flow is still out of reach, and several research efforts have focused on developing cheaper alternatives to high-fidelity numerical simulations. Among these, data-driven approaches have become quite attractive in recent years, as they aim to leverage existing high-fidelity datasets to extract insight into the statistics and mathematical characteristics of the flow. This insight enables us to develop reduced-order models (ROMs), which are coarse-grained, low-dimensional approximations of their high-fidelity counterparts.

From the standpoint of ROM and machine learning, the size of the data has direct ramifications for engineering problems. For instance, the computer memory available for ROM in engineering

*Corresponding author: arvindm@lanl.gov

applications (such as feedback control) is minimal, making three-dimensional (3D) turbulence challenging to capture in these models. As a result, explicit data compression or coarse-graining has been a long-sought-after goal in data-driven ROMs. However, purely data-driven algorithms, such as neural networks, are devoid of physics and do not guarantee primary constraints such as mass conservation, leading to unphysical ROMs despite reducing data size [1]. Furthermore, boundary conditions (BCs) are not rigorously imposed, despite being a core component of modern computational science. Finally, a significant challenge is the construction of *nonintrusive ROMs* [2–4]. ROMs have to be nonintrusive when the source or numerical method used in the training dataset is unavailable or challenging to replicate, such as in the case of experimental data or simulation from commercial solvers [5]. This scenario is common in many applications, and the problem is more challenging since we have to rely only on the physics of the problem, not the numerical/experimental methodology. Therefore, a deep-learning nonintrusive ROM should constrain physics with minimal knowledge of the process that generated the training data. This work aims to accomplish nonintrusive learning of high-dimensional data in the context of 3D turbulence while simultaneously enforcing hard physics constraints and BCs rigorously.

A. Deep learning for high dimensional turbulence

A revolution is underway in computational physics with the promise of deep learning approaches in modeling unresolved physics, accurate coarse-graining and ROMs. This is particularly attractive for fluid flow [6–13] and, in particular, turbulence, where the curse of dimensionality coupled with the complexity of the Navier-Stokes equations have hindered much progress over the decades. While recent successes of neural networks (NNs) for turbulence ROMs have gained popularity, we maintain that the curse of dimensionality is as relevant as ever, even in the context of deep learning for physics. There are two key issues with learning high dimensional physics datasets: (1) The computational and memory limitations in employing enough training parameters (2) The black-box nature of NNs that do not guarantee physical laws such as constitutive equations and BCs. Thus, computational challenges combined with the physics-agnostic nature of NNs can significantly affect our confidence in the learned high-dimensional physics models.

A simple but important example is the continuity equation, which has several applications in physics. For incompressible flows, this equation becomes the divergence-free condition for the velocity field V:

$$\nabla \cdot V = 0 \tag{1}$$

Consider a standard NN, which is physics-agnostic, trained to model V. In the vast space of NN parameter sets, only a small subset of models will produce flows that are consistent with the continuity equation in Eq (1). Some efforts to model turbulence have taken this approach [6–8], using deep learning as a black box tool. However, this ignores the continuity equation as a fundamental physical law. A recent approach to incorporate it relies on penalizing the network in the loss function [9–13] to encourage solutions to obey Eq. (1) as well as known BC's. We call this approach a *soft constraint* for physics-informed modeling. The loss function encourages the network to find models which are near the physical law, while the overall network remains a black-box. Test examples beyond the scope of training may violate continuity or BC's; as the NN model architecture itself is still blind to the physics. Additionally, the weight of the soft constraint in the loss function becomes an additional hyperparameter that must be tuned during model selection [14–16]. While soft constraints have been popular due to their flexibility, they provide no guarantees due to lack of an *inductive bias* [17–20] in the model; i.e., a model can be expected to perform better if it has built-in knowledge concerning the admissible solution space of a problem. In other words, a model which structurally enforces physical laws is by definition more robust because it offers guarantees on network behavior for data both inside and outside of the training space. Recent work by Wang [12] demonstrated the importance of inductive bias in fluid mechanics machine learning, however, Eqn 1 is still implemented as a soft constraint. While useful, utilizing a soft constraint only approximately

realizes these hard physical laws through the difficult-to-interpret non-convex optimization used for training.

Owing to the demonstrated successes of these works for 1D/2D problems, we direct attention to model the full nonlinearity and complexity of turbulence, which is fundamentally a 3D phenomenon [21] There are significant computational bottlenecks in extending NNs to 3D turbulence datasets for practical applications. This is an acute problem for approaches like Physics Informed Neural Networks (PINNs) [9,22], which need expensive sampling of millions of points for nonchaotic 1D/2D equations. While there has been recent work in adapting NN architectures to learn 3D turbulence, they sacrifice explicit physical constraints [1,23–25] for computational efficiency, raising reliability concerns. A notable work by Kim *et al.* [26] constrains conservation of mass in 3D fluids via soft constraints for computer animation applications.

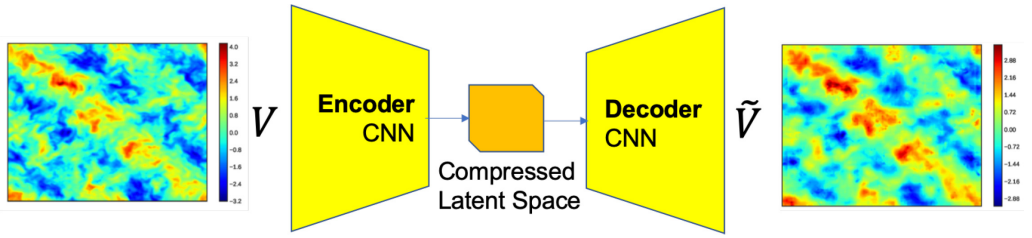
B. Embedding physical operators in convolutional neural networks

Convolutional neural networks (CNNs) are well positioned to ingest high dimensional datasets because they utilize *parameter sharing* [27], where a filtering kernel is convolved across the domain to learn the correlation structure in the data. Early layers in the CNN are sensitive to short-range correlations, and later layers build on this to learn long-range correlation as well. Because the same parameters are reused (shared) in convolutions across the spatial domain, CNNs need orders of magnitude fewer parameters than fully connected NNs. This is paramount for 2D and 3D flow problems, where the amount of data is exponential in the spatial dimension of the problem, and a fully connected approach thus requires exponentially more parameters. The reader is referred to Refs. [28], [27], and [29] for details of CNNs. Recent physics-informed machine learning has brought into focus the importance of the *structure* of CNN kernels, as explored in partial differential equations (PDE)-Net [30] and related works [31–33].

A growing application of CNNs for turbulence is the convolutional autoencoder (CAE) [34] for coarse-graining of high fidelity turbulence [1,23,35,36]. CAEs provide for compression of high-dimensional data like turbulence by using an *encoder* to map the input space into a *latent space* of lower dimensionality by interleaving coarse-graining with learnable convolution layers, and a *decoder* which learns to map this latent space back to the original data in a similar fashion. As this compression forms a natural basis for the construction of ROMs [1,11,37–39] of turbulent flow and other physical systems, this is a useful domain for research into enforcing hard physical constraints.

The divergence operator $\nabla\cdot$ represents local mass balance, and is a crucial operator in any model of fluid flow. A core aspect of embedding the divergence-free condition as a hard constraint in NN architectures is an accurate and unambiguous definition of an operator which is also amenable to the backpropagation algorithm used during training. Backpropagation through the physics operators ensures that the network is not "blind" and has intimate knowledge of the constraints through which it must make predictions. To this end, there are three major challenges: First, constructing spatial derivative operators (i.e. $\nabla\times$, $\nabla\cdot$, $\nabla^2\cdot$, etc.) that are compatible with the backpropagation algorithm for NN training. Second, the velocity fields are subject to BCs. Third, the realization of divergence and BC constraints ought to be directly imposed, rather than utilizing soft constraints; We adopt the philosophy of most PDE solvers, where conservation laws and BC constraints are *strictly enforced at all times*.

In this work, we attempt to address these challenges with a stronger approach which constrains the NN architecture to explore only the physically admissible solution space. We also require that such networks be designed for high-dimensional, fully turbulent flows for practical applicability. The primary motivation is that hard constraints can provide physics guarantees [40], since the NNs are no longer physics-agnostic and the architecture ensures the BCs are prescribed accurately. However, hard constraints are more challenging to implement; the case of embedding $\nabla\cdot V = 0$ requires the NN to understand operators such as curl and divergence, while concurrently enforcing BCs. In this work, we report first attempts on a general methodology to address these challenges in a CNN framework with strong inductive bias, as applied to 3D turbulence. Our approach does



(a) Unconstrained Convolutional Autoencoder with latent space mapping between V and coarse grained $\tilde{V}$, where encoder learns a parsimonious representation of the velocity, and the decoder learns a mapping to reconstruct $\tilde{V}$ from the latent space. (Slice of U-velocity component for illustration)

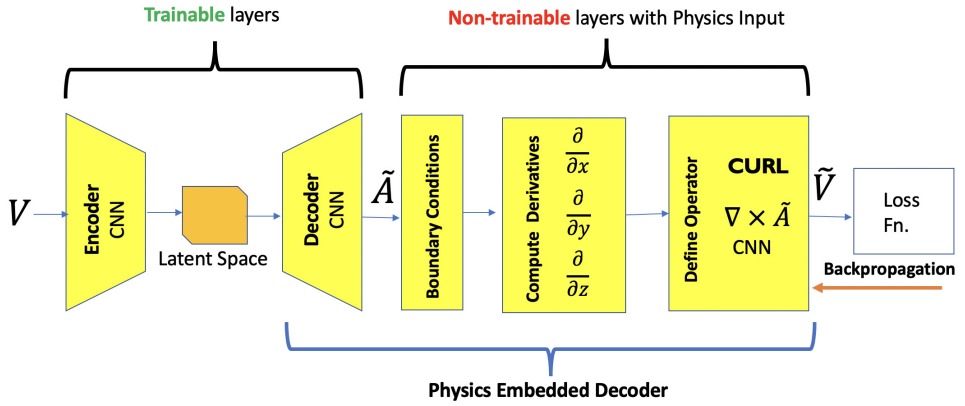


FIG. 1. (a) Unconstrained convolutional autoencoder with latent space mapping between V and coarse grained $\tilde{V}$, where encoder learns a parsimonious representation of the velocity, and the decoder learns a mapping to reconstruct $\tilde{V}$ from the latent space. (Slice of U-velocity component for illustration). (b) Physics embedded convolutional autoencoder with hard divergence-free constraints for coarse grained $\tilde{V}$. The conventional decoder from (a) is replaced with the physics-embedded decoder, which learns an intermediate potential $\tilde{A}$ that obeys the constraint $\nabla \times \tilde{A}$ which outputs a divergence-free coarse grained velocity field $\tilde{V}$ through nontrainable finite difference layers.

not modify the number of trainable parameters. Furthermore, it adds explainability by interpreting time-tested strategies from numerical methods and computational fluid dynamics (CFD) as specific instances of CNN kernels, and so can thus be used to design network architectures that include derivative expressions. This offers a way to improve model performance which is completely separate from hyperparameter search. In the next section, we describe our approach for embedding incompressibility into a CNN. In the third section, we describe our experiments and results in implementing this strategy. Finally, we offer conclusions and directions for future work.

II. PHYSICS EMBEDDED CNN ARCHITECTURE WITH HARD CONSTRAINTS

To demonstrate the aforementioned physics embeddings, we choose a CAE for coarse-graining of high fidelity turbulence. Figure 1(a) shows the standard, physics-agnostic CAE architecture, with the 3D velocity field training dataset as input V. The CAE encoder learns a compressed latent space, significantly smaller than V. The velocity field is reconstructed from the latent space by a CAE decoder that is learned simultaneously with the encoder. However, this data compression is accompanied by some loss of information, and therefore the reconstructed V is denoted as $\tilde{V}$. Therefore we can consider $\tilde{V}$ as a coarse-grained approximation of V. Since the CAE is

unconstrained, the challenge is to embed physical constraints to ensure that the CAE only learns a latent space such that $\nabla \cdot \tilde{V} = 0$.

Since embedding Eq. (1) requires computing derivatives on a field that resides on a discretized solution mesh, it is pragmatic to adopt strategies from well-known finite difference (FD)/finite volume (FV) numerical methods. These FD/FV algebraic discretizations can be analytically derived from Taylor series expansions around a fixed point [41,42]. These discretizations can further be expressed in matrix form called *stencils*. Upon examination, it is apparent that the CNN kernels are *structurally equivalent* to the FV stencils [30–32]. This simple, but powerful connection allows us to embed these stencils as CNN layers with fixed kernel weights to compute spatial derivatives. For details on this approach, see Appendix B. We employ a technique to enforce BCs in the CNN using the idea of *ghost cells* [43–45] from CFD, as outlined in Appendix C. This allows us to implement BCs to a given order of discretization accuracy. Both these techniques involve a close fusion of concepts from neural networks, numerical methods and CFD.

Having implemented spatial derivatives, we model the flow using a potential formulation [46,47] based on the Helmholtz decomposition of the velocity field, with vector potential A and scalar potential ψ:

$$V = \nabla \times A + \nabla \psi. \tag{2}$$

The divergence constraint annihilates all terms associated with the vector potential A, giving rise to a Laplace constraint only on the scalar potential: $\nabla^2 \psi = 0$. For steady boundary conditions, the scalar potential corresponds to a steady background flow, which can be subtracted from the data so that the learning problem is restricted to the turbulent fluctuations. In particular, for data analyzed here, the boundary conditions are periodic, so $\psi = 0$ is a valid solution. The key idea is as follows: Instead of only predicting a velocity field, we choose to make an intermediate prediction for vector potential $\tilde{A}$, while framing the final network prediction $\tilde{V}$ in the target velocity space via the curl, implemented as a numerical stencil:

$$V = \nabla \times A. \tag{3}$$

Then, predictions $\tilde{V}$ will *automatically* obey Eq. (1) up to the accuracy of the stencil.

Figure 1(b) shows the autoencoder with a physics-embedded CNN AutoEncoder (PhyCAE) where this strategy is implemented, in addition to those in Appendix B. The CAE encoder learns a latent space from V, from which the decoder constructs $\tilde{A}$. While the flow dataset itself does not directly contain A (and A itself can only be defined up to gauge transformations), the network can implicitly learn $\tilde{A}$; we constrain the learning by requiring that the curl of the decoder prediction $\tilde{A}$ be equal to $\tilde{V}$. The physics is thus embedded in the modified decoder, which consists of a CNN layer enforcing BCs with ghost cells (Appendix C), followed by another CNN layer which computes all the spatial derivatives necessary to define a curl operator (Appendix B). The last CNN layer uses these to compute the curl operator on the $\tilde{A}$ field. Therefore, all layers after the decoder CNN in the PhyCAE are nontrainable and fully explainable, as they merely construct Eq. (3) with numerical methods.

III. RESULTS

A. Training

To illustrate the performance of the physics embedded autoencoders, we train two cases: The standard CAE with zero padding in Fig. 1(a), and the PhyCAE in Fig. 1(b). The dataset is high-fidelity direct numerical simulation (DNS) of a 3D homogeneous isotropic turbulence (HIT), which is described in Appendix A and Ref. [48].

The CAE architecture has three layer encoder-decoder with an ADAM optimizer and L2 loss, with six filters at each level, and was evaluated for turbulence by Mohan [1]. We intentionally use fewer filters to avoid over-training and illustrate the effects of physics input in the NNs. In this work, we define the compression ratio as the ratio between the size of a raw flow snapshot tensor to its

compressed version, as it focuses on reduction in the amount bytes needed to store and compute the snapshot. In the present work, the size of a single flow snapshot with three flow variables is (3×128^3) and that of the latent space (6×15^3), as it contains six latent channels. The compression ratio is therefore $\approx$310. The PhyCAE architecture comprises of the CAE layers above with the same hyperparameters, but with the addition of the physics embedded layers in the decoder. We emphasize again that these layers add no extra learnable parameters to the network. Since the dataset is statistically stationary with a total of 12 eddy turnover times, we train both networks on 0–0.75 eddy times. The test dataset is from 4–4.75 eddy times, so that we test if the network has learned the statistical behavior of the fluid correlations at future time. We also note that all calculations here are performed with single precision, hence the relative numerical precision is $\approx10^{-7}$.

B. Effect of hard constraints on training

In this section, we investigate how stringently CAE and PhyCAE adhere to the $\nabla \cdot V$ constraint, and explore the characteristics of PhyCAE from the perspective of accuracy, computational efficiency, training speed, and reliability.

NNs make predictions every epoch, and if the training loss converges to a minima with (typically several thousand) more epochs, we expect the predictions to have "learned" some physical laws. One of the key expectations from any physics-embedded hard constraint, is that the network must be cognizant of these physical laws *from the very first epoch* and the predictions never deviate far from the imposed constraints, i.e., the stay on the physical subspace. In other words, while improved optimization methods (adaptive regularization, optimizers, optimal hyperparameters, minimization tricks etc.) can assist the physics embeddings in finding minima with the desired physical laws, the hard constraints should ensure that the network predictions stay in the physically constrained subspace *by design* and not solely by learning. As a result, we should expect PhyCAE to find minima faster and be more consistent with the physics than the CAE. To quantify how well the constraint is realized, we measure the total absolute divergence (TAD) across each example averaged over the examples, given by the formula $\sum |\nabla \cdot \tilde{V}|$. A model which perfectly enforces incompressible flow minimizes this quantity. However, the TAD is not zero even in the PhyCAE due to the second-order discretization stencils used to apply derivative operators and the limitations of single-precision floating-point arithmetic.

Figure 2 shows the TAD on the training data as a function of the network training time for both the CAE and PhyCAE. For CAE, we see a spike in the divergence as high as $\sim10^{-1}$, and approaches to a final value of $\sim10^{-2}$, although, in principle, it might increase with additional training. The TAD for the PhyCAE starts at $\sim10^{-2}$ and trends downward during training, even oscillating near numerical zero, and typically between 10^{-4} and 10^{-5}. We remark that much like any PDE solver, the discretization errors affect the accuracy of the hard constraint, such as the second-order scheme used in this work. However, such deviations are to be expected when constructing nonintrusive ROMs, since we are (a) using a ROM numerical method that is not the same as that used to generate the training data; this in turn affects (b) stochastic optimization solvers in neural network backpropagation, which is a nondeterministic process affected by several numerical factors that are hard to control.

Despite these intrinsic challenges, we see that the robustness of our approach shines through: The *best-case* for the CAE divergence is comparable to the *worst-case* for the PhyCAE, and after training, the PhyCAE performs more than two orders of magnitude better with respect to TAD. We have likewise computed the TAD after training on the test dataset after training. In this case, the TAD for the PhyCAE is $\sim10^{-5}$, while CAE is $\sim10^{-2}$, further emphasizing the generality and robustness of our physics embeddings. The physics-aware inductive bias of the PhyCAE allows it to perform far better than the CAE, while training with identical hyperparameters and number of trainable parameters. Due to the interpretable hard-constraint approach of the PhyCAE, this could be further decreased by improving the spatial discretization method in Eq. (B1) from a second- to a higher-order scheme. This extension is straightforward since the CNN allows for kernels of larger

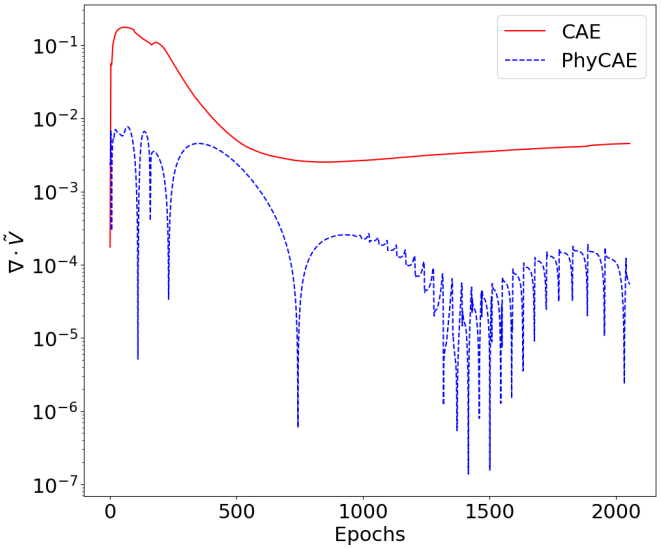


FIG. 2. Variation of total absolute divergence $\sum |\nabla \cdot \tilde{V}|$ with training epochs for CAE vs PhyCAE. Final divergence: $\approx 10^{-5}$ for PhyCAE vs $\approx 10^{-2}$ for CAE, showing that the constraint is always strongly enforced in comparison, despite oscillations arising from different numerical schemes being used as a nonintrusive ROM.

sizes produced by higher order numerical schemes. This would require a corresponding change in the number of ghost cells, which can be implemented as outlined in Eq. (C1) in Appendix C.

C. Turbulence diagnostics

We now briefly describe three important tests of 3D turbulence which are used as diagnostic metrics for the accuracy of the flow predicted by the trained model. While these metrics do not explicitly address the divergence of the velocity, they target specific properties of turbulence and are important to provide confidence in the accuracy of the predicted model.

1. Kinetic 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. Self-similarity hypothesis results in the expectation that within the inertial range, the energy spectrum, $E(k)$, scales as $k^{-5/3}$.

Figure 3 shows the energy spectrum of the coarse grained $\tilde{V}$ predicted by the CAE and PhyCAE compared with the ground truth value from the DNS test data V. The results show excellent large scale (low wave numbers) and inertial range accuracy by the PhyCAE, very similar to that of CAE. Most of the discrepancies are localized at the small scales (high wave numbers), due to the information loss that occurs during coarse graining. However, since most practical applications of ROMs focus only on large/inertial scales, we find these discrepancies acceptable.

2. PDF of longitudinal velocity gradient

Small scale turbulence is highly intermittent and departs from self-similar scalings. One measure of this departure can be seen in the PDF of the longitudinal velocity gradient, which presents

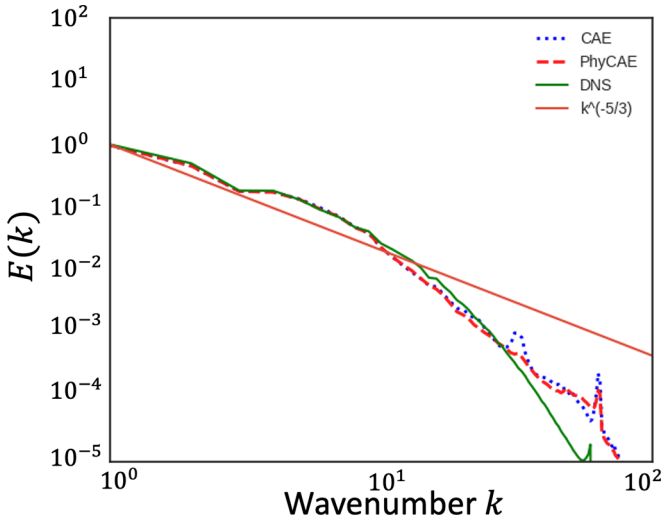


FIG. 3. Energy spectra of $\tilde{V}$ for CAE and PhyCAE vs DNS V. The spectra show the effect of coarse-graining is restricted to highest wave numbers and some regions of the inertial range. PhyCAE obtains comparable statistics with the added benefit of the hard constraint being strictly enforced, without additional parameters to train.

non-Gaussian tails and nonzero skewness [49]. The skewness is related to the 3D structure of turbulence, which is further discussed in the next section.

The longitudinal velocity gradient PDFs are computed for both $\tilde{V}$ and V in Fig. 4. Once again, we see excellent matches between predicted PhyCAE and DNS velocities, with minor discrepancies in the tails, which are likely a manifestation of the small scale errors. CAE is quite close to PhyCAE in observed accuracy, with the latter being slightly more accurate in the tails.

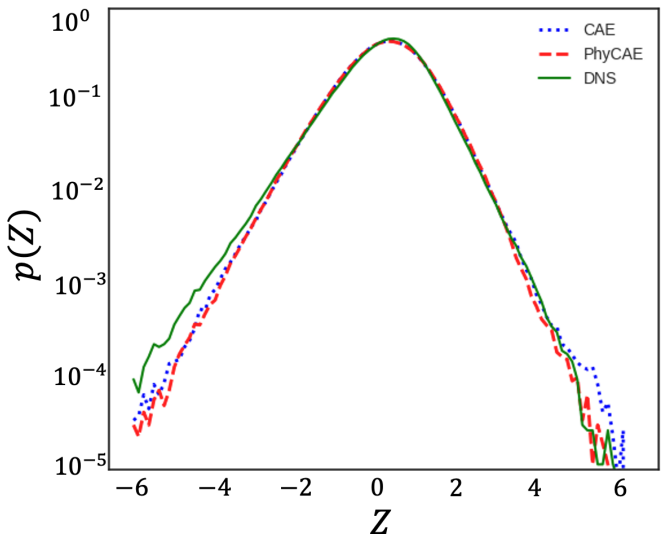


FIG. 4. PDFs of longitudinal velocity gradient of $\tilde{V}$ for CAE and PhyCAE vs DNS V. The effect of coarse-graining is seen at the tails of the PDF, where discrepancies indicate small-scale feature loss.

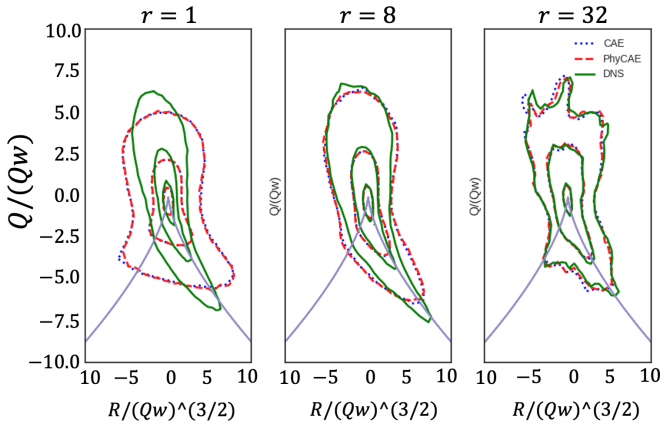


FIG. 5. $Q-R$ plane PDFs of $\tilde{V}$ for CAE and PhyCAE vs DNS V at different scales, where r is measured in grid spaces, such that $r=1$ corresponds to viscous scales, $r=8$ corresponds to inertial range, and $r=32$ corresponds to large scales. The axes are normalized by $Q_w = Wij\,Wij/2$, where W is the rotation tensor. Coarse-graining causes loss in accuracy primarily in the small scales ($r=1$) and some inertial scales ($r=8$), with only minor deviations in the large scales ($r=32$), which are important for ROMs.

3. Statistics of coarse-grained velocity gradient: $Q-R$ plane

To further address the 3D turbulence structure, we utilize isolines of probability in the $Q-R$ plane, which expresses intimate features of the turbulent flow topology, having a nontrivial shape documented in literature [50,51]. Here, Q and R are the second and third invariants of the anisotropic part of the filtered, at scale r, velocity gradient tensor. 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 quadrants, 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 observation scale r. Here, we study the $Q-R$ plane filtered at different scales r, to examine large ($r=32$), inertial ($r=8$), and small scale ($r=1$) behaviors [52]. This allows us to selectively analyze the accuracy of our predictions at different scales, since we are interested in modeling primarily the large and inertial ranges.

The $Q-R$ plane PDFs in Fig. 5 show a clear picture of the PhyCAE predictive capability. Both the networks show that large scale flow topology is captured extremely well, with inertial scales having minor, but acceptable, discrepancies. The small scale error is quite significant, as seen from the energy spectra and longitudinal velocity PDF. Overall, the $Q-R$ plane shows that we are indeed capable of predicting coarse-grained 3D fields that have excellent large scale resolution. We note that the unconstrained CAE also produces results of comparable accuracy, but lacks in conserving continuity. On the other hand, PhyCAE shows excellent predictive capability while also closely adhering to the divergence-free criterion as outlined in the previous section.

4. Baseline comparison for coarse-graining accuracy

In order to provide context for the coarse-graining capabilities provided by PhyCAE, we create a baseline comparison with filtering performed with Fourier bases. With the PhyCAE, each compressed flow snapshot, i.e., the latent space, has $15^3 \times 6 = 20250$ floating point values, and it encapsulates all three velocity components. For Fourier based filtering, we employ a sharp cutoff for wave numbers above the cutoff value. Neural network autoencoders with nonlinear activation functions are considered as a nonlinear modal decomposition techniques compared to linear methods [53] like Fourier decomposition. Therefore, a direct comparison with the Fourier

basis is not feasible. Instead, we study this from the lens of coarse graining, where the size (in number of floating point values) of the coarse-grained snapshot matrix relative to the accuracy is the metric of interest. We compare the two methods by keeping the size of the latent space consistent. Considering the three velocity components as independent, a cut-off wave number $k_{\text{cutoff}} = 11.5$, would lead to a total of 20469 floating point values to store the three components in the complex space (where a complex number was counted as two floating point values), which is close to the size of autoencoder latent space. While this comparison would make sense in general, only two velocities are independent in the particular case of the zero velocity divergence constraint. Thus, the third component can be found from the divergence condition. Accounting for this, a cut-off wave number $k_{\text{cutoff}} = 13.2$ would lead to a total of 20646 floating point values storing the two independent velocity components in the complex space.

We now compare the L2 reconstruction error between the Fourier filtering and PhyCAE reconstructions, since we use it as a training metric in the neural networks. The Fourier filtering has an L2 error of 701.688, while the PhyCAE neural network has an error of 1476.886. The lower L2 error in the Fourier filtering is a characteristic of how compression is accomplished in each approach. The Fourier modes perfectly capture the large scales by retaining the corresponding basis, while completely ignoring the other scales beyond the cutoff wave number. In contrast, the PhyCAE approach aims to model as many scales as possible, with higher errors in the smaller scales than the larger scales, as seen in Fig. 5. Therefore, the Fourier approach accomplishes compression by explicit truncation of the original flow, while the PhyCAE approach accomplishes the same by explicit modeling of the flow. Since L2 error is an averaged quantity over all scales of the flow, even minor discrepancies in the larger scales can lead to massive fluctuations in L2 error. Therefore, PhyCAE does not outperform Fourier filtering in compression, concerning L2 reconstruction, despite showing excellent statistical properties.

IV. CONCLUSION

Our work introduces and demonstrates the effectiveness of a simple and interpretable method of enforcing incompressibility of a fluid flow within CNNs. Since this is enforced as a structural aspect of the CNN, the method is a hard constraint rather than a soft constraint, and so yields no additional hyperparameters to tune. Another useful consequence is that the constraint is enforced through the training procedure. This is accomplished by utilizing static, nontrainable layers for spatial derivatives, whose kernels are stencils that can be derived using the standard discretizations. Likewise, we implement boundary conditions by modifying the padding operation of a CNN. Using these tools, we train the decoder to output a vector potential that describes perturbations around a background flow. We demonstrate this scheme for a CAE architecture; however, it is generally applicable to any task where a CNN should output an incompressible velocity field, for example, flow forecasting [1,37] or generative modeling of flows. A comparison with Fourier-based filtering shows that the PhyCAE approach is able to retain more scales in the turbulence spectra at the same degree of compression, despite minor discrepancies.

Another interesting feature of this scheme is the gauge ambiguity of the vector potential $\tilde{A}$. In traditional vector potential approaches to PDEs, a gauge invariance needs to be resolved by fixing a gauge in which computations take place. Our approach did not specify a gauge, and future work might explore the effective gauge conditions learned by the model and characterize them in relation to the classical body of work on gauge fixing. Problems involving incompressible fluids are common, but further work could seek to extend the context of this strategy. Thus, future work could focus on addressing more general constraints of the form

$$L(V) = 0, \tag{4}$$

for more general differential operators L and physical fields V, by enforcing them as CNN layers; work on soft constraints has identified this as a general target for physics-informed machine learning [9,22], and we concur with this direction.

While this work is directly applicable to uniform and structured meshes, some limitations exist in its current form. For instance, the CNN architecture implicitly requires a structured mesh, which may not be possible in several complex geometries, such as flight vehicles. A valuable avenue of research would be an extension to unstructured meshes, possibly exploiting recent developments in GraphCNNs [54]. Furthermore, additional research is needed to make CNNs efficiently compress 3D datasets, since dimensionality reduction with CNN kernels requires loading the entire dataset into GPU memory. This can be computationally prohibitive for large CFD simulations of even a few terabytes in size, which is still comparatively small in the era of exascale computing. Another direction of future work is to explore the accuracy and stability of this approach with different numerical schemes used as hard constraint embeddings, especially in the context of nonintrusive ROMs. In conclusion, although much work needs to be done, research addressing such problems with hard constraints, i.e., inductive bias through structural aspects of model architecture, would make ROMs of many physical systems more robust, interpretable, and practically applicable.

ACKNOWLEDGMENTS

The authors thank D. Daniel for generating the DNS data and also thank J. Miller, V. Gyrya, P. Mitra, and S. Arunajatesan for useful 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 No. 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.

APPENDIX A: 3D HOMOGENEOUS ISOTROPIC TURBULENCE DATASET

The dataset consists of a 3D DNS of homogeneous, isotropic turbulence, in a box of size 128^3. We denote this dataset as HIT for the remainder of this work. We provide a brief overview of the simulation and its physics in this section, and a detailed discussion can be found in Daniel *et al.* [48]. The ScalarHIT dataset is obtained using the incompressible version of the CFDNS [48,55] code, which uses a classical pseudospectral algorithm. We solve the incompressible Navier-Stokes equations:

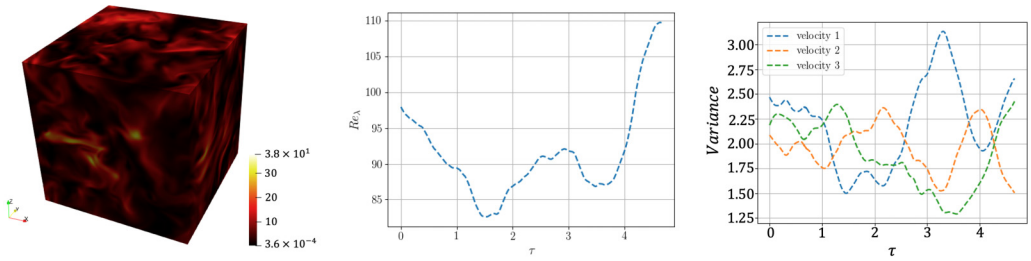
$$\partial_{x_i} v_i = 0, \quad \partial_t v_i + v_j \partial_{x_j} v_i = -\frac{1}{\rho}\partial_{x_i} p + \nu \Delta v_i + f_i^v,$$

where f^v is a low band forcing, restricted to small wave numbers $k < 1.5$ [1]. The 128^3 pseudospectral simulations are dealiased using a combination of phase-shifting and truncation to achieve a maximum resolved wave number of $k_{\max} = \sqrt{2}/3 \times 128 \sim 60$.

For illustration, Fig. 6(a) shows the turbulent kinetic energy at a time instant. Figure 6(b) shows the variation in the Taylor-microscale-based Reynolds number with the eddy turnover time, which characterizes the large turbulence scales. Finally, the variances in all 3 velocity components are shown in Fig. 6(c). Based on the sampling rate, each eddy turnover time τ consists of 33 snapshots. The training dataset uses 22 snapshots $\approx 0 - 0.75\tau$ and test dataset also consists of 22 snapshots in $\approx 4 - 4.75\tau$.

APPENDIX B: SPATIAL DERIVATIVE COMPUTATION IN CNN KERNELS

In deep learning, CNNs are used to learn the spatial features with a (symmetric or nonsymmetric) kernel f as a convolution operation throughout the domain of interest g at a layer n. The n^{th} CNN layer thus computes a field y as $y_n = f * g_{n-1}$, where g_{n-1} is the domain computed by the $(n-1)^{th}$ layer. Therefore, $y_n = g_n$ at layer $n+1$, such that $y_{n+1} = f * g_n$. The kernel translation is also an important hyperparameter, called *striding*, that can be performed for every point in the mesh



(a) Instantaneous turbulent kinetic energy

(b) Reynolds number (based on Taylor microscale)

(c) Individual velocity variances

FIG. 6. Representative statistics of the DNS dataset of homogeneous isotropic turbulence.

(1-step), or by skipping over a two or more points every time the kernel is translated (2-step and higher), as shown in the illustration in Fig. 7. Subsequently, these kernel weights are iteratively learned by backpropagation and gradient-based optimization.

Since we intend to compute derivatives on a field that resides on a discretized solution mesh, it is pragmatic to adopt strategies from well-known FD/FV numerical methods to compute derivatives, which can be analytically derived from Taylor series expansions around a fixed point [41,42]. These forward, backward, upwind or central difference discrete numerical approximations of a continuous derivative can be approximated to a desired m^{th} order [56,57] of accuracy. Consider a variable of interest ϕ that resides on a 3D mesh. A standard second-order central difference FV scheme for partial derivatives is shown in Eq. (B1). Its coefficients can be expressed as a matrix, called a numerical *stencil* as shown in Eq. (B2):

$$\begin{aligned}
\frac{\partial \phi}{\partial x} &= \frac{\phi(x+\delta x)-\phi(x-\delta x)}{2\delta x} + O(\delta x)^2,\\
\frac{\partial \phi}{\partial y} &= \frac{\phi(y+\delta y)-\phi(y-\delta y)}{2\delta y} + O(\delta y)^2,\\
\frac{\partial \phi}{\partial z} &= \frac{\phi(z+\delta z)-\phi(z-\delta z)}{2\delta z} + O(\delta z)^2,
\end{aligned} \tag{B1}$$

$$\begin{aligned}
\frac{\partial \phi}{\partial x} &= \begin{bmatrix} -\frac{1}{2\delta x} & 0 & \frac{1}{2\delta x} \\ -\frac{1}{2\delta x} & 0 & \frac{1}{2\delta x} \\ -\frac{1}{2\delta x} & 0 & \frac{1}{2\delta x} \end{bmatrix},\\
\frac{\partial \phi}{\partial y} &= \begin{bmatrix} -\frac{1}{2\delta y} & -\frac{1}{2\delta y} & -\frac{1}{2\delta y} \\ 0 & 0 & 0 \\ \frac{1}{2\delta y} & \frac{1}{2\delta y} & \frac{1}{2\delta y} \end{bmatrix},\\
\frac{\partial \phi}{\partial z} &= \begin{bmatrix} -\frac{1}{2\delta z} & 0 & \frac{1}{2\delta z} \\ -\frac{1}{2\delta z} & 0 & \frac{1}{2\delta z} \\ -\frac{1}{2\delta z} & 0 & \frac{1}{2\delta z} \end{bmatrix}.
\end{aligned} \tag{B2}$$

Upon examination, it is apparent that the CNN kernels are *structurally equivalent* to the FV stencils [30–32]. Therefore, FV stencils are essentially CNN kernels with fixed, nontrainable weights that compute a derivative to the desired order of accuracy. Furthermore, the CNN kernel and numerical stencil operations are *mathematically identical* in the context of 1-step striding. Similar to the CNN convolution operator above, domain g is the numerical mesh and f is the FV kernel. The

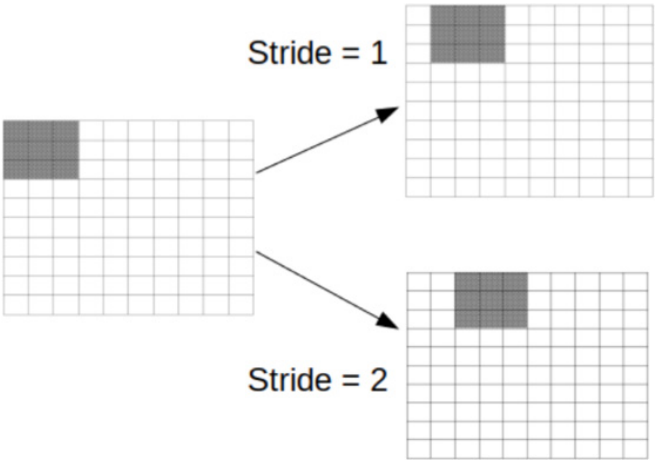


FIG. 7. Illustration of 1-step and 2-step striding mechanisms in convolutional neural networks for dimensionality reduction.

derivative y' is then computed as $y' = f * g$, at any layer n. Subsequent layers can be constructed to compute higher-order derivatives with appropriate stencil kernels.

This simple, but powerful, connection allows us to embed these stencils as CNN layers with fixed kernel weights to compute our derivatives of interest. Furthermore, we can explicitly define the order of accuracy in derivative computation by leveraging higher order numerical schemes. This allows us to trade off compute costs and accuracy while simultaneously being *interpretable*.

APPENDIX C: ENFORCING PERIODIC BOUNDARY CONDITIONS

Boundary conditions are a critical physics component that any computational approach, traditional PDE solvers or deep learning based, must rigorously impose in order to preserve the physics of the flow. The HIT flow has spatially periodic BCs in all three directions. In order to perform realistic physics embedding in CNNs, we present here a method to enforce BCs in CNNs, to a desired order of discretization accuracy.

Figure 8 shows the CNN kernel defined by a numerical stencil as described in Appendix B. The $(3\times3\times3)$ kernel performs convolution on the mesh, with 1-step striding. As a result, the outermost column/row of cells in the domain are forfeited. Due to the loss of boundary cells, the kernel is not able to accurately compute the derivative with the BCs, which can cause significant inaccuracies in the solution. In the machine learning community, a popular fix is to recover the original dimensionality by "padding" these dimensions by a constant valued scalar (typically zero) or replicate with the value of adjacent cells.

However, this is a well-known issue in the CFD community, and is more pronounced when using higher-order numerical stencils [43] with kernel sizes of $(5\times5\times5)$ or $(7\times7\times7)$. A standard approach to resolve this discrepancy while simultaneously satisfying the BCs, is by employing *ghost cells* [43–45]. Ghost cells are "virtual" cells which are defined at mesh boundaries, so that a derivative of the desired order consistent with the numerical stencil can be computed.

In the solution domain illustrated in Fig. 8, x, y, and z have dimensions of $N+1$, $M+1$ and $L+1$, respectively, which reduce to N, M and L due to the $(3\times3\times3)$ convolution kernel. Therefore, i^{th} direction now ranges from $0 \rightarrow M$, the j^{th} from $0 \rightarrow N$ and k^{th} direction from $0 \rightarrow L$. The k^{th} direction in the 3D domain is not shown here for convenience. Periodic BCs imply the flow leaving the domain in one direction enter the domain in the opposite direction. Ghost cells $(N+1)$ are now

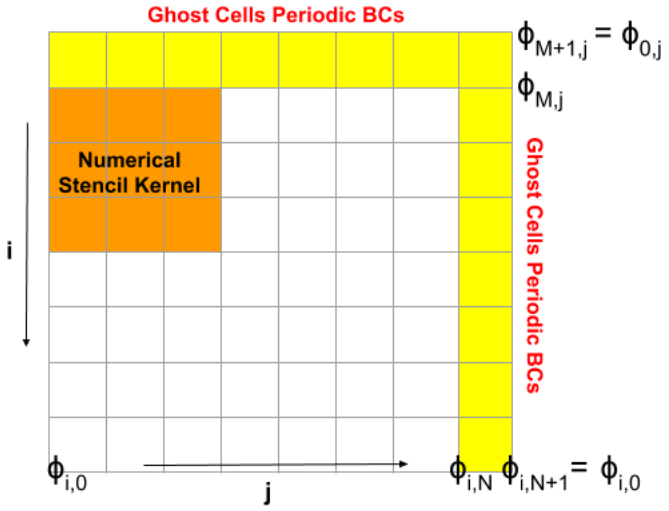


FIG. 8. CNN kernel in the physics embedded decoder in Fig. 1(b) computing numerical derivative in interior nodes (also shown periodic BCs prescribed as ghost cells).

created to mimic this behavior at the boundaries, shown as the yellow cells in Fig. 8. The solution at $(N+1)$ is set as $\phi_{i,N+1,k} = \phi_{i,0,k}$, which exactly satisfies the periodic BC constraint. Likewise in the i^{th} axis, $\phi_{M+1,j,k} = \phi_{0,j,k}$ and $\phi_{i,j,L+1} = \phi_{i,j,0}$ are the BCs in the k^{th} axis.

Figure 9 shows the net effect of the ghost cells, with the CNN kernel computing derivatives with BCs. It is important to note that the ghost cells are flexible for higher-order numerical schemes as well, by adding more cells to account for the larger kernel. For instance, in the j^{th} axis we can add

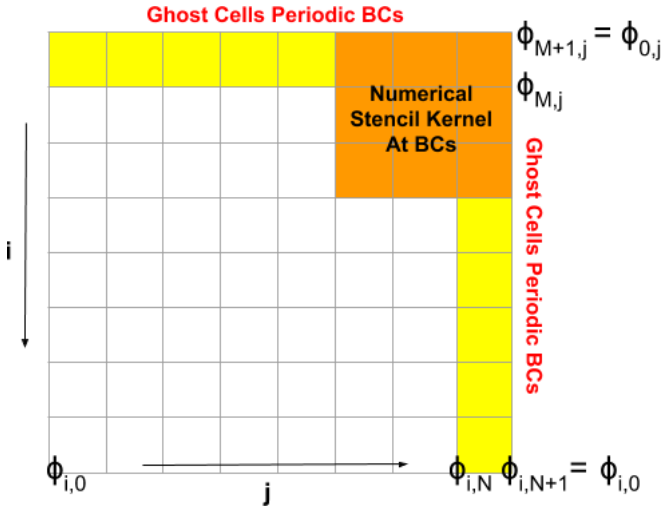


FIG. 9. CNN kernel in the physics embedded decoder in Fig. 1(b) computing numerical derivative at ghost cells boundary nodes with periodic BCs.

three columns of ghost cells with periodic BCs such that

$$\begin{aligned}\phi_{i,N+1,k} &= \phi_{i,0,k},\\ \phi_{i,N+2,k} &= \phi_{i,1,k},\\ \phi_{i,N+3,k} &= \phi_{i,2,k}.\end{aligned} \tag{C1}$$

We can thus specify BCs via ghost cells by developing a custom padding based on the desired order of numerical accuracy.

[1] A. T. Mohan, D. Tretiak, M. Chertkov, and D. Livescu Spatio-temporal deep learning models of 3D turbulence with physics informed diagnostics. J. Turbul. **21**, 484 (2020).

[2] D. Xiao, F. Fang, C. Pain, and G. Hu, Non-intrusive reduced-order modelling of the Navier-Stokes equations based on RBF Interpolation, Int. J. Numer. Methods Fluids **79**, 580 (2015).

[3] J. S. Hesthaven and S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. **363**, 55 (2018).

[4] D. Xiao, C. Heaney, F. Fang, L. Mottet, R. Hu, D. Bistrian, E. Aristodemou, I. Navon, and C. Pain, A domain decomposition non-intrusive reduced order model for turbulent flows, Comput. Fluids **182**, 15 (2019).

[5] J. Yu, C. Yan, and M. Guo, Non-intrusive reduced-order modeling for fluid problems: A brief review, Proc. Inst. Mech. Eng. Part G: J. Aerosp. Enging. **233**, 5896 (2019).

[6] K. Fukami, K. Fukagata, and K. Taira, Super-resolution reconstruction of turbulent flows with machine learning, J. Fluid Mech. **870**, 106 (2019).

[7] K. Fukami, Y. Nabae, K. Kawai, and K. Fukagata, Synthetic turbulent inflow generator using machine learning, Phys. Rev. Fluids **4**, 064603 (2019).

[8] J. Kim and C. Lee, Prediction of turbulent heat transfer using convolutional neural networks, J. Fluid Mech. **882**, A18 (2020).

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

[10] J.-L. Wu, K. Kashinath, A. Albert, D. Chirila, Prabhat *et al.*, Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems, J. Comput. Phys. **406**, 109209 (2020).

[11] N. B. Erichson, M. Muehlebach, and M. W. Mahoney, Physics-informed autoencoders for lyapunov-stable fluid flow prediction, arXiv:1905.10866 (2019).

[12] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu, Towards physics-informed deep learning for turbulent flow prediction, arXiv:1911.08655 (2019).

[13] B. Lusch, J. N. Kutz, and S. L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nat. Commun. **9**, 4950 (2018).

[14] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in *Neural Networks: Tricks of the Trade* (Springer, Berlin, 2012), pp. 437–478.

[15] L. N. Smith, Cyclical learning rates for training neural networks, in *Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE, New York, 2017), pp. 464–472.

[16] T. M. Breuel, The effects of hyperparameters on SGD training of neural networks, arXiv:1508.02788 (2015).

[17] D. F. Gordon and M. Desjardins, Evaluation and selection of biases in machine learning, Springer Mach. Learn. **20**, 5 (1995).

[18] T. M. Mitchell, *The Need for Biases in Learning Generalizations*, Rutgers Computer Science Tech report (Rutgers, New Jersey, 1980).

[19] N. Cohen and A. Shashua, Inductive bias of deep convolutional networks through pooling geometry, arXiv:1605.06743 (2016).

[20] A. Gaier and D. Ha, Weight agnostic neural networks, arXiv:1906.04358 (2019).
[21] H. Tennekes, J. L. Lumley, J. L. Lumley *et al.*, *A First Course in Turbulence* (MIT Press, Cambridge, MA, 1972).
[22] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, arXiv:1907.04502 (2019).
[23] R. King, O. Hennigh, A. Mohan, and M. Chertkov, From deep to physics-informed learning of turbulence: Diagnostics, arXiv:1810.07785 (2018).
[24] A. T. Mohan and D. V. Gaitonde, A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks, arXiv:1804.09269 (2018).
[25] S. Wiewel, M. Becher, and N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, in *Computer Graphics Forum* (Wiley Online Library, Hoboken, NJ, 2019), Vol. 38, pp. 71–82.
[26] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, Deep fluids: A generative network for parameterized fluid simulations, in *Computer Graphics Forum* (Wiley Online Library, Hoboken, NJ, 2019), Vol. 38, pp. 59–70.
[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, 2016).
[28] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature (London) **521**, 436 (2015).
[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (Lake Tahoe, 2012), pp. 1097–1105.
[30] Z. Long, Y. Lu, X. Ma, and B. Dong, PDE-net: Learning PDEs from data, arXiv:1710.09668 (2017).
[31] Z. Long, Y. Lu, and B. Dong, Pde-net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. **399**, 108925 (2019).
[32] B. Dong, Q. Jiang, and Z. Shen, Image restoration: Wavelet frame shrinkage, nonlinear evolution PDEs, and beyond, Multiscale Model. Simul. **15**, 606 (2017).
[33] J.-F. Cai, B. Dong, S. Osher, and Z. Shen, Image restoration: Total variation, wavelet frames, and beyond, J. Am. Math. Soc. **25**, 1033 (2012).
[34] E. Oja, Neural networks, principal components, and subspaces, Int. J. Neur. Syst. **01**, 61 (1989).
[35] F. J. Gonzalez and M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, arXiv:1808.01346 (2018).
[36] T. Murata, K. Fukami, and K. Fukagata, Nonlinear mode decomposition with convolutional neural networks for fluid dynamics, J. Fluid Mech. **882**, A13 (2020).
[37] O. Hennigh, Lat-net: Compressing lattice Boltzmann flow simulations using deep neural networks, arXiv:1705.09036.
[38] W. Chen, A. R. Tan, and A. L. Ferguson, Collective variable discovery and enhanced sampling using autoencoders: Innovations in network architecture and error function design, J. Chem. Phys. **149**, 072312 (2018).
[39] W. Wang and R. Gómez-Bombarelli, Coarse-graining auto-encoders for molecular dynamics, npj Computat. Mater. **5**, 125 (2019).
[40] J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. **807**, 155 (2016).
[41] J. H. Ferziger, *Numerical Methods for Engineering Qpplication* (Wiley, New York, 1981), Vol. 1.
[42] D. B. Spalding, A novel finite difference formulation for differential expressions involving both first and second derivatives, Int. J. Numer. Methods Eng. **4**, 551 (1972).
[43] E. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof, Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations, J. Comput. Phys. **161**, 35 (2000).
[44] Y.-H. Tseng and J. H. Ferziger, A ghost-cell immersed boundary method for flow in complex geometry, J. Comput. Phys. **192**, 593 (2003).
[45] P. A. Berthelsen and O. M. Faltinsen, A local directional ghost cell approach for incompressible viscous flow problems with irregular boundaries, J. Comput. Phys. **227**, 4354 (2008).
[46] G. J. Hirasaki and J. Hellums, A general formulation of the boundary conditions on the vector potential in three-dimensional hydrodynamics, Q. Appl. Math. **26**, 331 (1968).
[47] O. Biro and K. Preis, On the use of the magnetic vector potential in the finite-element analysis of three-dimensional eddy currents, IEEE Trans. Magn. **25**, 3145 (1989).

[48] D. Daniel, D. Livescu, and J. Ryu, Reaction analogy based forcing for incompressible scalar turbulence, Phys. Rev. Fluids **3**, 094602 (2018).
[49] S. Pope, *Turbulent Flows* (Cambridge University Press, Cambridge, 2000).
[50] A. Perry and M. Chong, A description of eddying motions and flow patterns using critical-point concepts, Annu. Rev. Fluid Mech. **19**, 125 (1987).
[51] Y. Tian, F. Jaberi, and D. Livescu, Density effects on the post-shock turbulence structure and dynamics, J. Fluid Mech. **880**, 935 (2019).
[52] M. Chertkov, A. Pumir, and B. I. Shraiman, Lagrangian tetrad dynamics and the phenomenology of turbulence, Phys. Fluids **11**, 2394 (1999).
[53] M. Sakurada and T. Yairi, Anomaly detection using autoencoders with nonlinear dimensionality reduction, in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* (ACM Press, Gold Coast, Australia, 2014), pp. 4–11.
[54] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, Dynamic graph cnn for learning on point clouds, arXiv:1801.07829 (2018).
[55] M. Petersen and D. Livescu, Forcing for statistically stationary compressible isotropic turbulence, Phys. Fluids **22**, 116101 (2010).
[56] Y. Morinishi, T. S. Lund, O. V. Vasilyev, and P. Moin, Fully conservative higher order finite difference schemes for incompressible flow, J. Comput. Phys. **143**, 90 (1998).
[57] M. R. Visbal and D. V. Gaitonde, On the use of higher-order finite-difference schemes on curvilinear and deforming meshes, J. Comput. Phys. **181**, 155 (2002).