PNAS
www.pnas.org

Supplementary Information for

Machine learning accelerated computational fluid dynamics

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner and Stephan Hoyer

This PDF file includes:

Supplementary text
Figs. S1 to S6 (not allowed for Brief Reports)
Table S1 (not allowed for Brief Reports)
SI References

Supporting Information Text

1. Direct numerical simulation

Here we describe the details of the numerical solver that we use for data generation, model comparison and as the starting point of our machine learning models. Our solver uses a staggered-square mesh (1) in a finite volume formulation: the computational domain is broken into computational cells where the velocity field is placed on the edges, while the pressure is solved at the cell centers. Our choice of real-space formulation of the Navier-Stokes equations, rather than a spectral method is motivated by practical considerations: real space simulations are much more versatile when dealing with boundary conditions and non-rectangular geometries. We now describe the implementation details of each component.

Convection and diffusion. We implement convection and diffusion operators based on finite-difference approximations. The Laplace operator in the diffusion term is approximated using a second order central difference approximation. The convection term is solved by advecting all velocity components simultaneously, using a high order scheme based on Van-Leer flux limiter (2). For the results presented in the paper we used explicit time integration for both convection and diffusion using Euler discretization. This choice is motivated by performance considerations: for the simulation parameters used in the paper (high Reynold number) implicit diffusion is not required for stable time-stepping, and is approximately twice as slow, due to the additional linear solves required for each velocity component. For diffusion dominated simulations, implicit diffusion would result in faster simulations.

Pressure. To account for pressure we use a projection method, where at each step we solve the corresponding Poisson equation. The solution is obtained using either a fast diagonalization approach with explicit matrix multiplication (3) or a real-valued fast Fourier transform (FFT). The former is well suited for small simulation domains on TPUs as it has better accelerator utilization, while FFT has the best computational complexity and performs best in large simulations. For wall-clock evaluations we choose between the fast diagonalization and FFT approach by choosing the fastest for a given grid.

Forcing and closure terms. We incorporate forcing terms together with the accelerations due to convective and diffusive processes. In an LES setting the baseline and ground truth simulations additionally include a subgrid scale model that is also treated explicitly. We use the Smagorinsky-Lilly model (Eq. (1)) where Δ is the grid spacing and $C_s = 0.2$:

$$\tau_{ij} = -2\,(C_s\Delta)^2|\bar{S}|\bar{S}_{ij} \qquad [1]$$
$$\bar{S}_{ij} = \frac{1}{2}(\partial_i u_j + \partial_j u_i) \qquad |\bar{S}| = 2\sqrt{\sum_{i,j}\bar{S}_{ij}\bar{S}_{ij}}$$

Coarsening. We coarsen to produce labeled "ground truth" datasets on coarser resolution grids, and to initialize our baseline models at different resolutions. To coarsen in space, we average velocity components that lie on the face of coarse-grid cells, as illustrated in Fig. S1. This ensures that the discrete incompressibility condition [Eq. (**??**)] remains exactly satisfied. Besides spatial coarsening, we also coarsen in time with sub-sampling by the same factor, which ensures a fixed ratio between the time-step and grid spacing based on the CFL condition.

2. Datasets and simulation parameters

In the main text we introduced five datasets: two Kolmogorov flows at $Re = 1000$ and $Re = 4000$, Kolmogorov flow with $Re = 1000$ on a $2\times$ larger domain, decaying turbulence and an LES dataset with Reynolds number 10^5. Dataset generation consisted of three steps: (1) burn-in simulation from a random initial condition; (2) simulation for a fixed duration using high resolution solver; (3) downsampling of the solution to a lower grid for training and evaluation.

The burn-in stage is fully characterized by the *burn-in time* and *initialization wavenumber* which represent the discarded initial transient and the peak wavenumber of the log-normal distribution from which random initial conditions were sampled from. The maximum amplitude of the initial velocity field was set to 7 for forced simulations and 4.2 in the decaying turbulence, which was selected to minimize the burn-in time, as well as maintain standard deviation of the velocity field close to 1.0. The initialization wavenumber was set to 4. Simulation parameters include *simulation resolution* along each spatial dimension, *forcing* and *Reynolds number*. The resulting velocity trajectories are then downsampled to the *save resolution*. We determine the time-step based on a Courant?Friedrichs?Lewy (CFL) factor fixed at $C_{\max} = 0.5$, which is standard for numerical simulations, i.e., $\Delta t = C_{\max}\Delta x/||\mathbf{v}||_{\max}$. On a $2\pi \times 2\pi$ domain, this results in a time-step of size 2.1914×10^{-4} on a 2048×2048 grid and 7.0125×10^{-3} on a 64×64 grid. Parameters specifying all five datasets are shown in Table S1. We varied the size of our datasets from 12200 time slices at the downsampled resolution for decaying turbulence to 34770 slices in the forced turbulence settings. Such extended trajectories allow us to analyze stability of models when performing long-term simulations.

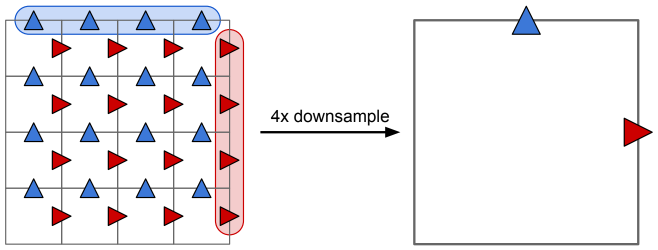

Fig. S1. Example of downsampling staggered velocities to a 4x coarser grid. Velocity components on the fine-mesh located on faces of a coarse-mesh unit-cell (in the indicated shaded regions) are averaged to compute the velocities on the coarse-grid. Velocity components in the interior of the unit-cell are discarded.

We note that there are multiple ways to adjust the simulation parameters to instantiate a Kolmogorov flow with higher Reynolds numbers. In terms of adjustments to the length scale L, kinematic viscosity scale ν and forcing strength scale χ, the Reynolds number scales as $Re \propto L^{3/2}\sqrt{\chi}nu$. A naive approach to achieve a 4-fold increase in the Reynolds number would be to change the length scale $L \to 4^{2/3}L$. While a valid scaling, such setting would lead to a distribution shift in the inputs to neural network components, since the velocity scale $U \propto \sqrt{L\chi}$ would increase. To avoid such issues we used a scaling in which $L \to 2L$; $\nu \to \nu/2$; $\chi \to \chi/2$. This choice achieves a 4-fold increase of the Reynolds number while preserving the velocity scale, as well as the integration time step to which the learned components could specialize. When reporting the results, spatial and time dimensions are scaled back to a fixed value to enable direct comparisons across the experiments.

For comparison of our models with numerical methods we use the corresponding simulation parameters while changing only the resolution of the underlying grid. As mentioned in the Appendix 1, when measuring the performance of all solvers on a given resolution we choose solver components to maximize efficiency.

3. Learned interpolations

To improve numerical accuracy of the convective process our models use psuedo-linear models for interpolation (4, 5). The process of interpolating u_i to $u(x)$ is broken into two steps:

1. Computation of local stencils for interpolation target x.
2. Computation of a weighted sum $\sum_{i=1}^{n} a_i u_i$ over the stencil.

Rather using a fixed set of interpolating coefficients a_i (e.g., as done for typical polynomial interpolation), we choose a_i from the output of a neural network additionally constrained to satisfy $\sum_{i=1}^{n} a_i = 1$, which guarantees that the interpolation is at least first-order accurate. Higher order constraints would not be appropriate, since schemes enforcing strict higher order accuracy can introduce spurious oscillations (2). We enforce first-order accuracy by generating interpolation coefficients with an affine transformation $\mathbf{a} = A\mathbf{x} + \mathbf{b}$ on the unconstrainted outputs x of the neural network, where A is the null-space of the constraint matrix (a $1 \times n$ matrix of ones) of shape $n \times (n-1)$ and $\mathbf{b}$ is an arbitrary valid set of coefficients (we use linear interpolation from the nearest source term locations). This is a special case of the procedure for arbitrary polynomial accuracy constraints on finite difference coefficients described in (4, 5). In this work, we use a 4×4 patch centered over the top-right corner of each unit-cell, which means we need 15 unconstrained neural network outputs to generate each set of interpolation coefficients.

We have investigated the interpolation rules learned by our model by visualizing the predicted interpolation coefficients Fig. S2 for different regions of the flow field. We find that our model discovers augmented versions of physically motivated schemes such as upwinding Fig. S2 (a)(bottom left), as well as more complicated rules around complex flow features.

4. Neural network architectures

All of the ML based models used in this work are based on fully convolutional architectures. Fig. S3 depicts our three modeling approaches (pure ML, learned interpolation and learned correction) and three architecturs for neural network sub-components (Basic ConvNet, Encoder-Process-Decoder and ResNet). Outputs and inputs for each neural network layer are linearly scaled such that appropriate values are in the range $(-1, 1)$.

For our physics augmented solvers (learned interpolation and learned correction), we used the basic ConvNet archicture, with $N_{\text{out}} = 120$ (8 interpolations that need 15 inputs each) for learned interpolation and $N_{\text{out}} = 2$ for learned correction. Our experiments found accuracy was slightly improved by using larger neural networks (such as the Encoder-Process-Decoder and ResNet architectures), but not sufficiently to justify the increased computational cost.

For pure ML solvers, we used EPD and ResNet models that do not build impose physical priors beyond time continuity of the governing equations. In both cases a neural network is used to predict the acceleration due to physical processes that is then summed with the forcing function to compute the state at the next time. Both EPD and ResNet models consist of a sequence of CNN blocks with skip-connections. The main difference is that the EPD model has an arbitrarily large hidden state (in our case, size 64), whereas the ResNet model has a fixed size hidden state equal to 2, the number of velocity components.

5. Details of accuracy measurements

In the main text we have presented accuracy results based on correlation between the predicted flow configurations and the reference solution. We reach the same conclusions based on other common choices, such as squared and absolute errors as shown in Fig. S4 comparing learned discretizations to DNS method on Kolmogorov flow with Reynolds number 1000.

As mentioned in the main text, we additionally evaluated our models on enlarged domains while keeping the flow complexity fixed. Because our models are local, this is the simplest generalization test. As shown in Fig. S5 (and Fig. 5 in the main text), the improvement for $2\times$ larger domains is identical to that found on a smaller domain.

Dataset	Resolution	Re	Burn-in time
Kolmogorov *Re* $= 1000$	$2048 \to 64$	1000	40
Kolmogorov *Re* $= 4000$	$4096 \to 128$	4000	40
Kolmogorov *Re* $= 1000$	$4096 \to 128$	1000	40
Decaying turbulence	$2048 \to 64$	1000 at start	4.5
LES *Re* $= 10^5$	$2048 \to 64$	10^5	40

Table S1. Simulation parameters used for generation of the datasets in this manuscript. For resolution, the notation $X \to Y$ indicates that simulation was performed at resolution X and the result was downsampled to resolution Y.

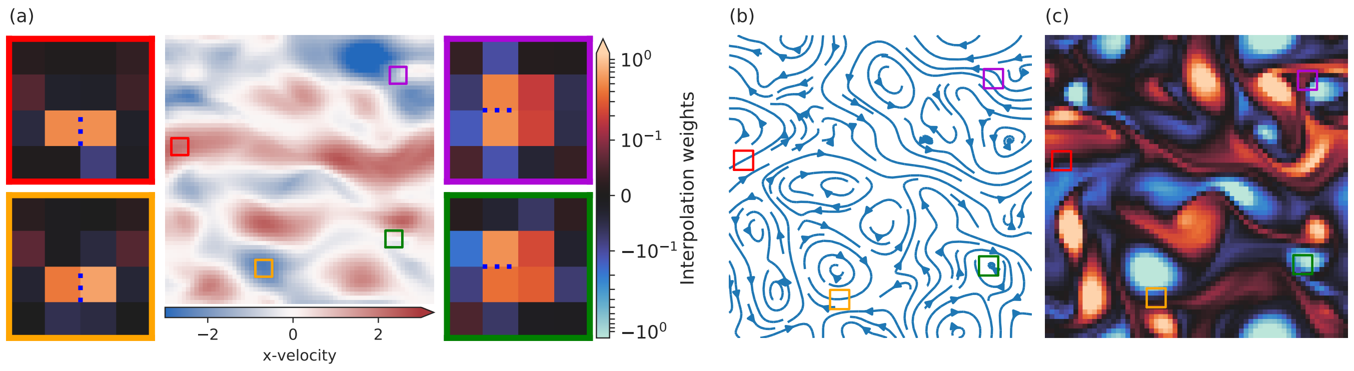

Fig. S2. (a) Visualization of x component of the velocity field and predicted interpolation weights for 4 different flow configurations. (b) Steam plot of the velocity field for which interpolation weights are computed. (c) Vorticity representation of the velocity field.

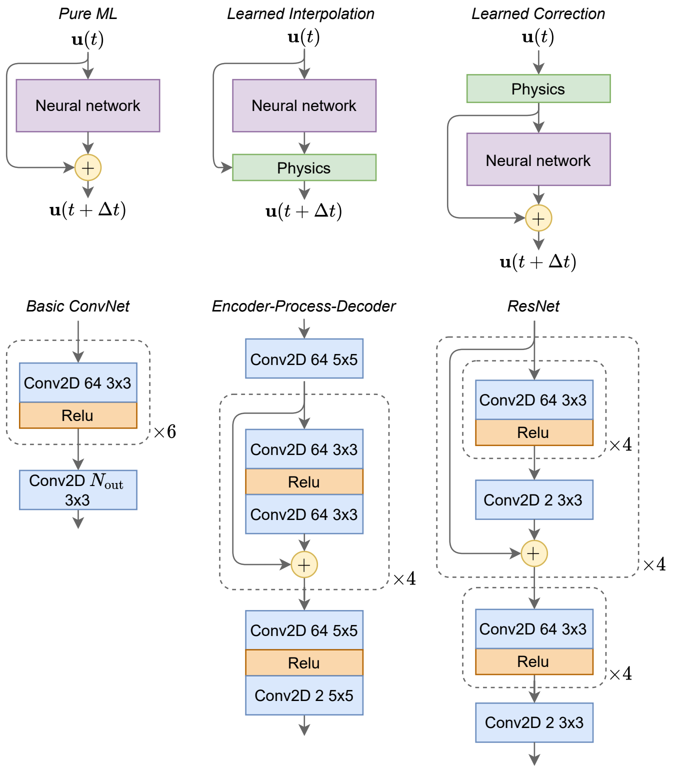

Fig. S3. Modeling approaches and neural network architecutres used in this paper. Rescaling of inputs and outputs from neural network sub-modules with fixed constants is not depicted. Dashed lines surround components that are repeatedly applied the indicated number of times.

6. Details of overview figure

The Pareto frontier of model performance shown in Fig. 1(a) is based on extrapolated accuracy to an 8× larger domain. Due to computational limits, we were unable to run the simulations on the 16384×16384 grid for measuring accuracy, so the time until correlation goes below 0.95 was instead taken from the 1× domain size, which as described in the preceding section matched performance on the 2× domain.

The 512×512 learned interpolation model corresponds to the depicted results in Fig. 2, whereas the 256×256 and 1024×1024 models were retrained on the same training dataset for 2× coarser or 2× finer coarse-graining.

7. Model comparison by training run

Our figure showing model performance in the main text [Fig. 5] does not allow for comparing models with the same learned weights across generalization tasks or evaluation metrics. Figure S6 presents an alternative view of the same results, colored by random number seed used for initializing neural network weights instead of by model architecture. We see that models with poor performance within the training data regime (forced turbulence) also have poor performance on the generalization tests, but is no clear relationship for performance across different generalization tasks or between pointwise and statistical accuracy.

8. Hyperparameter tuning

All models were trained using Adam (6) optimizer. Our initial experiments showed that none of the models had a consistent dependence on the optimization parameters. The set that worked well for all models included learning rate of 10^{-3}, $b1 = 0.9$ and $b2 = 0.99$. For each model we performed a hyperparameter sweep over the length of the training trajectory used to compute the loss, model capacity such as size and number of hidden layers, as well as a sweep over different random initializations to assess reliability and consistency of training.

References

1. JM McDonough, Lectures in computational fluid dynamics of incompressible flow: Mathematics, algorithms and implementations. (2007).
2. PK Sweby, High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM journal on numerical analysis* **21**, 995–1011 (1984).
3. RE Lynch, JR Rice, DH Thomas, Direct solution of partial difference equations by tensor product methods. *Numer. Math.* **6**, 185–199 (1964).
4. Y Bar-Sinai, S Hoyer, J Hickey, MP Brenner, Learning data-driven discretizations for partial differential equations. *Proc. Natl. Acad. Sci.* **116**, 15344–15349 (2019).
5. J Zhuang, D Kochkov, Y Bar-Sinai, MP Brenner, S Hoyer, Learned discretizations for passive scalar advection in a 2-d turbulent flow. *arXiv preprint arXiv:2004.05477* (2020).
6. DP Kingma, J Ba, Adam: A method for stochastic optimization in *3rd International Conference on Learning Representations ICLR.* (2015).

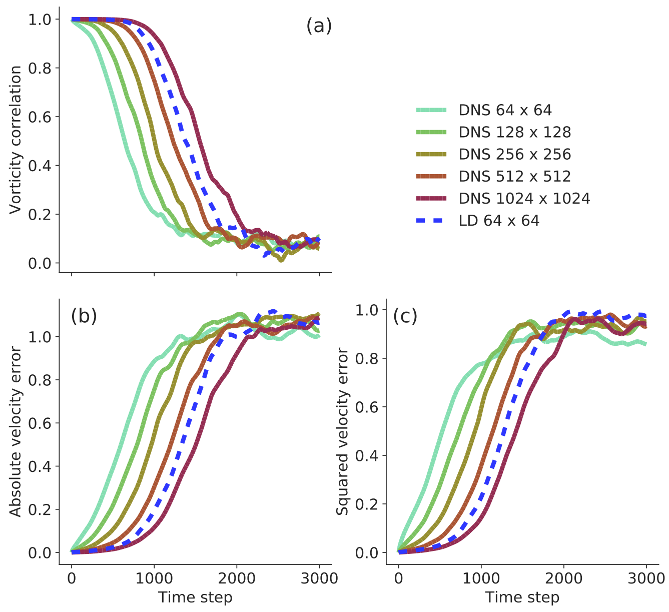

Fig. S4. Comparison of ML + CFD model to DNS running at different resolutions using varying metrics. (a) Vorticity correlation as presented in the main text. (b) Mean absolute error. (c) Absolute error of the kinetic energy.

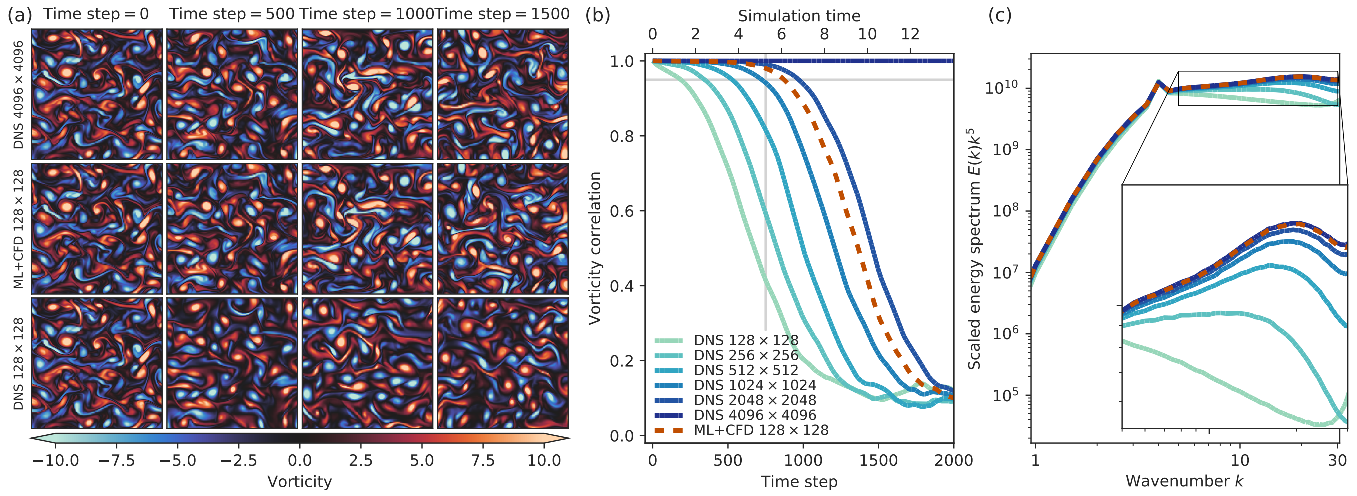

Fig. S5. Comparison of ML + CFD model to DNS running at different resolutions when evaluated on a $2\times$ larger domain with characteristic length-scales matching those of the training data. (a) Visualization of the vorticity at different times. (b) Vorticity correlation as a function of time. (c) Scaled energy spectrum $E(k) * k^5$.

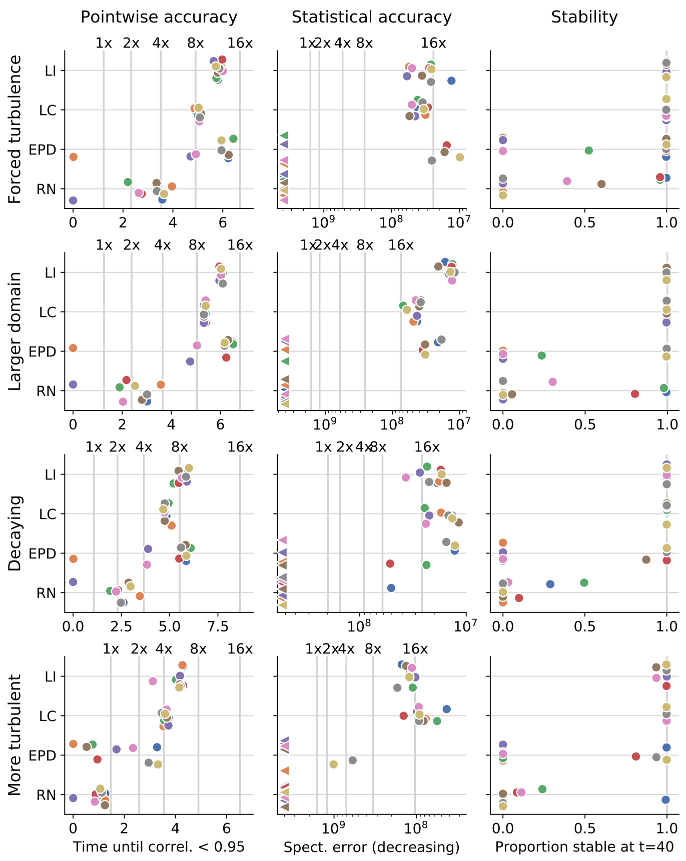

Fig. S6. Detailed model comparison like Fig. 5, except colored by random number seed rather than model architecture. Markers with the same color and the same model architecture (LI, LC, EPD or RN) correspond to the same learned weights.