arXiv:2102.01010v1 [physics.flu-dyn] 28 Jan 2021

Machine learning accelerated computational fluid dynamics

Dmitrii Kochkov,[1,*] Jamie A. Smith,[1,*] Ayya Alieva,[1] Qing Wang,[1] Michael P. Brenner,[1,2] and Stephan Hoyer[1,†]

[1] *Google Research*

[2] *School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138*

Numerical simulation of fluids plays an essential role in modeling many physical phenomena, such as weather, climate, aerodynamics and plasma physics. Fluids are well described by the Navier-Stokes equations, but solving these equations at scale remains daunting, limited by the computational cost of resolving the smallest spatiotemporal features. This leads to unfavorable trade-offs between accuracy and tractability. Here we use end-to-end deep learning to improve approximations inside computational fluid dynamics for modeling two-dimensional turbulent flows. For both direct numerical simulation of turbulence and large eddy simulation, our results are as accurate as baseline solvers with 8-10x finer resolution in each spatial dimension, resulting in 40-80x fold computational speedups. Our method remains stable during long simulations, and generalizes to forcing functions and Reynolds numbers outside of the flows where it is trained, in contrast to black box machine learning approaches. Our approach exemplifies how scientific computing can leverage machine learning and hardware accelerators to improve simulations without sacrificing accuracy or generalization.

I. INTRODUCTION

Simulation of complex physical systems described by nonlinear partial differential equations are central to engineering and physical science, with applications ranging from weather [1, 2] and climate [3, 4] , engineering design of vehicles or engines [5], to wildfires [6] and plasma physics [7]. Despite a direct link between the equations of motion and the basic laws of physics, it is impossible to carry out direct numerical simulations at the scale required for these important problems. This fundamental issue has stymied progress in scientific computation for decades, and arises from the fact that an accurate simulation must resolve the smallest spatiotemporal scales.

A paradigmatic example is turbulent fluid flow [8], underlying simulations of weather, climate, and aerodynamics. The size of the smallest eddy is tiny: for an airplane with chord length of 2 meters, the smallest length scale (the Kolomogorov scale) [9] is $O(10^{-6})m$. Classical methods for computational fluid dynamics (CFD), such as finite differences, finite volumes, finite elements and pseudo-spectral methods, are only accurate if fields vary smoothly on the mesh, and hence meshes must resolve the smallest features to guarantee convergence. For a turbulent fluid flow, the requirement to resolve the smallest flow features implies a computational cost scaling like Re^3, where $Re = UL/\nu$, with U and L the typical velocity and length scales and ν the kinematic viscosity. A tenfold increase in Re leads to a thousandfold increase in the computational cost. Consequently, direct numerical simulation (DNS) for e.g. climate and weather are impossible. Instead, it is traditional to use smoothed versions of the Navier Stokes equations [10, 11] that allow coarser grids while sacrificing accuracy, such as Reynolds Averaged Navier-Stokes (RANS) [12, 13], and Large-Eddy Simulation (LES) [14, 15]. For example, current state-of-art LES with mesh sizes of $\mathcal{O}(10)$ to $\mathcal{O}(100)$ million has been used in the design of internal combustion engines [16], gas turbine engines [17, 18], and turbo-machinery [19]. Despite promising progress in LES over the last two decades, there are severe limits to what can be accurately simulated. This is mainly due to the first-order dependence of LES on the sub-grid scale (SGS) model, especially for flows whose rate controlling scale is unresolved [20].

Here, we introduce a method for calculating the accurate time evolution of solutions to nonlinear partial differential equations, while using an order of magnitude coarser grid than is traditionally required for the same accuracy. This is a novel type of numerical solver that does not average unresolved degrees of freedom, but instead uses discrete equations that give pointwise accurate solutions on an unresolved grid. We discover these algorithms using machine learning, by replacing the components of traditional solvers most affected by the loss of resolution with learned alternatives. As shown in Fig. 1(a), for a two dimensional direct numerical simulation of a turbulent flow, our algorithm maintains accuracy while using $10\times$ coarser resolution in each dimension, resulting in a $\sim$ 80 fold improvement in computational time with respect to an advanced numerical method of similar accuracy. The model learns how to interpolate local features of solutions and hence can accurately generalize to different flow conditions such as different forcings and even different Reynolds numbers [Fig. 1(b)]. We also apply the method to a high resolution LES simulation of a turbulent flow and show similar performance enhancements, maintaining pointwise accuracy on $Re = 100,000$

* D.K. and J.A. S. contributed equally to this work.
† To whom correspondence should be addressed. E-mail: dkochkov@google.com, jamieas@google.com, shoyer@google.com

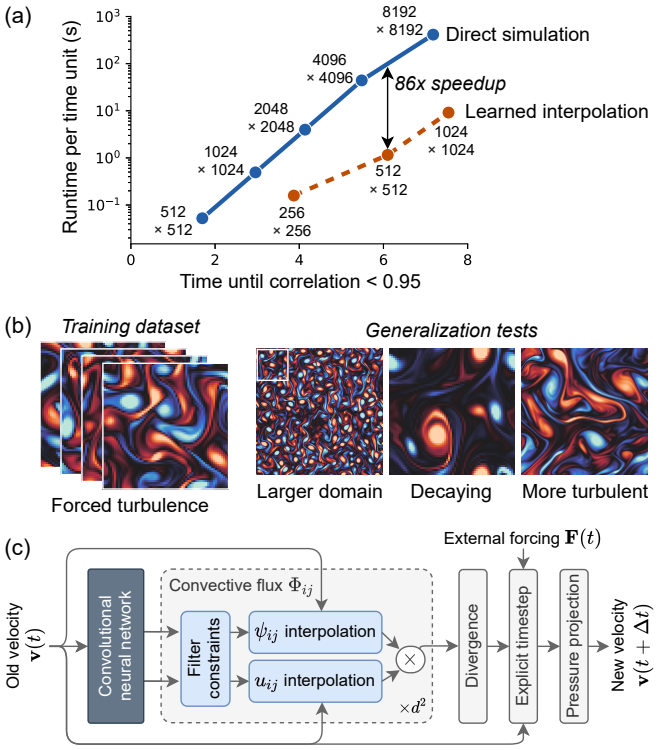

FIG. 1. Overview of our approach and results. (a) Accuracy versus computational cost with our baseline (direct simulation) and ML accelerated (learned interpolation) solvers. The x axis corresponds to pointwise accuracy, showing how long the simulation is highly correlated with the ground truth, whereas the y-axis shows the computational time needed to carry out one simulation time-unit on a single TPU core. Each point is annotated by the size of the corresponding spatial grid: for details see the appendix. (b) Illustrative training and validation examples, showing the strong generalization capabilities of our model. (c) Structure of a single time step for our "learned interpolation" model, with a convolutional neural net controlling learned approximations inside the convection calculation of a standard numerical solver. ψ and u refer to advected and advecting velocity components. For d spatial dimensions there are d^2 replicates of the convective flux module, corresponding to the flux of each velocity component in each spatial direction.

LES simulations using $8\times$ times coarser grids with ~ 40 fold computational speedup.

There has been a flurry of recent work using machine learning to improve turbulence modeling. One major family of approaches uses ML to fit closures to classical turbulence models based on agreement with high resolution direct numerical simulations (DNS) [21–24]. While potentially more accurate than traditional turbulence models, these new models have not achieved reduced computational expense. Another major thrust uses "pure" ML, aiming to replace the entire Navier Stokes simulation with approximations based on deep neural networks [25–30]. A pure ML approach can be extremely efficient, avoiding the severe time-step constraints required for stability with traditional approaches. Because these models do not include the underlying physics, they often struggle to enforce constraints, such as conservation of momentum and incompressibility. While these models often perform well on data from the training distribution, they often struggle with generalization. For example, they perform worse when exposed to novel forcing terms. A third approach, which we build upon in this work, uses ML to correct errors in cheap, under-resolved simulations [31–33]. These models borrow strength from the coarse-grained simulations.

In this work we design algorithms that accurately solve the equations on coarser grids by replacing the components most affected by the resolution loss with better performing learned alternatives. We use *data driven discretizations* [34, 35] to interpolate differential operators onto a coarse mesh with high accuracy [Fig. 1(c)]. We train the solver inside a standard numerical method for solving the underlying PDEs as a differentiable program, with the neural networks and the numerical method written in a framework (JAX [36]) supporting reverse-mode automatic differentiation. This allows for end-to-end gradient based optimization of the entire algorithm, similar to prior work on density functional theory [37], molecular dynamics [38] and fluids [31, 32]. The methods we derive are equation specific, and require training a coarse resolution solver with high resolution ground truth simulations. Since the dynamics of a partial differential equation are local, the high resolution simulations can be carried out on a small domain. The models remains stable during long simulations and has robust and predictable generalization properties, with models trained on small domains producing accurate simulations on larger domains, with different forcing functions and even with different Reynolds number. Comparison to pure ML baselines shows that generalization arises from the physical constraints inherent in the formulation of the method.

II. BACKGROUND

A. Navier-Stokes

Incompressible fluids are modeled by the Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -\nabla\cdot(\mathbf{u}\otimes\mathbf{u}) + \frac{1}{Re}\nabla^2\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{f} \tag{1a}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{1b}$$

where $\mathbf{u}$ is the velocity field, $\mathbf{f}$ the external forcing, and $\otimes$ denotes a tensor product. The density ρ is a constant, and the pressure p is a Lagrange multiplier used to enforce (1b). The Reynolds number Re dictates the balance between the convection (first) or diffusion (second) terms in the right hand side of (1a). Higher Reynolds number flows dominated by convection are more complex

and thus generally harder to model; flows are considered "turbulent" if $Re \gg 1$.

Direct numerical simulation (DNS) solves (1) directly, whereas large eddy simulation (LES) solves a spatially filtered version. In the equations of LES, $\mathbf{u}$ is replaced by a filtered velocity $\overline{\mathbf{u}}$ and an sub-grid term $-\nabla \cdot \boldsymbol{\tau}$ is added to the right side of (1a), with the sub-grid stress defined as $\boldsymbol{\tau} = \overline{\mathbf{u} \otimes \mathbf{u}} - \overline{\mathbf{u}} \otimes \overline{\mathbf{u}}$. Because $\overline{\mathbf{u} \otimes \mathbf{u}}$ is un-modeled, solving LES also requires a choice of *closure* model for $\boldsymbol{\tau}$ as a function of $\overline{\mathbf{u}}$. Numerical simulation of both DNS and LES further requires a *discretization* step to approximate the continuous equations on a grid. Traditional discretization methods (e.g., finite differences) converge to an exact solution as the grid spacing becomes small, with LES converging faster because it models a smoother quantity. Together, discretization and closure models are the two principle sources of error when simulating fluids on coarse grids [31, 39].

B. Learned solvers

Our principle aim is to accelerate DNS without compromising accuracy or generalization. To that end, we consider ML modeling approaches that enhance a standard CFD solver when run on inexpensive to simulate coarse grids. We expect that ML models can improve the accuracy of the numerical solver via effective super-resolution of missing details [34]. Because we want to train neural networks for approximation *inside* our solver, we wrote a new CFD code in JAX [36], which allows us to efficiently calculate gradients via automatic differentiation. Our CFD code is a standard implementation of a finite volume method on a regular staggered mesh, with first-order explicit time-stepping for convection, diffusion and forcing, and implicit treatment of pressure; for details see the appendix.

The algorithm works as follows: in each time-step, the neural network generates a latent vector at each grid location based on the current velocity field, which is then used by the sub-components of the solver to account for local solution structure. Our neural networks are convolutional, which enforces translation invariance and allows them to be local in space. We then use components from standard standard numerical methods to enforce inductive biases corresponding to the physics of the Navier Stokes equations, as illustrated by the light gray boxes in Fig. 1(c): the convective flux model improves the approximation of the discretized convection operator; the divergence operator enforces local conservation of momentum according to a finite volume method; the pressure projection enforces incompressibility and the explicit time step operator forces the dynamics to be continuous in time, allowing for the incorporation of additional time varying forces. "DNS on a coarse grid" blurs the boundaries of traditional DNS and LES modeling, and thus invites a variety of data-driven approaches. In this work we focus on two types of ML components: learned interpolation and learned correction. Both focus on the convection term, the key term in (1) for turbulent flows.

1. *Learned interpolation (LI)*

In a finite volume method, $\mathbf{u}$ denotes a vector field of volume averages over unit cells, and the cell-averaged divergence can be calculated via Gauss' theorem by summing the surface flux over the each face. This suggests that our only required approximation is calculating the convective flux $\mathbf{u} \otimes \mathbf{u}$ on each face, which requires interpolating $\mathbf{u}$ from where it is defined. Rather than using typical polynomial interpolation, which is suitable for interpolation without prior knowledge, here we use an approach that we call *learned interpolation* based on data driven discretizations [34]. We use the outputs of the neural network to generate interpolation coefficients based on local features of the flow, similar to the fixed coefficients of polynomial interpolation. This allows us to incorporate two important priors: (1) the equation maintains the same symmetries and scaling properties (e.g., rescaling coordinates $\mathbf{x} \to \lambda\mathbf{x}$) as the original equations, and (2) as the mesh spacing vanishes, the interpolation module retains polynomial accuracy so that our model performs well in the regime where traditional numerical methods excel.

2. *Learned correction (LC)*

An alternative approach, closer in spirit to LES modeling, is to simply model a residual correction to the discretized Navier-Stokes equations [(1)] on a coarse-grid [31, 32]. Such an approach generalizes traditional closure models for LES, but in principle can also account for discretization error. We consider *learned correction* models of the form $\mathbf{u}_t = \mathbf{u}_t^* + \mathrm{LC}(\mathbf{u}_t^*)$, where LC is a neural network and $\mathbf{u}_t^*$ is the uncorrected velocity field from the numerical solver on a coarse grid. Modeling the residual is appropriate both from the perspective of a temporally discretized closure model, and pragmatically because the relative error between $\mathbf{u}_t$ and $\mathbf{u}_t^*$ in a single time step is small. LC models have fewer inductive biases than LI models, but they are simpler to implement and potentially more flexible. We also explored LC models restricted to take the form of classical closure models (e.g., flow-dependent effective tensor viscosity models), but the restrictions hurt model performance and stability.

3. *Training*

The training procedure tunes the machine learning components of the solver to minimize the discrepancy between an expensive high resolution simulation and a simulation produced by the model on a coarse grid. We

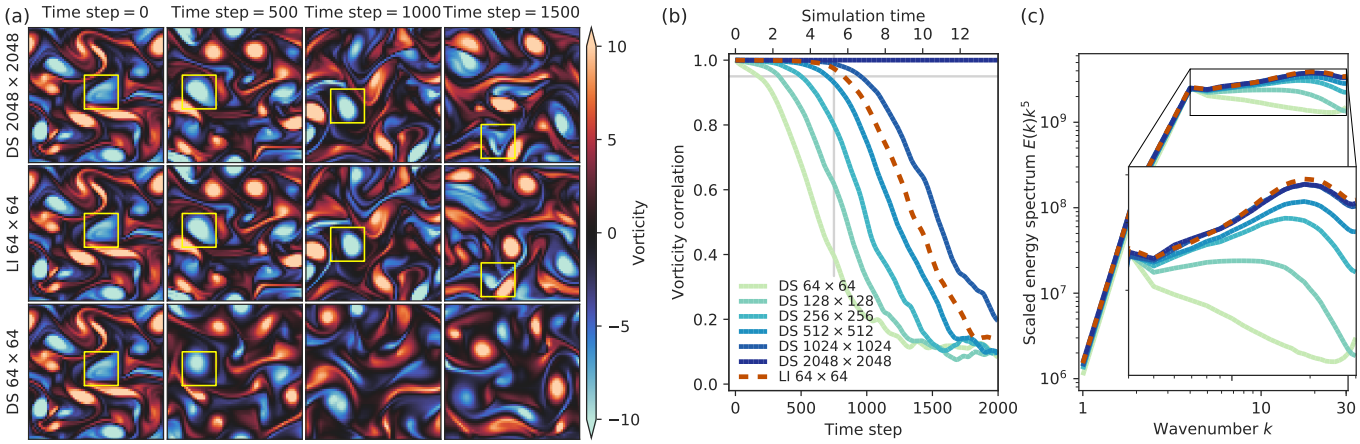

FIG. 2. Learned interpolation achieves accuracy of direct simulation at $\sim 10\times$ higher resolution. (a) Evolution of predicted vorticity fields for reference (DS 2048×2048), learned (LI 64×64) and baseline (DS 64×64) solvers, starting from the same initial velocities. The yellow box traces the evolution of a single vortex. (b) Comparison of the vorticity correlation between predicted flows and the reference solution for our model and DNS solvers. (c) Energy spectrum scaled by k^5 averaged between time-steps 10000 and 20000, when all solutions have decorrelated with the reference solution.

accomplish this via supervised training where we use a cumulative point-wise error between the predicted and ground truth velocities as the loss function

$$L(x, y) = \sum_{t_i}^{t_T} \text{MSE}(\mathbf{u}(t_i), \tilde{\mathbf{u}}(t_i)).$$

The ground truth trajectories are obtained by using a high resolution simulation that is then coarsened to the simulation grid. Including the numerical solver in the training loss ensures fully "model consistent" training where the model sees its own outputs as inputs [23, 32, 39], unlike typical *a priori* training where simulation is only performed offline. As an example, for the Kolmogorov flow simulations below with Reynolds number 1000, our ground truth simulation had a resolution of 2048 cells along each spatial dimension. We subsample these ground truth trajectories along each dimension and time by a factor of 32. For training we use 32 trajectories of 4800 sequential time steps each, starting from different random initial conditions. To evaluate the model, we generate much longer trajectories (tens of thousands of time steps) to verify that models remain stable and produce plausible outputs. We unroll the model for 32 time steps when calculating the loss, which we find improves model performance for long time trajectories [32], in some cases using gradient checkpoints at each model step to reduce memory usage [40].

III. RESULTS

We take a utilitarian perspective on model evaluation: simulation methods are good insofar as they demonstrate accuracy, computational efficiency and generalizability. In this case, accuracy and computational efficiency require the method to be faster than the DNS baseline, while maintaining accuracy for long term predictions; generalization means that although the model is trained on specific flows, it must be able to readily generalize well to new simulation settings, including to different forcings and different Reynolds numbers. In what follows, we first compare the accuracy and generalizability of our method to both direct numerical simulation and several existing ML-based approaches for simulations of two dimensional turbulence flow. In particular, we first consider Kolmogorov flow [41], a parametric family of forced turbulent flows obeying the Navier-Stokes equation [(1)], with forcing $f = \sin(4y)\hat{\mathbf{x}} - 0.1\mathbf{u}$, where the second term is a velocity dependent drag preventing accumulation of energy at large scales [42]. Kolmogorov flow produces a statistically stationary turbulent flow, with flow complexity controlled by a single parameter, the Reynolds number *Re*.

A. Accelerating DNS

The accuracy of a direct numerical simulation quickly degrades once the grid resolution cannot capture the smallest details of the solution. In contrast, our ML-based approach strongly mitigates this effect. Figure 2 shows the results of training and evaluating our model on Kolmogorov flows at Reynolds number Re = 1000. All datasets were generated using high resolution DNS, followed by a coarsening step.

1. *Accuracy*

The scalar vorticity field $\omega = \partial_x u_y - \partial_y u_x$ is a convenient way to describe a two-dimensional incompressible flows [42]. Accuracy can be quantified by correlating vor-

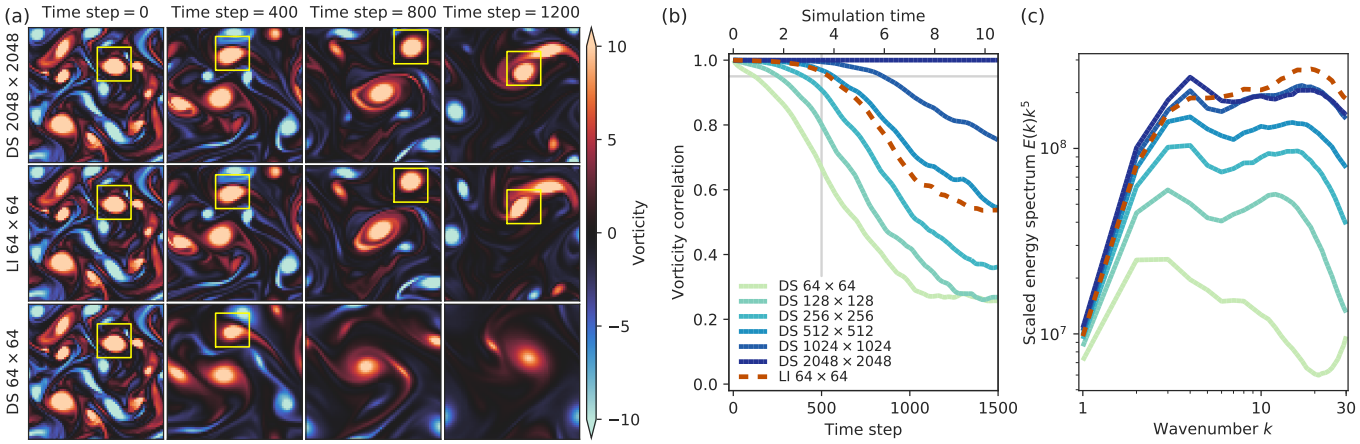

FIG. 3. Learned interpolation generalizes well to decaying turbulence. (a) Evolution of predicted vorticity fields as a function of time. (b) Vorticity correlation between predicted flows and the reference solution. (c) Energy spectrum scaled by k^5 averaged between time-steps 2000 and 2500, when all solutions have decorrelated with the reference solution.

ticity fields,[43] $C(\omega, \hat{\omega})$ between the ground truth solution ω and the predicted state $\hat{\omega}$. Fig. 2 compares the learned interpolation model (64×64) to fully resolved DNS of Kolmogorov flow (2048×2048) using an initial condition that was not included in the training set. Strikingly, the learned discretization model matches the pointwise accuracy of DNS with a ~ 10 times finer grid. The eventual loss of correlation with the reference solution is expected due to the chaotic nature of turbulent flows; this is marked by a vertical grey line in Fig. 2(b), indicating the first three Lyapunov times. Fig. 2 (a) shows the time evolution of the vorticity field for three different models: the learned interpolation matches the ground truth (2048×2048) more accurately than the 512×512 baseline, whereas it greatly outperforms a baseline solver at the same resolution as the model (64×64).

The learned interpolation model also produces a similar energy spectrum $E(\mathbf{k}) = \frac{1}{2}|\mathbf{u}(\mathbf{k})|^2$ to DNS. With decreasing resolution, DNS cannot capture high frequency features, resulting in an energy spectrum that "tails off" for higher values of k. Fig. 2 (c) compares the energy spectrum for learned interpolation and direct simulation at different resolutions after 10^4 time steps. The learned interpolation model accurately captures the energy distribution across the spectrum.

2. *Computational efficiency*

The ability to match DNS with a ~ 10 times coarser grid makes the learned interpolation solver much faster. We benchmark our solver on a single core of Google's Cloud TPU v4, a hardware accelerator designed for accelerating machine learning models that is also suitable for many scientific computing use-cases [44–46]. The TPU is designed for high throughput vectorized operations, and extremely high throughput matrix-matrix multiplication in low precision (bfloat16). On sufficiently large grid sizes (256×256 and larger), our neural net makes good use of matrix-multiplication unit, achieving 12.5x higher throughput in floating point operations per second than our baseline CFD solver. Thus despite using 150 times more arithmetic operations, the ML solver is only about 12 times slower than the traditional solver at the same resolution. The $10\times$ gain in effective resolution in three dimensions (two space dimensions and time, due to the Courant condition) thus corresponds to a speedup of $10^3/12 \approx 80$.

3. *Generalization*

In order to be useful, a learned model must accurately simulate flows outside of the training distribution. We expect our models to generalize well because they learn local operators: interpolated values and corrections at a given point depend only on the flow within a small neighborhood around it. As a result, these operators can be applied to any flow that features similar local structures as those seen during training. We consider three different types of generalization tests: (1) larger domain size, (2) unforced decaying turbulent flow, and (3) Kolmogorov flow at a larger Reynolds number.

First, we test generalization to larger domain sizes with the same forcing. Our ML models have essentially the exact same performance as on the training domain, because they only rely upon local features of the flows.(see Appendix E and Fig. 5).

Second, we apply our model trained on Kolmogorov flow to *decaying turbulence*, by starting with a random initial condition with high wavenumber components, and letting the turbulence evolve in time without forcing. Over time, the small scales coalesce to form large scale structures, so that both the scale of the eddies and the Reynolds number vary. Figure 3 shows that a learned discretization

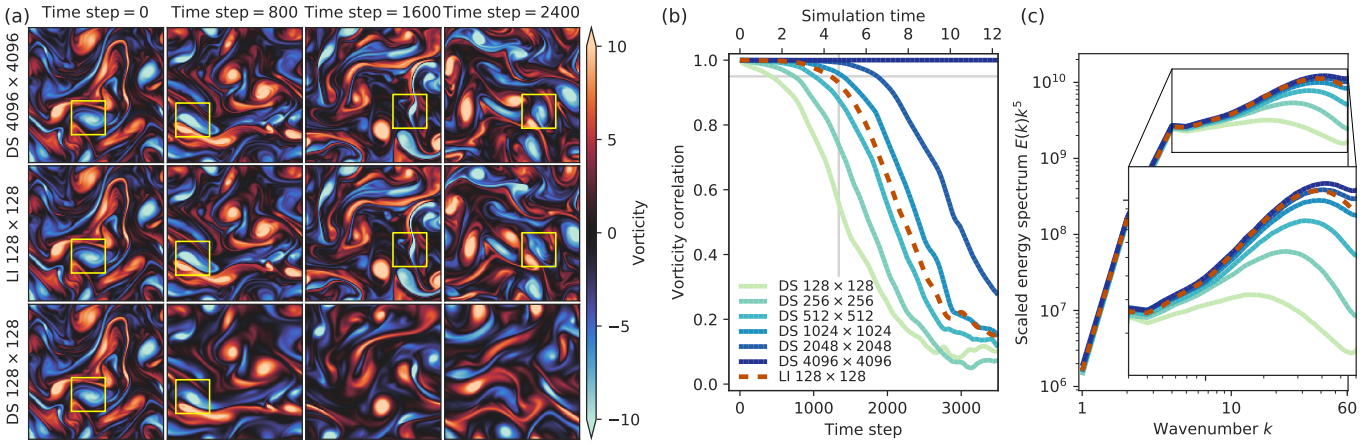

FIG. 4. Learned interpolations can be scaled to simulate higher Reynolds numbers without retraining. (a) Evolution of predicted vorticity fields as a function of time for Kolmogorov flow at $Re = 4000$. (b) Vorticity correlation between predicted flows and the reference solution. (c) Energy spectrum scaled by k^5 averaged between time-steps 6000 and 12000, when all solutions have decorrelated with the reference solution.

model trained on Kolmogorov flows Re = 1000 can match the accuracy of DNS running at ~ 7 times finer resolution. A standard numerical method at the same resolution as the learned discretization model is corrupted by numerical diffusion, degrading the energy spectrum as well as pointwise accuracy.

Our final generalization test is harder: can the models generalize to higher Reynolds number where the flows are more complex? The universality of the turbulent cascade [1, 47, 48] implies that at the size of the smallest eddies (the Kolmogorov length scale), flows "look the same" regardless of Reynolds number when suitably rescaled. This suggests that we can apply the model trained at one Reynolds number to a flow at another Reynolds number by simply rescaling the model to match the new smallest length scale. To test this we construct a new dataset for a Kolmogorov flow with Re = 4000. The theory of two-dimensional turbulence [49] implies that the smallest eddy size decreases as $1/\sqrt{\text{Re}}$, implying that the smallest eddies in this flow are 1/2 that for original flow with Re = 1000. We therefore can use a trained Re = 1000 model at Re = 4000 by simply halving the grid spacing. Fig. 4 (a) shows that with this scaling, our model achieves the accuracy of DNS running at 7 times finer resolution. This degree of generalization is remarkable, given that we are now testing the model with a flow of substantially greater complexity. Fig. 4 (b) visualizes the vorticity, showing that higher complexity is captured correctly, as is further verified by the energy spectrum shown in Fig. 4 (c).

B. Comparison to other ML models

Finally, we compare the performance of learned interpolation to alternative ML-based methods. We consider three popular ML methods: ResNet (RN) [50], Encoder-Processor-Decoder (EPD) [51, 52] architectures and the learned correction (LC) model introduced earlier. These models all perform explicit time-stepping without any additional latent state beyond the velocity field, which allows them to be evaluated with arbitrary forcings and boundary conditions, and to use the time-step based on the CFL condition. By construction, these models are inivariant to translation in space and time, and have similar runtime for inference (varying within a factor of two). To evaluate training consistency, each model is trained 9 times with different random initializations on the same Kolmogorov Re = 1000 dataset described previously. Hyperparameters for each model were chosen as detailed in Appendix G, and the models are evaluated on the same generalization tasks. We compare their performance using several metrics: time until vorticity correlation falls below 0.95 to measure pointwise accuracy for the flow over short time windows, the absolute error of the energy spectrum scaled by k^5 to measure statistical accuracy for the flow over long time windows, and the fraction of simulated velocity values that does not exceed the range of the training data to measure stability.

Fig. 5 compares results across all considered configurations. Overall, we find that learned interpolation (LI) performs best, although learned correction (LC) is not far behind. We were impressed by the performance of the LC model, despite its weaker inductive biases. The difference in effective resolution for pointwise accuracy (8× vs 10× upscaling) corresponds to about a factor of two in run-time. There are a few isolated exceptions where pure black box methods outperform the others, but not consistently. A particular strength of the learned interpolation and correction models is their consistent performance and generalization to other flows, as seen from the narrow spread of model performance for different random initialization and their consistent dominance over other models in the generalization tests. Note that even a mod-

est 4× effective coarse-graining in resolution still corresponds to a 5× computational speed-up. In contrast, the black box ML methods exhibit high sensitivity to random initialization and do not generalize well, with much less consistent statistical accuracy and stability.

C. Acceleration of LES

Finally, up until now we have illustrated our method for DNS of the Navier Stokes equations. Our approach is quite general and could be applied to any nonlinear partial differential equation. To demonstrate this, we apply the method to accelerate LES, the industry standard method for large scale simulations where DNS is not feasible.

Here we treat the LES at high resolution as the ground truth simulation and train an interpolation model on a coarser grid for Kolmogorov flows with Reynolds number $Re = 10^5$ according to the Smagorinsky-Lilly model SGS model [8]. Our training procedure follows the exact same approach we used for modeling DNS. Note in particular that we do not attempt to model the parameterized viscosity in the learned LES model, but rather let learned interpolation model this implicitly. Fig. 6 shows that learned interpolation for LES still achieves an effective 8× upscaling, corresponding to roughly 40× speedup.

IV. DISCUSSION

In this work we present a data driven numerical method that achieves the same accuracy as traditional finite difference/finite volume methods but with much coarser resolution. The method learns accurate local operators for convective fluxes and residual terms, and matches the accuracy of an advanced numerical solver running at 8–10× finer resolution, while performing the computation 40–80× faster. The method uses machine learning to interpolate better at a coarse scale, within the framework of the traditional numerical discretizations. As such, the method inherently contains the scaling and symmetry properties of the original governing Navier Stokes equations. For that reason, the methods generalize much better than pure black-box machine learned methods, not only to different forcing functions but also to different parameter regimes (Reynolds numbers).

What outlook do our results suggest for speeding up 3D turbulence? In general, the runtime T for efficient ML augmented simulation of time-dependent PDEs should scale like

$$T \sim (C_{\mathrm{ML}} + C_{\mathrm{physics}}) \left(\frac{N}{K}\right)^{d+1}, \qquad (2)$$

where C_{ML} is the cost of ML inference per grid point, C_{physics} is the cost of baseline numerical method, N is the number of grid points along each dimension of the resolved grid, d is the number of spatial dimensions and K is the effective coarse graining factor. Currently, $C_{\mathrm{ML}}/C_{\mathrm{physics}} \approx 12$, but we expect that much more efficient machine learning models are possible, e.g., by sharing work between time-steps with recurrent neural nets. We expect the 10× decrease in effective resolution discovered here to generalize to 3D and more complex problems. This suggests that speed-ups in the range of 10^3–10^4 may be possible for 3D simulations. Further speed-ups, as required to capture the full range of turbulent flows, will require either more efficient representations for flows (e.g., based on solution manifolds rather than a grid) or being satisfied with statistical rather than pointwise accuracy (e.g., as done in LES modeling).

In summary, our approach expands the Pareto frontier of efficient simulation in computational fluid dynamics, as illustrated in Fig. 1(a). With ML accelerated CFD, users may either solve expensive simulations much faster, or in-

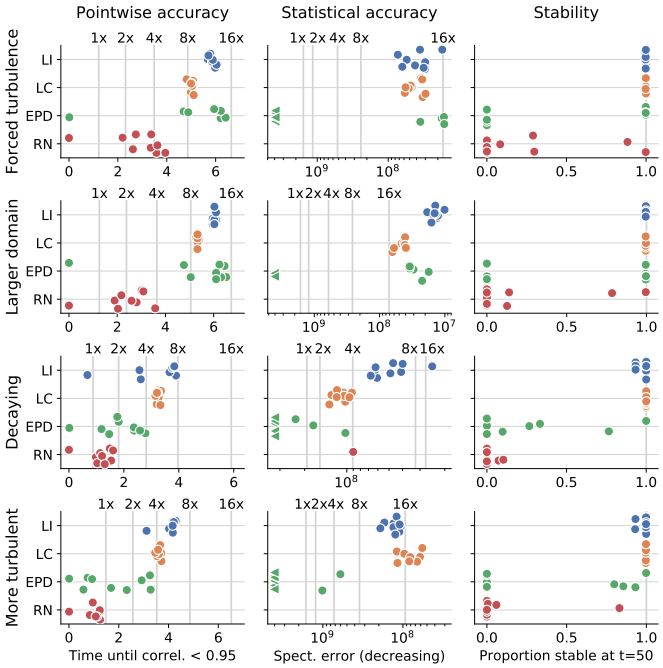

FIG. 5. Learned discretizations outperform a wide range of baseline methods in terms of accuracy, stability and generalization. Each row within a subplot shows performance metrics for nine model replicates with the same architecture, but different randomly initialized weights. The models are trained on forced turbulence, with larger domain, decaying and more turbulent flows as generalization tests. Vertical lines indicate performance of non-learned baseline models at different resolutions (all baseline models are perfectly stable). The pointwise accuracy test measures error at the start of time integration, whereas the statistical accuracy and stability tests are both performed on simulations at time 50 after about 7000 time integration steps (twice the number of steps for more turbulent flow). The points indicated by a left-pointing triangle in the statistical accuracy tests are clipped at a maximum error.

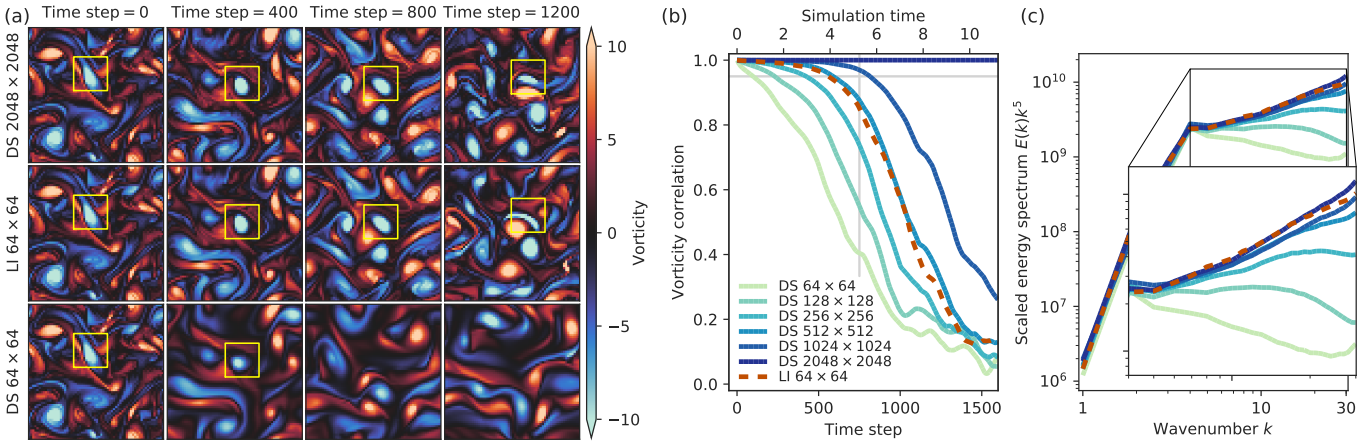

FIG. 6. Learned discretizations achieve accuracy of LES simulation running on 8× times finer resolution. (a) Evolution of predicted vorticity fields as a function of time. (b) Vorticity correlation between predicted flows and the reference solution. (c) Energy spectrum scaled by k^5 averaged between time-steps 3800 and 4800, when all solutions have decorrelated with the reference solution.

crease accuracy without additional costs. To put these results in context, if applied to numerical weather prediction, increasing the duration of accurate predictions from 4 to 7 time-units would correspond to approximately 30 years of progress [53]. These improvements are possible due to the combined effect of two technologies still undergoing rapid improvements: modern deep learning models, which allow for accurate simulation with much more compact representations, and modern accelerator hardware, which allows for evaluating said models with a remarkably small increase in computational cost. We expect both trends to continue for the foreseeable future, and to eventually impact all areas of computationally limited science.

ACKNOWLEDGEMENT

We thank John Platt and Rif A. Saurous for encouraging and supporting this work and for important conversations, and Yohai bar Sinai, Anton Geraschenko, Yifan Chen and Jiawei Zhuang for important conversations.

Appendix A: Direct numerical simulation

Here we describe the details of the numerical solver that we use for data generation, model comparison and the starting point of our machine learning models. Our solver uses a staggered-square mesh [54] 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 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 [55]. For the results presented in the paper we used explicit time integration 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 [56] or a real-valued fast Fourier transform (FFT). The former is well suited for small simulation domains as it has better accelerator utilization, while FFT has 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 ((A1)) where Δ is the grid spacing and $C_s = 0.2$:

$$\tau_{ij} = -2\left(C_s\Delta\right)^2|\bar{S}|\bar{S}_{ij} \tag{A1}$$
$$\bar{S}_{ij} = \frac{1}{2}(\partial_i u_j + \partial_j u_i) \qquad |\bar{S}| = 2\sqrt{\sum_{i,j}\bar{S}_{ij}\bar{S}_{ij}}$$

Appendix B: 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*. Note that besides the spatial downsampling we also perform downsampling in time to maintain the Courant–Friedrichs–Lewy (CFL) factor fixed at 0.5, standard for numerical simulations. Parameters specifying all five datasets are shown in Table A1. 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. We note that when performing simulations at higher Reynolds numbers we used rescaled grids to maintain fixed time stepping. This is motivated to potentially allow methods to specialize on a discrete time advancements. 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 A, when measuring the performance of all solvers on a given resolution we choose solver components to maximize efficiency.

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 ($t = 0$)	4
LES $Re = 10^5$	$2048 \to 64$	10^5	40

TABLE A1. 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.

Appendix C: Learned interpolations

To improve numerical accuracy of the convective process our models use psuedo-linear models for interpolation [34, 35]. 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. We do so 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 nullspace 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 [34, 35]. 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.

Appendix D: Neural network architectures

All of the ML based models used in this work are based on fully convolutional architectures. Fig. A1 depicts our three modeling approaches (pure ML, learned interpolation and learned correction) and three architecturs for neural network sub-components (Basic ConvNet,

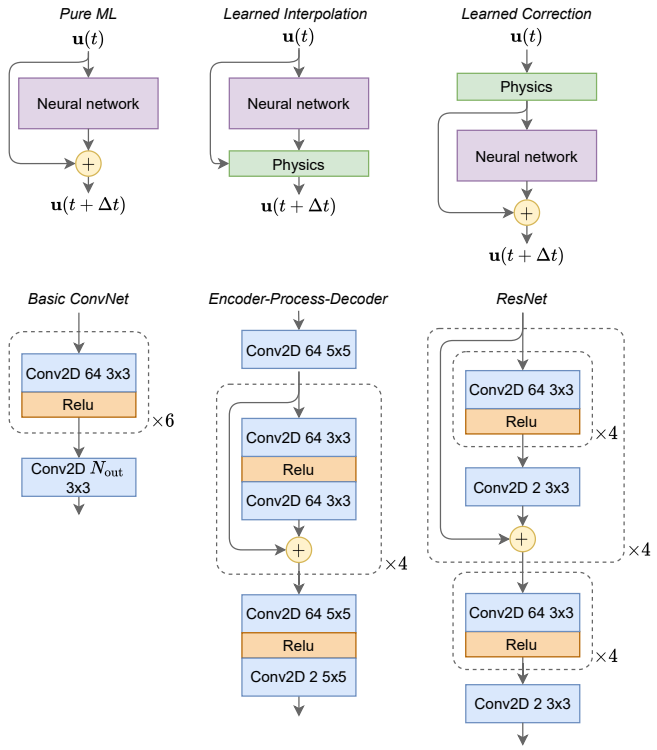

FIG. A1. Modeling approaches and neural network architecutres used in this paper. Rescaling of inputs and outputs from neural network sub-modules with fixed constants is not depicted. Dashed lines surround components that are repeatedly applied the indicated number of times.

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, 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.

Appendix E: 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. A2 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. A3 (and Fig. 5 in the main text), the improvement for 2× larger domains is identical to that found on a smaller domain.

Appendix F: 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.

Appendix G: Hyperparameter tuning

All models were trained using Adam [57] 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 t_T 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.

[1] L. F. Richardson, *Weather prediction by numerical process* (Cambridge university press, 2007).

[2] P. Bauer, A. Thorpe, and G. Brunet, The quiet revolution of numerical weather prediction, Nature **525**, 47 (2015).

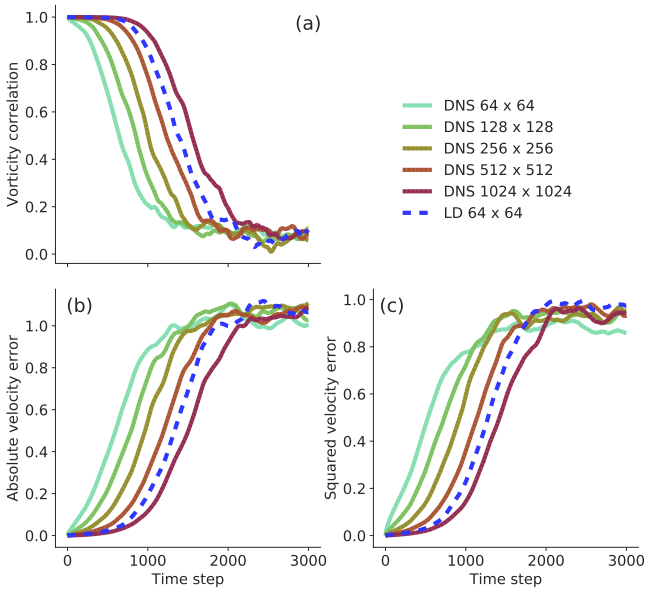

FIG. A2. Comparison of ML + CFD model to DNS running at different resolutions using varying metrics. (a) Vorticity correlation as presented in the main text. (b) Mean absolute error. (c) Absolute error of the kinetic energy.

[3] T. Schneider, J. Teixeira, C. S. Bretherton, F. Brient, K. G. Pressel, C. Schär, and A. P. Siebesma, Climate goals and computing the future of clouds, Nature Climate Change **7**, 3 (2017).

[4] P. Neumann, P. Düben, P. Adamidis, P. Bauer, M. Brück, L. Kornblueh, D. Klocke, B. Stevens, N. Wedi, and J. Biercamp, Assessing the scales in numerical weather and climate predictions: will exascale be the rescue?, Philosophical Transactions of the Royal Society A **377**, 20180148 (2019).

[5] J. Anderson, Basic philosophy of cfd, in *Computational Fluid Dynamics* (Springer, 2009) pp. 3–14.

[6] A. Bakhshaii and E. A. Johnson, A review of a new generation of wildfire–atmosphere modeling, Canadian Journal of Forest Research **49**, 565 (2019).

[7] W. M. Tang and V. Chan, Advances and challenges in computational plasma science, Plasma physics and controlled fusion **47**, R1 (2005).

[8] S. B. Pope, *Turbulent flows* (Cambridge University Press, 2000).

[9] U. Frisch, *Turbulence: the legacy of AN Kolmogorov* (Cambridge University Press, 1995).

[10] R. D. Moser, S. W. Haering, and G. R. Yalla, Statistical properties of subgrid-scale turbulence models, Annual Review of Fluid Mechanics **53** (2020).

[11] C. Meneveau and J. Katz, Scale-invariance and turbulence models for large-eddy simulation, Annual Review of Fluid Mechanics **32**, 1 (2000).

[12] J. Boussinesq, Theorie de l'ecoulement tourbillant, Mémoires de l'Académie des Sciences **23**, 46 (1877).

[13] G. Alfonsi, Reynolds-Averaged Navier–Stokes equations for turbulence modeling, Appl. Mech. Rev. **62** (2009).

[14] J. Smagorinsky, General circulation experiments with the primitive equations: I. the basic experiment, Mon. Weather Rev. **91**, 99 (1963).

[15] M. Lesieur and O. Metais, New trends in Large-Eddy simulations of turbulence, Annu. Rev. Fluid Mech. **28**, 45 (1996).

[16] Q. Malé, G. Staffelbach, O. Vermorel, A. Misdariis, F. Ravet, and T. Poinsot, Large eddy simulation of Pre-Chamber ignition in an internal combustion engine, Flow Turbul. Combust. **103**, 465 (2019).

[17] P. Wolf, G. Staffelbach, L. Y. M. Gicquel, J.-D. Müller, and T. Poinsot, Acoustic and large eddy simulation studies of azimuthal modes in annular combustion chambers, Combust. Flame **159**, 3398 (2012).

[18] L. Esclapez, P. C. Ma, E. Mayhew, R. Xu, S. Stouffer, T. Lee, H. Wang, and M. Ihme, Fuel effects on lean blow-out in a realistic gas turbine combustor, Combust. Flame **181**, 82 (2017).

[19] C. P. Arroyo, P. Kholodov, M. Sanjosé, and S. Moreau, CFD modeling of a realistic turbofan blade for noise prediction. part 1: Aerodynamics, in *Proceedings of Global Power and Propulsion Society* (2019).

[20] S. B. Pope, Ten questions concerning the large-eddy simulation of turbulent flows, New Journal of Physics **6** (2004).

[21] J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. **807**, 155 (2016).

[22] K. Duraisamy, G. Iaccarino, and H. Xiao, Turbulence modeling in the age of data, Annual Review of Fluid Mechanics **51**, 357 (2019).

[23] R. Maulik, O. San, A. Rasheed, and P. Vedula, Subgrid modelling for two-dimensional turbulence using neural networks, Journal of Fluid Mechanics **858**, 122 (2019).

[24] A. Beck, D. Flad, and C.-D. Munz, Deep neural networks for data-driven les closure models, Journal of Computational Physics **398**, 108910 (2019).

[25] 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*, Vol. 38 (Wiley Online Library, 2019) pp. 59–70.

[26] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv preprint arXiv:2003.03485 (2020).

[27] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, Model reduction and neural networks for parametric pdes, arXiv preprint arXiv:2005.03180 (2020).

[28] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu, Towards physics-informed deep learning for turbulent flow prediction, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020) pp. 1457–1466.

[29] B. Lusch, J. N. Kutz, and S. L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nature communications **9**, 1 (2018).

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

[31] J. Sirignano, J. F. MacArt, and J. B. Freund, Dpm: A deep learning pde augmentation method with application to large-eddy simulation, Journal of Computational Physics **423**, 109811 (2020).

[32] K. Um, P. Holl, R. Brand, N. Thuerey, *et al.*, Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers, in *34th Conference on Neural*

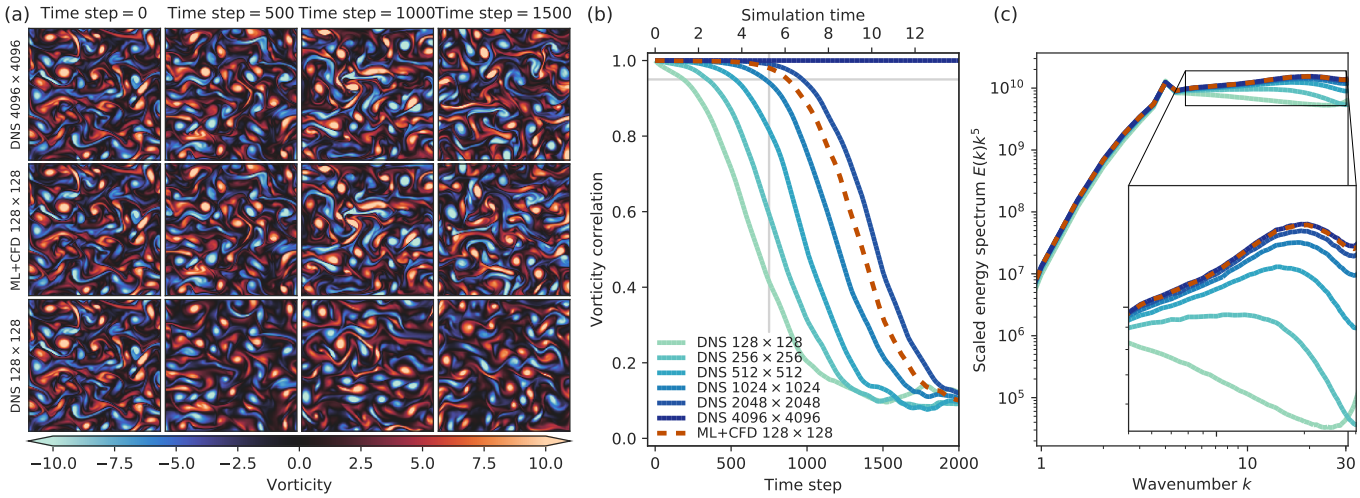

FIG. A3. Comparison of ML + CFD model to DNS running at different resolutions when evaluated on a 2× larger domain with characteristic length-scales matching those of the training data. (a) Visualization of the vorticity at different times. (b) Vorticity correlation as a function of time. (c) Scaled energy spectrum $E(k) * k^5$.

Information Processing Systems (NeurIPS 2020) (2020) p. in press.

[33] J. Pathak, M. Mustafa, K. Kashinath, E. Motheau, T. Kurth, and M. Day, Using machine learning to augment coarse-grid computational fluid dynamics simulations, arXiv preprint arXiv:2010.00072 (2020).

[34] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, Learning data-driven discretizations for partial differential equations, Proceedings of the National Academy of Sciences **116**, 15344 (2019).

[35] J. Zhuang, D. Kochkov, Y. Bar-Sinai, M. P. Brenner, and S. Hoyer, Learned discretizations for passive scalar advection in a 2-d turbulent flow, arXiv preprint arXiv:2004.05477 (2020).

[36] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, Jax: composable transformations of python+ numpy programs, 2018, URL http://github. com/google/jax , 18 (2020).

[37] L. Li, S. Hoyer, R. Pederson, R. Sun, E. D. Cubuk, P. Riley, and K. Burke, Kohn-Sham equations as regularizer: building prior knowledge into machine-learned physics, Phys. Rev. Lett. , in press (2020).

[38] S. S. Schoenholz and E. D. Cubuk, JAX, M.D.: A framework for differentiable physics, in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)* (2020) p. in press.

[39] K. Duraisamy, Machine learning-augmented reynolds-averaged and large eddy simulation models of turbulence, (2020), arXiv:2009.10675 [physics.flu-dyn].

[40] A. Griewank, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, Optim. Methods Softw. **1** (1994).

[41] G. J. Chandler and R. R. Kerswell, Invariant recurrent solutions embedded in a turbulent two-dimensional kolmogorov flow, J. Fluid Mech. **722**, 554 (2013).

[42] G. Boffetta and R. E. Ecke, Two-Dimensional turbulence, Annu. Rev. Fluid Mech. **44**, 427 (2012).

[43] In our case the Pearson correlation reduces to a cosine distance because the flows considered here have mean velocity of 0.

[44] Z. Ben-Haim, V. Anisimov, A. Yonas, V. Gulshan, Y. Shafi, S. Hoyer, and S. Nevo, Inundation modeling in data scarce regions, (2019), arXiv:1910.05006 [cs.LG].

[45] K. Yang, Y.-F. Chen, G. Roumpos, C. Colby, and J. Anderson, High performance monte carlo simulation of ising model on TPU clusters, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19 No. Article 83 (Association for Computing Machinery, New York, NY, USA, 2019) pp. 1–15.

[46] T. Lu, Y.-F. Chen, B. Hechtman, T. Wang, and J. Anderson, Large-Scale discrete fourier transform on TPUs, (2020), arXiv:2002.03260 [cs.MS].

[47] A. N. Kolmogorov, The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers, Cr Acad. Sci. URSS **30**, 301 (1941).

[48] R. H. Kraichnan and D. Montgomery, Two-dimensional turbulence, Reports on Progress in Physics **43**, 547 (1980).

[49] G. Boffetta and R. E. Ecke, Two-dimensional turbulence, Annual Review of Fluid Mechanics **44**, 427 (2012).

[50] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.

[51] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261 (2018).

[52] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia, Learning to simulate complex physics with graph networks, arXiv preprint arXiv:2002.09405 (2020).

[53] P. Bauer, A. Thorpe, and G. Brunet, The quiet revolution of numerical weather prediction, Nature **525**, 47 (2015).

[54] J. M. McDonough, Lectures in computational fluid dynamics of incompressible flow: Mathematics, algorithms and implementations, (2007).

[55] P. K. Sweby, High resolution schemes using flux limiters for hyperbolic conservation laws, SIAM journal on numerical analysis **21**, 995 (1984).

[56] R. E. Lynch, J. R. Rice, and D. H. Thomas, Direct solution of partial difference equations by tensor product methods, Numer. Math. **6**, 185 (1964).

[57] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).