Learning data driven discretizations for partial differential equations

Yohai Bar-Sinai*†1, Stephan Hoyer*‡2, Jason Hickey[2], and Michael P. Brenner[1,2]

[1]School of Engineering and Applied Sciences, Harvard University, Cambridge, MA
[2]Google Research, 1600 Amphitheatre Pkwy, Mountain View CA 94043

Abstract

The numerical solution of partial differential equations (PDEs) is challenging because of the need to resolve spatiotemporal features over wide length and timescales. Often, it is computationally intractable to resolve the finest features in the solution. The only recourse is to use approximate coarse-grained representations, which aim to accurately represent long-wavelength dynamics while properly accounting for unresolved small scale physics. Deriving such coarse grained equations is notoriously difficult, and often *ad hoc*. Here we introduce *data driven discretization*, a method for learning optimized approximations to PDEs based on actual solutions to the known underlying equations. Our approach uses neural networks to estimate spatial derivatives, which are optimized end-to-end to best satisfy the equations on a low resolution grid. The resulting numerical methods are remarkably accurate, allowing us to integrate in time a collection of nonlinear equations in one spatial dimension at resolutions 4-8x coarser than is possible with standard finite difference methods.

Solutions of nonlinear partial differential equations can have enormous complexity, with nontrivial structure over a large range of length and timescales. Developing effective theories that integrate out short length scales and fast time scales is a long standing goal. As examples, geometric optics is an effective theory of Maxwell equations at scales much longer than the wavelength of light [1]; Density Functional Theory models the full many-body quantum wavefunction with a lower dimensional object – the electron density field [2]; and the effective viscosity of a turbulent fluid parametrizes how small scale features affect large scale behavior [3]. These models derive their coarse-grained dynamics by more or less systematic integration of the underlying governing equations (by using, respectively, WKB theory, Local Density Approximation and a closure relation for the Reynold stress). The gains from coarse graining are, of course, enormous. Conceptually, it allows a deep understanding of emergent phenomena that would otherwise be masked by irrelevant details. Practically, it allows computation of vastly larger systems.

Averaging out unresolved degrees of freedom invariably replaces them by effective parameters that mimic typical behavior. In other words, we identify the salient features of the dynamics at short-and-fast scales and replace these with terms that have a similar average effect on the long-and-slow scales. Deriving reliable effective equations is often challenging [4]. Here we approach this challenge from the perspective of statistical inference. The coarse-grained representation of the function contains only partial information about it, since short scales are not modeled. Deriving coarse-grained dynamics requires first inferring the small scale structure using the partial information (reconstruction) and then incorporating its effect on the coarse-grained field. We propose to perform reconstruction using machine-learning algorithms, which have become extraordinarily efficient at identifying and reconstructing recurrent patterns in data. Having reconstructed the fine features, modeling their effect can be done using our physical knowledge about the system. We call our method *data-driven discretization*. It is qualitatively different from coarse graining techniques that are currently in use: instead of analyzing equations of motion to derive effective behavior, we directly learn from high resolution solutions to these equations.

Related work Several related approaches for computationally extracting effective dynamics have been previously introduced. Classic works used neural networks for discretizing dynamical systems [5, 6]. Similarly, equation-free modeling approximates coarse-scale derivatives by remapping coarse initial conditions to fine scales which are integrated exactly [7]. The method has similar spirit to our approach, but it does not learn from fine-scale dynamics and use the memorized statistics in subsequent times to reduce the computational load. Recent works have applied machine learning to PDEs, either focusing on speed [8–10] or recovering unknown dynamics [11, 12]. Models focused on speed often replace the slowest component of a physical model with machine learning, e.g., the solution of Poisson's equation in incompressible fluid simulations [9], sub-grid cloud models in climate simulations [10], or building reduced order models that approximate dynamics in a lower dimensional space [8, 13, 14]. These approaches are promising, but learn higher-level components than our proposed method. An important development is the ability to satisfy some physical constraints exactly by plugging learned models into a fixed equation of motion. For example, valid fluid dynamics can be guaranteed by learning either velocity fields directly [12] or a vector potential for velocity in the case of incompressible dynamics [8]. Closely related to this work, neural networks can be used to calculate closure conditions for coarse grained turbulent flow models [15, 16]. However, these models rely on existing coarse-grained schemes specific to turbulent flows and do not discretize the equations directly. Lastly, [17] suggested discretizations whose solutions can be analytically guaranteed to converge to the center manifold of the governing equation, but not in a data-driven manner.

*YBS and SH contributed equally to this work
†ybarsinai@gmail.com
‡shoyer@google.com

1 Data driven sub-grid scale modeling

Consider a generic PDE, describing the evolution of a continuous field $v(x,t)$

$$\frac{\partial v}{\partial t} = F\left(t, x, v, \frac{\partial v}{\partial x_i}, \frac{\partial v}{\partial x_i \partial x_j}, \cdots\right) \ . \tag{1}$$

Most PDEs in the exact sciences can be cast in this form, including equations that describe hydrodynamics, electrodynamics, chemical kinetics and elasticity. A common algorithm to numerically solve such equations is the method of lines [18]: given a spatial discretization $x_1, \cdots, x_N$, the field $v(x,t)$ is represented by its values at node points $v_i(t) = v(x_i, t)$ (finite differences), or by its averages over a grid cell, $v_i(t) = \Delta x^{-1} \int_{x_i - \Delta x/2}^{x_i + \Delta x/2} v(x', t) dx'$ (finite volumes), where $\Delta x = x_i - x_{i-1}$ is the spatial resolution [19]. The time evolution of v_i can be computed directly from Eq. (1) by approximating the spatial derivatives at these points. There are various methods for this approximation – polynomial expansion, spectral differentiation, etc. – all yielding formulas resembling

$$\frac{\partial^n v}{\partial x^n} \approx \sum_i \alpha_i^{(n)} v_i \ , \tag{2}$$

where the $\alpha_i^{(n)}$ are precomputed coefficients. For example, the one dimensional (1D) finite difference approximation for $\frac{\partial v}{\partial x}$ to first-order accuracy is $\partial_x v(x_i) = \frac{v_{i+1} - v_i}{\Delta x} + \mathcal{O}(\Delta x)$.

Standard schemes use one set of pre-computed coefficients for all points in space, while more sophisticated methods alternate between different sets of coefficients according to local rules [20, 21]. This discretization transforms Eq. (1) into a set of coupled ordinary differential equations of the form

$$\frac{\partial v_i}{\partial t} = F\left(t, x, v_1, \cdots, v_N\right) \ , \tag{3}$$

that can be numerically integrated using standard techniques. The accuracy of the solution to Eq. (3) depends on Δx, converging to a solution of Eq. (2) as $\Delta x \to 0$. Qualitatively, accuracy requires that Δx is smaller than the spatial scale of the smallest feature of the field $v(x,t)$.

However, the scale of the smallest features is often orders of magnitude smaller than the system size. High performance computing has been driven by the ever increasing need to accurately resolve smaller scale features in PDEs. Even with petascale computational resources, the largest direct numerical simulation of a turbulent fluid flow ever performed has Reynolds number of order 1,000, using about 5×10^{11} grid points [22–24]. Simulations at higher Reynolds number require replacing the physical equations with effective equations that model the unresolved physics. These equations are then discretized and solved numerically, e.g., using the method of lines. This overall procedure essentially modifies Eq. (2), by changing the α_i to account for the unresolved degrees of freedom, replacing the discrete equations in Eq. (3) with a different set of discrete equations.

The main idea of this work is that unresolved physics can instead be learned directly from data. Instead of deriving an approximate coarse-grained continuum model and discretizing it, we suggest directly learning low-resolution discrete models that encapsulate unresolved physics. Rigorous mathematical work shows that the dimension of a solution manifold for a nonlinear PDE is finite [25, 26], and that approximate parameterizations can be constructed [27–29]. If we knew the solution

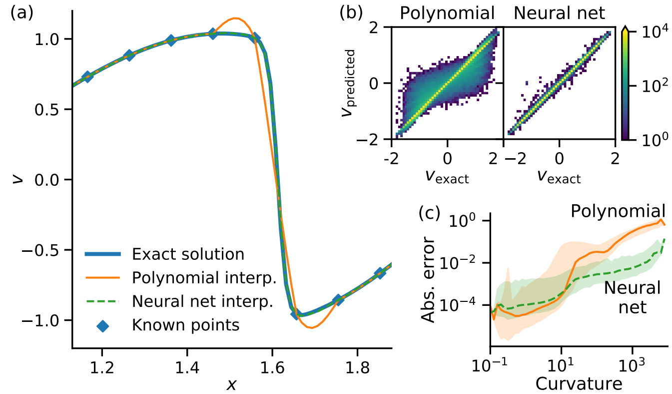

Figure 1: **Polynomial vs. neural net based interpolation** (a) Interpolation between known points (blue diamonds) on a segment of a typical solution of Burgers' equation. Polynomial interpolation exhibits spurious "overshoots" in the vicinity of shock fronts. These errors compound when integrated in time, such that a naive finite-difference method at this resolution quickly diverges. In contrast, the neural network interpolation is so close to the exact solution that it cannot be visually distinguished. (b) Histogram of exact vs. interpolated function values over our full validation dataset. The neural network vastly reduces the number of poor predictions. (c) Absolute error vs. local curvature. The bold line shows the median and shaded region shows the central 90% of the distribution over the validation set. The neural network makes much smaller errors in regions of high curvature, which correspond to shocks.

manifold we could generate *equation specific* approximations for the spatial derivatives in Eq. (2), approximations that have the potential to hold even when the system is under-resolved. In contrast to standard numerical methods, the coefficients $\alpha_i^{(n)}$ are equation-dependent. Different regions in space (e.g., inside and outside a shock) will use different coefficients. To discover these formulae, we use machine learning: we first generate a training set of high resolution data, and then learn the discrete approximations to the derivatives in Eq. (2) from this dataset. This produces a trade off in computational cost, which can be alleviated by carrying out high resolution simulations on small systems to develop local approximations to the solution manifold, and using them to solve equations in much larger systems at significantly reduced spatial resolution.

Burgers' equation For concreteness, we demonstrate this approach with a specific example in one spatial dimension. Burgers' equation is a simple nonlinear equation which models fluid dynamics in 1D and features shock formation. In its conservative form, it is written as:

$$\frac{\partial v}{\partial t} + \frac{\partial}{\partial x} J\left(v, \frac{\partial v}{\partial x}\right) = f(x,t) \ , \quad J \equiv \frac{v^2}{2} - \eta \frac{\partial v}{\partial x}, \tag{4}$$

where $\eta > 0$ is the viscosity and $f(x,t)$ is an external forcing term. J is the called a *flux*. Generically, solutions of Eq. (4) spontaneously develops sharp shocks, with specific relationships between the shock height, width and velocity [19] that define the local structure of the solution manifold.

With this in mind, consider a typical segment of a solution to Burgers' equation (Fig. 1(a)). We would like to compute the time derivative of the field given a low-resolution set of points (blue diamonds in Fig. 1). Standard finite difference formulas predict this time derivative by approximating v as a piecewise-polynomial function passing through the given points (orange curves in Fig. 1). But solutions to Burger's equations are not polynomials, they are shocks with characteristic properties. By using this information, we can derive a more accurate, albeit equation specific, formula for the spatial derivatives. For the method to work it should be possible to reconstruct the fine-scale solution from low resolution data. To this end, we ran many simulations of Eq. (4) and used the resulting data to train a neural network. Fig. 1 compares the predictions of our neural net (details below and in SI Appendix) to 4th order polynomial interpolation. This learned model is clearly far superior to the polynomial approximation, demonstrating that the spatial resolution required for parameterizing the solution manifold can be greatly reduced with equation-specific approximations rather than finite differences.

2 Models for time integration

The natural question to ask next is whether such parameterizations can be used for time integration. For this to work well, integration in time must be numerically stable, and our models need a strong generalization capacity: even a single error could throw off the solution for later times.

To achieve this, we use multi-layer neural networks to parametrize the solution manifold, because of their flexibility, including the ability to impose physical constraints and interpretability through choice of model architecture. The high-level aspects of the network's design, which we believe are of general interest, are described below. Additional technical details are described in the SI Appendix and source code is available online at `https://github.com/google/data-driven-discretization-1d`.

Pseudo-linear representation Our network represents spatial derivatives with a generalized finite-difference formula similar to Eq. (2): the output of the network is a list of coefficients $\alpha_1, \dots, \alpha_N$ such that the n-th derivative is expressed as a pseudo-linear filter, Eq. (2), where the coefficients $\alpha_i^{(n)}(v_1, v_2, \dots)$ depend on space and time through their dependence on the field values in the neighboring cells. Finding the optimal coefficients is the crux of our method.

The pseudo-linear representation is a direct generalization of the finite-difference scheme of Eq. (2). Moreover, exactly as in the case of Eq. (2), a Taylor expansion allows us to guarantee formal polynomial accuracy. That is, we can impose that approximation errors decay as $\mathcal{O}(\Delta x^m)$ for some $m \leq N - n$, by layering a fixed affine transformation (see SI). We found the best results when imposing linear accuracy, $m = 1$ with a 6-point stencil ($N = 6$), which we used for all results shown here. Lastly, we note that this pseudo-linear form is also a generalization of the popular ENO and WENO methods [20, 21], which choose a local linear filter (or a combination of filters) from a precomputed list according to an estimate of the solution's local curvature. WENO is an efficient, human-understandable, way of adaptively choosing filters, inspired by nonlinear approximation theory. We improve on WENO by replacing heuristics with directly optimized quantities.

Physical constraints Since Burgers' equation is an instance of the continuity equation, as with traditional methods, a major increase in stability is obtained when using a finite-volume scheme, ensuring the coarse-grained solution satisfies the conservation law implied by the continuity equation. That is, coarse-grained equations are derived for the cell averages of the field v, rather than its nodal values [19]. During training we provide the cell average to the network as the "true" value of the discretized field.

Integrating Eq. (4), it is seen that the change rate of the cell averages is completely determined by the fluxes at cell boundaries. This is an exact relation, in which the only challenge is estimating the flux given the cell averages. Thus, prediction is carried out in three steps: first, the network reconstructs the spatial derivatives on the boundary between grid cells (staggered grid). Then, the approximated derivatives are used to calculate the flux J using the exact formula Eq. (4). Lastly, the temporal derivative of the cell averages is obtained by calculating the total change at each cell by subtracting J at the cell's left and right boundaries. The calculation of the time derivative from the flux can also be done using traditional techniques that promote stability, such as monotone numerical fluxes [19]. For some experiments, we use Godunov flux, inspired by finite-volume ENO schemes [20, 21], but it did not improve predictions for our neural networks models.

Dividing the inference procedure into these steps is favorable in a few aspects: First, it allows to constrain the model at the various stages using traditional techniques: the conservative constraint, numerical flux and formal polynomial accuracy constraints are what we use here, but other constraints are also conceivable. Second, this scheme limits the machine-learning part to reconstructing the unknown solution at cell boundaries, which is the main conceptual challenge, while the rest of the scheme follows either the exact dynamics or traditional approximations for it. Third, it makes the trained model more interpretable since the intermediate outputs (e.g., J or α_i) have clear physical meaning. Lastly, these physical constraints contribute to more accurate and stable models, as detailed in the ablation study in the SI Appendix.

Choice of loss The loss of a neural net is the objective function minimized during training. Rather than optimizing the prediction accuracy of the spatial derivatives, we optimize the accuracy of the resulting time derivative[1]. This allows us to incorporate physical constraints in the training procedure and directly optimize the final predictions rather than intermediate stages. Our loss is the mean squared error between the predicted time derivative and labeled data produced by coarse graining the fully-resolved simulations.

Note that a low value of our training loss is a necessary but not sufficient condition for accurate and stable numerical integration over time. Many models with low training loss exhibited poor stability when numerically integrated (e.g., without the conservative constraint), particularly for equations with low dissipation. From a machine learning perspective, this is unsurprising: imitation learning approaches, such as our models, often exhibit such issues because the distribution of inputs

[1] For one specific case, namely the constant-coefficient model of Burgers' Equation with Godunov flux limiting, trained models showed poor performance (e.g., not monotonically increasing with resample factor) unless the loss explicitly included the time-integrated solution, as done in [9]. Results shown in Figs. 3 & 4 use this loss for the constant coefficient models with Burgers' Equation. See details in the SI.

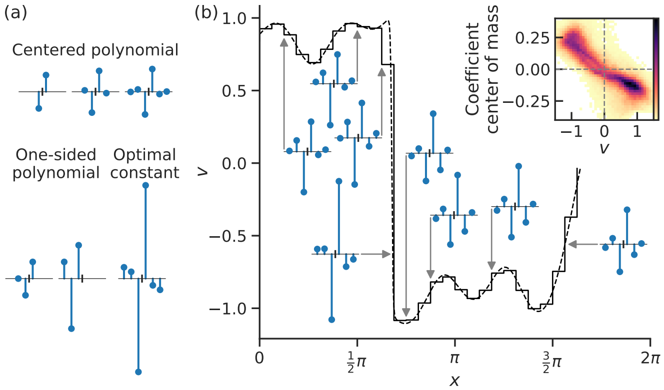

Figure 2: **Learned finite volume coefficients for Burgers' equation.** Fixed and spatiotemporally varying finite volume coefficients $\alpha_1^{(1)}, \ldots, \alpha_6^{(1)}$ (see Eq. (2)) for $\partial v/\partial x$. (a) Various centered and one-sided polynomial finite volume coefficients, along with optimized constant coefficients trained on this dataset (16x resample factor in Figure 3). The vertical scale, which is the same for all coefficient plots, is not shown for clarity. (b) An example temporal snapshot of a solution to Burgers' equation [Eq. (4)], along with data-dependent coefficients produced by our neural network model at each of the indicated positions on cell boundaries. The continuous solution is plotted as a dashed line, and the discrete cell-averaged representation is plotted as a piecewise constant solid line. The optimized constant coefficients are most similar to the neural network's coefficients at the shock position. Away from the shock, the solution resembles centered polynomial coefficients. (Inset) Relative probability density for neural network coefficient "center of mass" vs. field value v across our full test dataset. Center of mass is calculated by averaging the positions of each element in the stencil, weighted by the absolute value of the coefficient.

produced by the model's own predictions can differ from the training data [30]. Incorporating the time-integrated solution into the loss improved predictions in some cases (as in [9]), but did not guarantee stability, and could cause the training procedure itself to diverge due to decreased stability in calculating the loss. Stability for learned numerical methods remains an important area of exploration for future work.

Learned coefficients We consider two different parameterizations for learned coefficients. In our first parametrization, we learn optimized time- and space-independent coefficients. These fixed coefficients minimize the loss when averaged over the whole training set for a particular equation, without allowing the scheme to adapt the coefficients according to local features of the solution. Below, we refer to these as "optimized constant coefficients". In our second parametrization, we allow the coefficients to be an arbitrary function of the neighboring field values $\{v_i\}$, implemented as a fully-convolutional neural network [31]. We use the exact same architecture (three layers, each with 32 filters, kernel size of 5 and ReLU nonlinearity) for coarse-graining all equations discussed in this work.

Example coefficients predicted by our trained models are shown in Fig. 2 and SI Appendix, Fig. S3. Both the optimized constant and data-dependent coefficients differ from baseline polynomial schemes, particularly in the vicinity of the shock. The neural network solutions are particularly interesting: they do not appear to be using one-sided stencils near the shock, in contrast to traditional numerical methods such as WENO [21] which avoid placing large weights across discontinuities.

The output coefficients can also be interpreted physically. For example, coefficients for both $\partial v/\partial x$ (Fig. 2(b) inset) and v (SI Appendix, Fig. S3c) are either right- or left-biased, opposite the sign of v. This is in line with our physical intuition: Burgers' equation describes fluid flow, and the sign of v corresponds to the direction of flow. Coefficients that are biased in the opposite direction of v essentially look "upwind," a standard strategy in traditional numerical methods for solving hyperbolic PDEs [19], which helps constrain the scheme from violating temporal causality. Alternatively, upwinding could be built into the model structure by construction, as we do in models which use Godunov flux.

3 Results

Burgers' equation To assess the accuracy of the time integration from our coarse grained model, we computed "exact" solutions to Eq. (4) for different realizations of $f(x,t)$ at high enough resolution to ensure mesh convergence. These realizations of f were drawn from the same distribution as the those used for training, but were not in the training set. Then, for the same realization of the forcing we solved the equation at a lower, possibly under-resolved resolution using four different methods for calculating the flux: (a) a standard finite volume scheme with either 1st order or 3rd order accuracy; (b) 5th order upwind-biased WENO scheme with Godunov flux [21]; (c) spatial derivatives estimated by constant optimized coefficients, with and without Godonov flux; and (d) spatial derivatives estimated by the space- and time-dependent coefficients, computed with a neural net.

Results are shown in Fig. 3. Panel (a) compares the integration results for a particular realization of the forcing for different values of the *resample factor*, that is, the ratio between the number of grid points in the low resolution calculation and that of the fully converged solution[2]. Our learned models, both with constant and solution-dependent coefficients, can propagate the solution in time and dramatically outperform the baseline method at low resolution. Importantly, the ringing effect around the shocks, which leads to numerical instabilities, is practically eliminated.

Since our model is trained on fully resolved simulations, a crucial requirement for our method to be of practical use is that training can be done on small systems, but still produce models that perform well on larger ones. We expect this to be the case, since our models, being based on convolutional neural networks, use only local features and by construction are translation invariant. Panel (b) illustrates the performance of our model trained on the domain $[0, 2\pi]$ for predictions on a ten times larger spatial domain of size $[0, 20\pi]$. The learned model generalizes well. For example, it shows good performance by when function values are all positive in a region of size greater than 2π, which due to the conservation law cannot occur in the training dataset.

To make this assessment quantitative, we averaged over

[2]Physically, the natural measure of the spatial resolution is with respect to the internal length-scale of the equation which in the case of Burgers' equation is the typical shock width. However, since this analysis is meant to be applicable also to situations where the internal length-scale is a-priori unknown, we compare here to the length-scale at which mesh convergence is obtained.

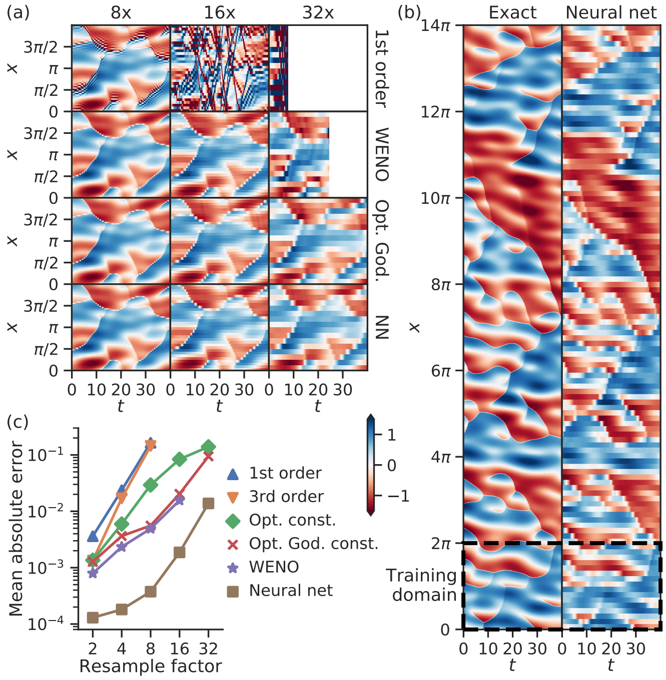

Figure 3: **Time integration results for Burgers' equation.** (a) A particular realization of a solution at varying resolution solved by the baseline 1st order finite volume method, WENO, optimized constant coefficients with Godunov flux, and the neural network, with the white region indicating times when the solution diverged. Both learned methods manifestly outperform the baseline method, and even outperform WENO at coarse resolutions. (b) Inference predictions for the 32x neural network model, on a ten times larger spatial domain (only partially shown). The box surrounded by the dashed line shows the spatial domain used for training. (c) Mean absolute error between integrated solutions and the ground truth, averaged over space, times less than 15, and 10 forcing realizations on the ten-times larger inference domain. These metrics almost exactly match results on the smaller training domain $[0, 2\pi]$ (Fig. S8). As ground truth, we use WENO simulations on a 1x grid. Markers are omitted if some simulations diverged or if the average error is worse than fixing $v = 0$.

many realizations of the forcing and calculated the mean absolute error integrated over time and space. Results on the ten-times larger inference domain are shown in panel (c): the solution from the full neural network has equivalent accuracy to increasing the resolution for the baseline by a factor of about 8x. Interestingly, even the simpler constant-coefficient method significantly outperforms the baseline scheme. The constant coefficient model with Godunov flux is particularly compelling. This model is faster than WENO, because there is no need to calculate coefficients on the fly, with comparable accuracy and better numerical stability at coarse resolution, as shown in panel (a) and Fig. 4.

These calculations demonstrate that neural networks can carry out coarse graining. Even if the mesh spacing is much larger than the shock width, the model is still able to accurately propagate dynamics over time, showing that it has learned an internal representation of the shock structure.

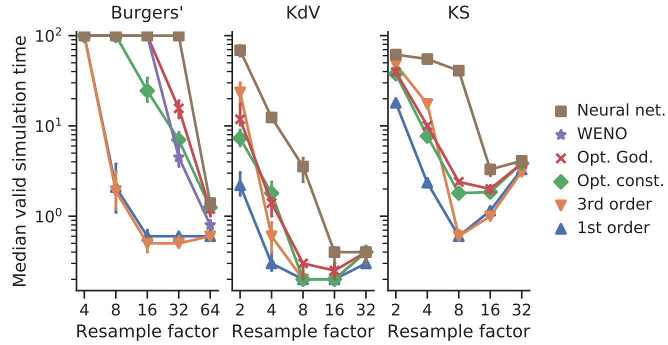

Figure 4: **Model performance across all of our test equations.** Each plot shows the median time for which an integrated solution remains "valid" for each equation, defined by the absolute error on at least 80% of grid points being less than the 20th percentile of the absolute error from predicting all zeros. These thresholds were chosen so that "valid" corresponds to relatively generous definition of an approximately correct solution. Error bars show the 95% confidence interval for the median across 100 simulations for each equation, determined by bootstrap resampling. Simulations for each equation were run out to a maximum of time of 100.

Other examples To demonstrate the robustness of this method, we repeated the procedure for two other canonical PDEs: the Korteweg-de Vries (KdV) equation [32], which was first derived to model solitary waves on a river bore and is known for being completely integrable and to feature soliton solutions; and the Kuramoto-Sivashinsky (KS) equation which models flame fronts and is a textbook example of a classically chaotic PDE [33]. All details about these equations are given in the SI. We repeated the training procedure outlined above for these equations, running high resolution simulations and collecting data to train equation-specific estimators of the spatial derivative based on a coarse grid. These equations are essentially non-dissipative, so we do not include a forcing term. The solution manifold is explored by changing the initial conditions, which are taken to be a superposition of long-wavelength sinusoidal functions with random amplitudes and phases (see SI for details).

To assess the accuracy of the integrated solution, for each initial condition we define "valid simulation time" as the first time that the low-resolution integrated solution deviates from the cell-averaged high-resolution solution by more than a given threshold. We found this metric more informative to compare across very different equations than absolute error.

Figure 4 shows the median valid simulation time as a function of the resample factor. For all equations and resolutions, our neural network models have comparable or better performance than all other methods. The neural network is particularly advantageous at low resolutions, demonstrating its improved ability to solve coarse-grained dynamics. The optimized constant coefficients perform better at coarse resolution than baseline methods, but not always at high resolutions. Finally, at large enough resample factors the neural network approximations also fails to reproduce the dynamics, as expected. These results also hold on a ten-times larger spatial domain, as shown in the SI, along with figures illustrating specific realizations and mean absolute error.

4 Discussion and conclusion

It has long been remarked that even simple nonlinear PDEs can generate solutions of great complexity. But even very complex, possibly chaotic, solutions are not just arbitrary functions: they are highly constrained by the equations they solve. In mathematical terms, despite the fact that the solution set of a PDE is nominally infinite dimensional, the inertial manifold of solutions is much smaller, and can be understood in terms of interactions between local features of the solutions to nonlinear PDEs. The dynamical rules for interactions between these features have been well studied over the past fifty years. Examples include, among many others, interactions of shocks in complex media, interactions of solitons [32], and the turbulent energy cascade [34].

Machine learning offers a different approach for modeling these phenomena, by using training data to parametrize the inertial manifold itself; said differently, it learns both the features and their interactions from experience of the solutions. Here we propose a simple algorithm for achieving this, motivated by coarse graining in physical systems. It is often the case that coarse graining a PDE amounts to modifying the weights in a discretized numerical scheme. Instead, we use known solutions to learn these weights directly, generating *data driven discretizations*. This effectively parametrizes the solution manifold of the PDE, allowing the equation to be solved at high accuracy with an unprecedented low resolution.

Faced with this success, it is tempting to try and leverage the understanding the neural network has developed in order to gain new insights about the equation or its coarse-grained representation. Indeed, in Fig. 2 we could clearly interpret the directionality of the weights as an upwind bias, the pseudolinear representation providing a clear interpretation of the prediction in a physically sensible way. However, extracting more abstract insight from network, such as the scaling relation between the shock height and width is a difficult challenge. This is a general problem in the field of machine learning, which is under intensive current research [35, 36].

Our results are promising, but two challenges remain before our approach can be deployed at large scales. The first challenge is speed. We showed that optimized constant coefficients can already improve accuracy, but our best models rely on the flexibility of neural networks. Unfortunately, our neural nets use many more convolution operations than the single convolution required to implement finite differences, e.g., $32^2 = 1024$ convolutions with a five point stencil between our second and third layers. We suspect that other machine learning approaches could be dramatically faster. For example, recent work on a related problem – inferring sub-pixel resolution from natural images – has shown that banks of pretrained linear filters can nearly match the accuracy of neural nets with orders of magnitude better performance [37, 38]. The basic idea is to divide input images into local patches, classify patches into classes based on fixed properties (e.g., curvature and orientation), and learn a single optimal linear filter for each class. Such computational architectures would also facilitate extracting physical insights from trained filters.

A second challenge is scaling to higher dimensional problems and more complex grids. Here we showcased the approach for regular grids in one dimension, but most problems in the real world are higher dimensional, and irregular and adaptive grids are common. We do expect larger potential gains in two and three dimensions, as the computational gain in terms of the number of grid points would scale like the square or the cube of the resample factor. Irregular grids may be more challenging, but deep learning methods that respect appropriate invariants have been developed both for arbitrary graphs [39] and collections of points in 3D space [40]. Similar to what we found here, we expect that hand-tuned heuristics for both gridding and grid coefficients could be improved upon by systematic machine learning. More broadly, data driven discretization suggests the potential of data driven numerical methods, combining the optimized approximations of machine learning with the generalization of physical laws.

Acknowledgements We thank Peyman Milanfar, Pascal Getreur, Ignacio Garcia Dorado and Dmitrii Kochkov for collaboration and important conversations, Peter Norgaard and Geoff Davis for feedback on drafts of the manuscript, and Chi-Wang Shu for guidance on the implementation of WENO. Y.B.S acknowledges support from the James S. McDonnel post-doctoral fellowship for the study of complex systems. M.P.B acknowledges support from NSF DMS-1715477 as well as the Simons Foundation.

References

[1] J. D. Jackson, *Classical Electrodynamics* (John Wiley & Sons, 1999).

[2] D. Sholl and J. A. Steckel, *Density functional theory: a practical introduction* (Wiley & Sons, 2011).

[3] C. J. Chen, *Fundamentals of turbulence modelling* (CRC Press, 1997).

[4] M. Van Dyke, NASA STI/Recon Technical Report A **75** (1975).

[5] R. Gonzalez-Garcia, R. Rico-Martinez, and I. Kevrekidis, Computers & chemical engineering **22**, S965 (1998).

[6] R. Rico-Martinez, I. Kevrekidis, and K. Krischer, Neural networks for chemical engineers , 409 (1995).

[7] I. G. Kevrekidis and G. Samaey, Annual Review of Physical Chemistry **60**, 321 (2009).

[8] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, arXiv Preprint 1806.02071 (2018).

[9] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, in *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, edited by D. Precup and Y. W. Teh (2017) pp. 3424–3433.

[10] S. Rasp, M. S. Pritchard, and P. Gentine, Proceedings of the National Academy of Sciences **115**, 9684 (2018), http://www.pnas.org/content/115/39/9684.full.pdf .

[11] S. L. Brunton, J. L. Proctor, and J. N. Kutz, Proceedings of the National Academy of Sciences **113**, 3932 (2016).

[12] E. de Bezenac, A. Pajot, and P. Gallinari, in *International Conference on Learning Representations* (2018).

[13] B. Lusch, J. N. Kutz, and S. L. Brunton, Nature Communications **9**, 4950 (2018).

[14] J. Morton, F. D. Witherden, A. Jameson, and M. J. Kochenderfer, in *Advances in Neural Information Processing Systems 31* (Curran Associates, Inc., 2018) pp. 9258–9268.

[15] J. Ling, A. Kurzawski, and J. Templeton, Journal of Fluid Mechanics **807**, 155 (2016).

[16] A. D. Beck, D. G. Flad, and C.-D. Munz, arXiv Preprint 1806.04482 (2018).

[17] A. Roberts, Applied numerical mathematics **37**, 371 (2001).

[18] W. E. Schiesser, *The numerical method of lines: integration of partial differential equations* (Academic Press San Diego, San Diego, 1991).

[19] R. J. LeVeque, *Numerical Methods for Conservation Laws* (Birkhauser Verlag, 1992).

[20] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy, Journal of computational physics , 231 (1987).

[21] C.-W. Shu, in *Advanced numerical approximation of nonlinear hyperbolic equations* (Springer, 1998) pp. 325–432.

[22] M. Lee and R. D. Moser, Journal of Fluid Mechanics **774**, 395 (2015).

[23] M. Clay, D. Buaria, T. Gotoh, and P. Yeung, Computer Physics Communications **219**, 313 (2017).

[24] K. P. Iyer, K. R. Sreenivasan, and P. K. Yeung, Physical Review E **95**, 021101 (2017).

[25] P. Constantin, C. Foias, B. Nicolaenko, and R. Temam, *Integral manifolds and inertial manifolds for dissipative partial differential equations*, Vol. 70 (Springer Science & Business Media, 2012).

[26] C. Foias, G. R. Sell, and R. Temam, Journal of differential equations **73**, 309 (1988).

[27] M. Jolly, I. Kevrekidis, and E. Titi, Physica D: Nonlinear Phenomena **44**, 38 (1990).

[28] E. S. Titi, Journal of mathematical analysis and applications **149**, 540 (1990).

[29] M. Marion, Journal of Dynamics and Differential Equations **1**, 245 (1989).

[30] S. Ross and D. Bagnell, in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, Vol. 9, edited by Y. W. Teh and M. Titterington (PMLR, Chia Laguna Resort, Sardinia, Italy, 2010) pp. 661–668.

[31] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning* (MIT press Cambridge, 2016).

[32] N. J. Zabusky and M. D. Kruskal, Physical Review Letters **15**, 240 (1965).

[33] D. Zwillinger, *Handbook of differential equations* (Gulf Professional Publishing, 1998).

[34] U. Frisch, *Turbulence: The Legacy of A. N. Kolmogorov* (Cambridge University Press, 1996).

[35] M. Sundararajan, A. Taly, and Q. Yan, arXiv Preprint 1703.01365 (2017).

[36] A. Shrikumar, P. Greenside, and A. Kundaje, in *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70 (PMLR, 2017) pp. 3145–3153.

[37] Y. Romano, J. Isidoro, and P. Milanfar, IEEE Transactions on Computational Imaging **3**, 110 (2017).

[38] P. Getreuer, I. Garcia-Dorado, J. Isidoro, S. Choi, F. Ong, and P. Milanfar, in *2018 IEEE International Conference on Computational Photography (ICCP)* (2018).

[39] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, in *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70 (PMLR, International Convention Centre, Sydney, Australia, 2017) pp. 1263–1272.

[40] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, in *Advances in Neural Information Processing Systems*, Vol. 30 (2017).

[41] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016) pp. 265–283.

[42] B. Fornberg, Mathematics of computation **51**, 699 (1988).

Supplemental Materials

Appendix I: Neural network model

Model details Complete source code, allowing production of a training dataset, training the network on it, and deployment of the resulting coarse-grained equation is freely available online at https://github.com/google/data-driven-discretization-1d. Our model, including all physical constraints, was implemented using the TensorFlow library [41]. In the calculations presented here, the model had three fully convolutional layers, each with 32 filters of a fixed kernel of size five and with a ReLU nonlinearity between each layer. Our neural network predictions at a single point are thus dependent on values of the local solution over a maximum range of 13 grid cells, independent of the model resolution. The code allows for easily tuning these hyper-parameters.

Fig. S1 presents a graphical depiction of our model architecture. The coarse grained function values are fed into a neural network. The network's output, the coefficients $\alpha_i^{(n)}$, are combined with the coarse grained values to estimate the spatial derivatives. These are used in the known physical equation for the flux, which is used to calculate the temporal derivative by a first-order divergence. Training minimizes either the difference between the calculated time derivative and the true one (most models), or the difference between the calculated evolved state at future times and the true evolved state (Burgers' equation with constant coefficients only, as noted below).

We trained our models using the Adam optimizer for 4.0000×10^4 steps total, decreasing the learning rate by a factor of 10 after 2.0000×10^4 steps. For most models we used an initial learning rate of 3×10^{-3}, with the exception of KdV and KS models with a resample factor of 16x and higher, for which we used an initial learning rate of 1×10^{-3}. We used a batch size of 128 times the resampling factor. All of our results show models trained with the time-derivative loss, with the exception of the "optimized constant coefficient" models for Burgers' equation, which was trained to predict 8 forward time-steps with the midpoint method. Each individual model trained to completion in less than an hour on a single Nvidia P100 GPU.

We found that some of our models had highly variable performance on different model training runs, due to randomness in the training procedure and a lack of guarantees of numerical stability in our training procedure. To ameliorate this issue and improve the interpretability of our results, we trained each model ten times and in most results only show predictions from only the best overall performing model for each task. Predictions from the worst-performing models are shown below in S10.

Godunov numerical flux For some models, we used Godunov numerical flux for the convective term v^2 in the flux. Following the example of numerical fluxes for WENO methods [21], we construct both left- and right-sided estimates of v (with separate α coefficients for each), v^- and v^+, and combine them according to the Godunov flux rule for $J(v) = v^2$:

$$J_{\text{godunov}}(v^-, v^+) = \begin{cases} \min[(v^-)^2, (v^+)^2] & \text{if } v^- \le v^+ \\ \max[(v^-)^2, (v^+)^2] & \text{if } v^- > v^+ \end{cases}. \tag{S-1}$$

Appendix II: PDE parameters

Equations We solved three PDEs in one space dimension x and time dimension t. All of our equations can be written in the same conservative form,

$$\frac{\partial v}{\partial t} + \frac{\partial J}{\partial x} = F(x,t) \ , \quad v(x, t=0) = v_0(x), \tag{S-2}$$

with different choices for the flux J, forcing F and initial conditions v_0:

$$\text{Burgers:} \quad J \equiv \frac{v^2}{2} - \eta \frac{\partial v}{\partial x}, \qquad F \equiv f(x,t), \qquad v_0 \equiv 0 \tag{S-3}$$

$$\text{Kortweg-de Vries (KdV):} \quad J \equiv 3v^2 + \frac{\partial^2 v}{\partial x^2}, \qquad F \equiv 0, \qquad v_0 \equiv f(x,0) \tag{S-4}$$

$$\text{Kuramoto-Shivashinski (KS):} \quad J \equiv \frac{v^2}{2} + \frac{\partial v}{\partial x} + \frac{\partial^3 v}{\partial x^3}, \qquad F \equiv 0, \qquad v_0 \equiv f(x,0). \tag{S-5}$$

where the function $f(x,t)$, described below, is initialized with random components to allow for sampling over a manifold of solution. The form of these equations is written to emphasize their similar structure; the standard form of KdV can be obtained by substituting $v \to -v$. All equations employ periodic boundary conditions. For training, we used domains of size $L = 2\pi$ for Burgers' equation, $L = 32$ for KdV and $L = 64$ for KS. For Burgers' equation, we set $\eta = 0.01$.

Random parameters In order to explore the solution manifold, we introduce randomness either to the forcing (Burgers' equation) or to the initial conditions (KS & KdV). The random signal that we use is a sum of long-wavelength sinusoidal functions:

$$f(x,t) = \sum_{i=1}^{N} A_i \sin(\omega_i t + 2\pi \ell_i x / L + \phi_i), \tag{S-6}$$

with the following parameters:

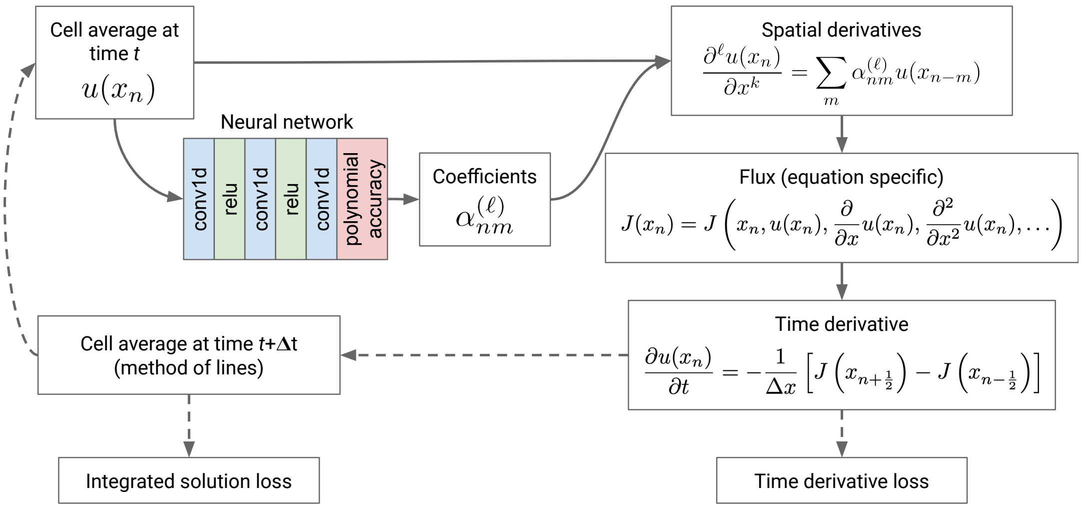

Figure S1: **Neural network architecture.** During training, the model is optimized to predict cell average time-derivatives or time evolved solution values from cell average values, based on a precomputed dataset of snapshots from high resolution simulations. During inference, the optimized model is repeatedly applied to predict time evolution using the method of lines.

	Burgers'	KdV and KS
A	$[-0.5, 0.5]$	
ω	$[-0.4, 0.4]$	
ϕ	$[0, 2\pi]$	
ℓ	$\{3, 4, 5, 6\}$	$\{1, 2, 3\}$
N	20	10

Each parameter is drawn independently and uniformly from its range. For generating Fig. 3(b) and 3(c) of the main text, i.e., deploying the trained model on a 10x larger domain, ℓ was allowed to take on integer value that would result in a wavenumber $k = 2\pi\ell/L$ within the range of the original training data, i.e., $\{30, 31, 32, \dots, 60\}$.

Training data To train the network we generate a set of 8000 high-resolution solutions to each equation, sampled at regular time intervals from 800 numerical integrations. Five randomly selected examples for each equation are shown in Fig. S2 To obtain high accuracy "exact" solutions to Burgers' equation on a fully resolved grid, we used a fifth-order WENO method and 512 grids points. For KdV and KS, we used a pseudo-spectral method for numerical differentiation (which we found to be more accurate than Finite Volumes), with 256 grid points. However, for calculating the loss during training we used third-order Finite Volumes on the spectral method's solution to calculate spatial derivatives. For unclear reasons, our learned models for KdV and KS at low resample factors did not perform well when using spectral derivatives for ground-truth. One possibility is that a lack of access to the global features used by the spectral methods made it impossible for our models (which are spatially local by construction) to exactly copy their behavior.

Time integration For numerical time integration to produce training and evaluation datasets, we used an explicit Runge-Kutta method of order 3(2) with adaptive time-stepping, as implemented in the SciPy package. For time integration during the training procedure itself, we used the midpoint method with a fixed time-step of 0.001 for Burgers' equation, based on the minimum time-step on the high resolution grid chosen by adaptive time-stepping. We used these methods due to their robustness and simplicity, but they are certainly *not* an optimized time integration methods for these equations. For example, a practical method for the viscous Burgers' equation should use implicit time-stepping, because the diffusive term limits the maximum stable explicit time step to be smaller than the square of the grid spacing. Nonetheless, we believe this is appropriate because the focus of this paper is not on improving the time discretization.

Appendix III: Polynomial accuracy constraints

Our method represents the spatial derivative at a point x_0 as a linear combination of the function values at N neighbors of x_0. That is, for a given derivative order ℓ, we write

$$\left.\frac{\partial^\ell f}{\partial x^\ell}\right|_{x=x_0} = \sum_{n=1}^{N} \alpha_n f(x_0 + h_n) \tag{S-7}$$

where h_n is the offset of the n-th neighbor on the coarse grid. Note that in the main text we only deal with uniformly spaced meshes, $h_n = n\,\Delta x$, but this formalism holds for an arbitrary mesh. A crucial advantage of this writing is that we can enforce arbitrary polynomial accuracy up to degree m, as long as $m \le N - \ell$, by imposing an affine constraint on the α's. That is, we can ensure that the error in approximating $f^{(\ell)}$ will be of order h^m for some $m \le N - \ell$.

To see this, note that the standard formula for deriving the finite difference coefficients for the ℓ-th derivative with an N-

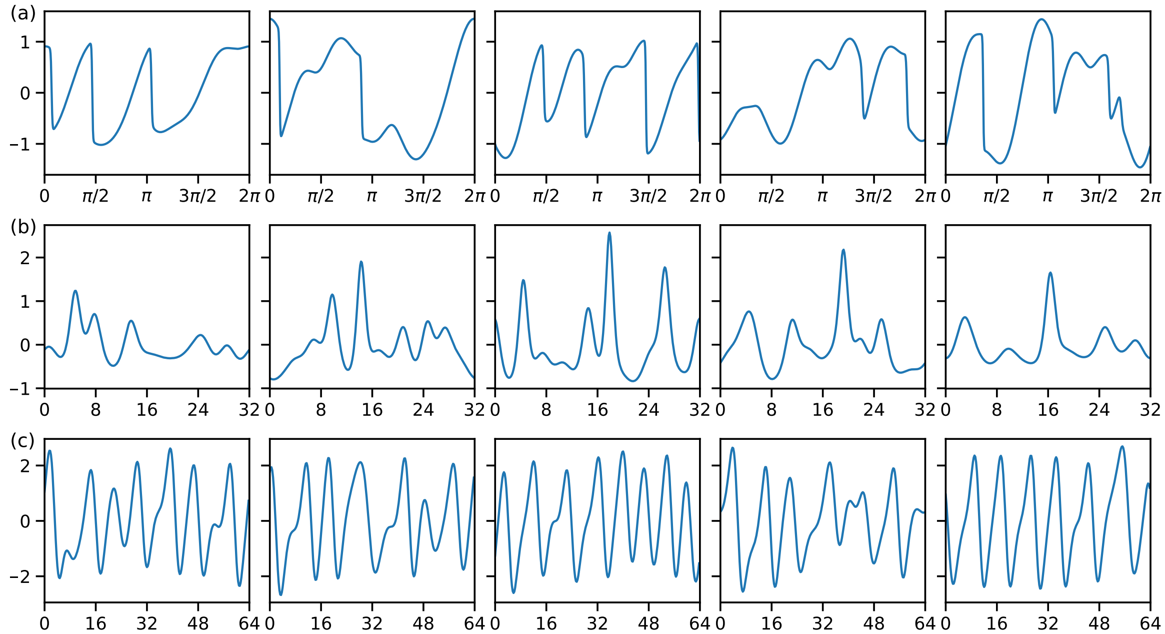

Figure S2: Five random samples from the training dataset each for (a) Burgers' equation, (b) KdV and (c) KS.

point stencil is obtained by solving the linear set of equations

$$\begin{pmatrix} h_1^0 & \cdots & h_N^0 \\ \vdots & \ddots & \vdots \\ h_1^{N-1} & \cdots & h_N^{N-1} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix} = \ell! \begin{pmatrix} \delta_{0,\ell} \\ \vdots \\ \delta_{i,\ell} \\ \vdots \\ \delta_{N-1,\ell} \end{pmatrix} \tag{S-8}$$

where $\delta_{i,j}$ is the Kronecker delta [42].

These equations are obtained by expanding Eq. (S-7) in a Taylor series up to order $m-1$ in the vicinity of x_0 and requiring equality of order $\mathcal{O}(h^m)$ for arbitrary functions. Each row in the set of equations corresponds to demanding that a term of order h^k will vanish, for $k = 0, 1, \ldots, N-1$. The resulting formula is approximate to polynomial order $N - \ell$ [42] and the system of equations is fully determined, that is, a unique solution exists, which is obtained by inverting the matrix. A similar set of linear constraints for finite volume methods can be derived by averaging over each unit cell [21].

Imposing a lower order approximation amounts to removing the bottom rows from the equation Eq. (S-8). Specifically, imposing accuracy of order m amounts to keeping the first $m - N + \ell$ rows, which makes the system under-determined. Therefore, any solution for $\boldsymbol{\alpha}$ can be written as a sum of an arbitrary fixed solution (say, the standard-finite difference formula of order m) plus a vector $\tilde{\boldsymbol{\alpha}}$ from the null-space of the matrix of Eq. (S-8) (with removed rows).

Appendix IV: Interpolation model

The neural network model for interpolation of Burgers' equation shown in Fig. 1 of the main text uses the same training datasets and a similar model structure to our time integration models. Instead of predicting cell average values, we train the model to predict the subsampled function values at all intermediate locations from a high-resolution simulation as zeroth order spatial derivatives, imposing first-order accuracy constraints. We use an 8x subsampled grid, so the outputs of our model are seven interpolated values between each adjacent pair of seed points. We use the mean squared error at each interpolated point as the loss, scaled such that linear interpolation on the training dataset has a loss of one, and an initial learning rate of 1×10^{-3}. Otherwise, all details match the neural network models described in Appendix I. The fourth order polynomial interpolation between points x_i and x_{i+1} is based on the function values at $\{v_{i-1}, v_i, v_{i+1}, v_{i+1}\}$, which closely corresponds to the interpolation used by traditional finite difference (not finite volume) method.

For Fig. 1(c), we define the curvature of each point as the maximum value of $|\partial^2 v/\partial x^2|$ over the interpolated interval on the high resolution grid. This heuristic, which is similar to the those used for identifying smooth regions by WENO methods [21], provides a good indicator of whether or not the solution is interpolating over a shock front.

Appendix V: Additional plots of coefficients

Figure S3 shows learned finite volume coefficients for reconstructing the field value v for Burgers' equation. Similar to the situation for the coefficients for $\partial v/\partial x$ shown in Fig. 2, the neural network has learned upwinding.

Figure S4 shows optimized constant coefficients for Burgers', KdV and KS equations at all resample factors for which the optimized coefficients showed any improvement over standard coefficients. The optimized coefficients often, but not always, differ significantly from standard centered coefficients, particularly for lower order derivatives.

Figure S5 shows optimized constant coefficients for all equations using the Godunov flux. For both Burgers' and KS, the optimized coefficients make use of "upwinding," especially at coarser resolutions.

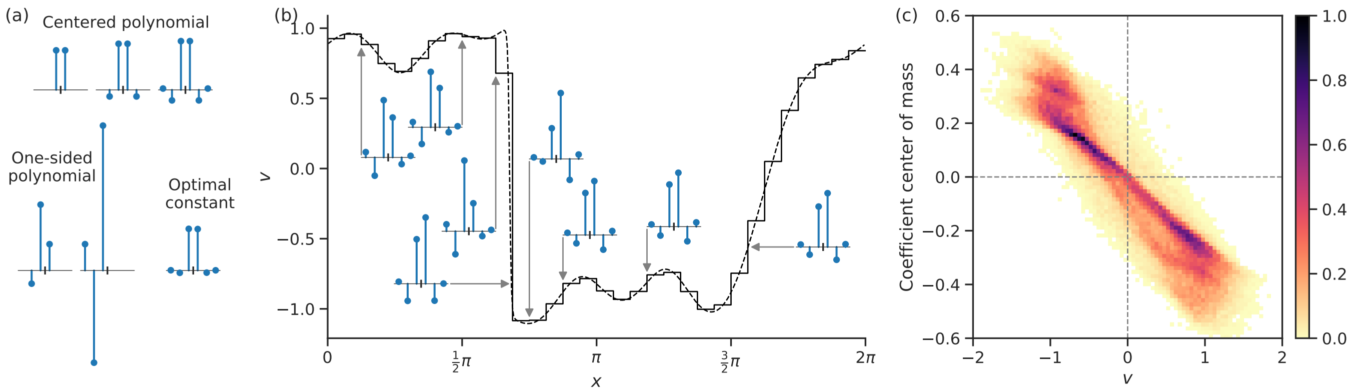

Figure S3: Like Fig. 2 of the main text, but showing coefficients for reconstructing the field value v instead of its first derivative.

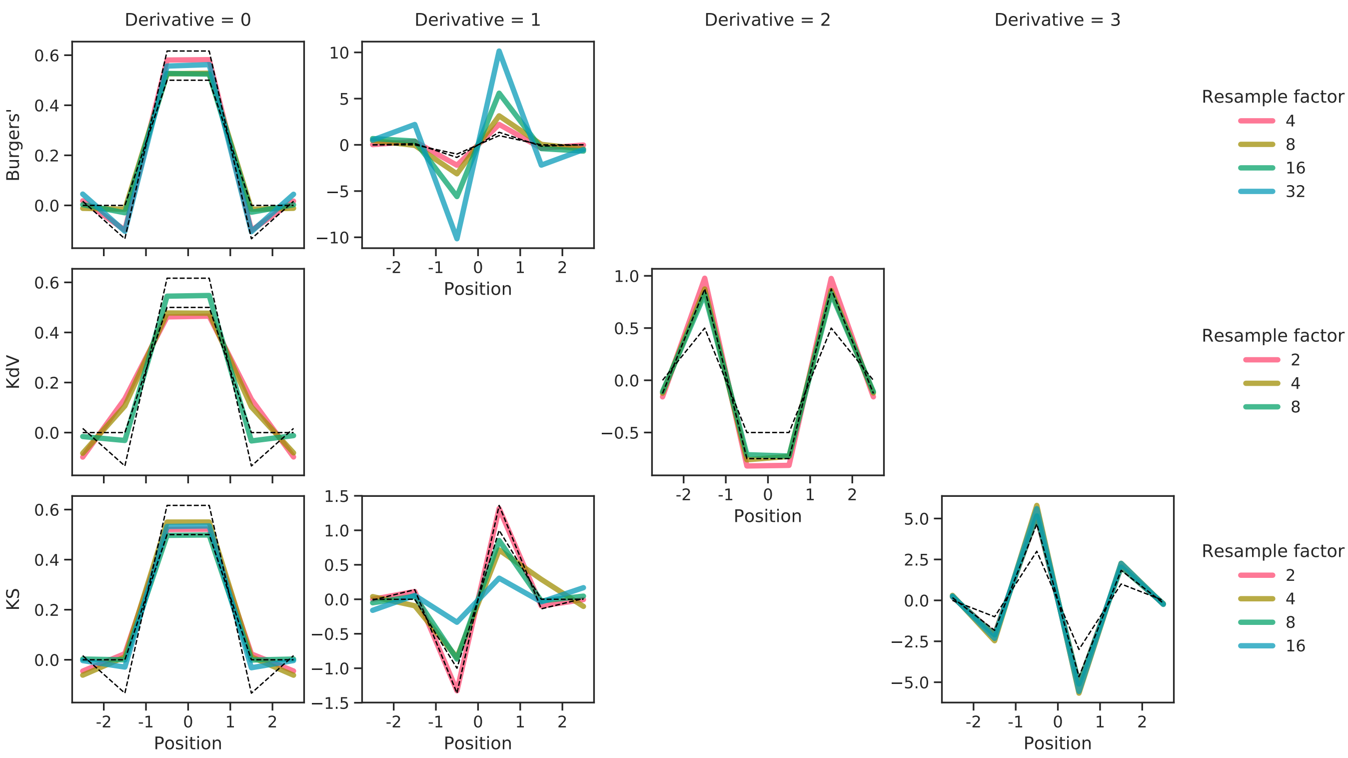

Figure S4: Optimized constant finite volume coefficients for the equations studied in this work at varying resample factor. The dashed black lines correspond to standard centered Finite Volumes coefficients: 2-point and 6-point stencils for zeroth and first derivatives, and 4-point and 6-point stencils for second and third derivatives.

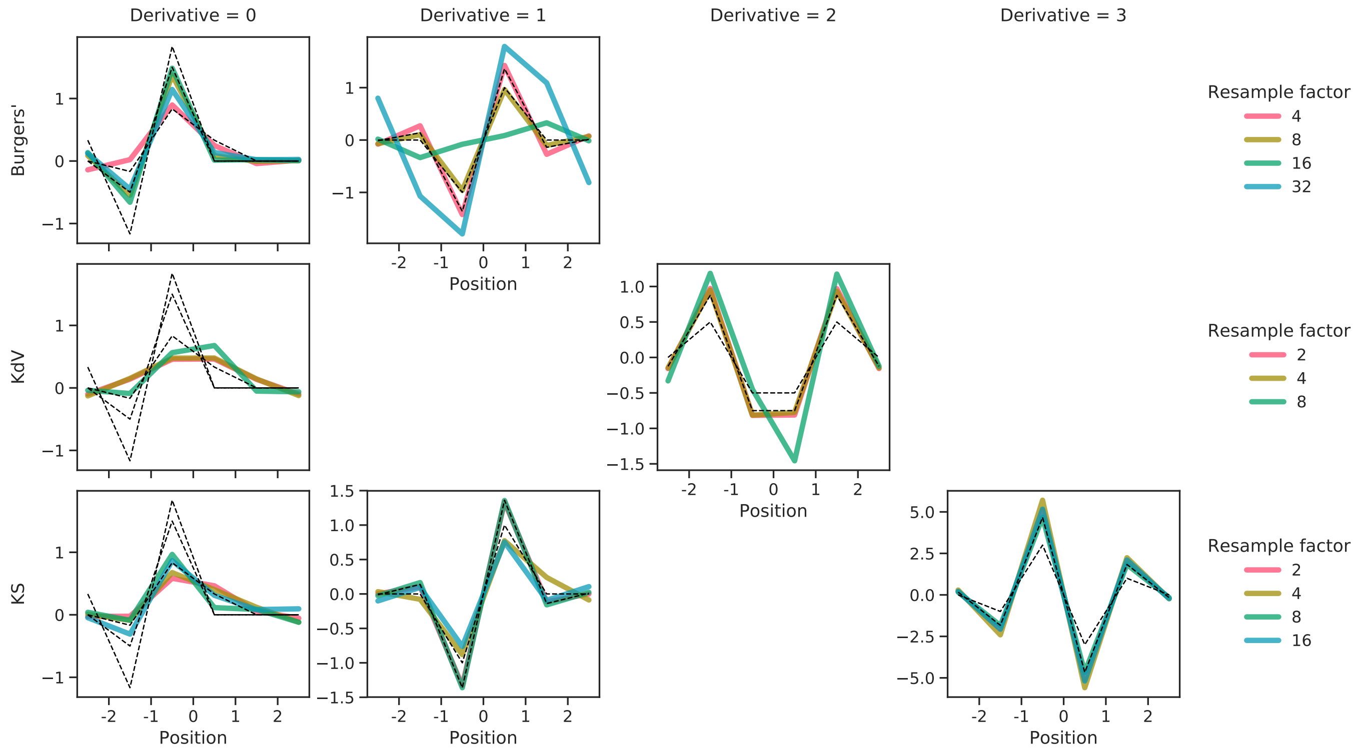

Figure S5: Like Figure S4, but for models that use Godunov monotone flux to combine left- and right-sided estimates u^- and u^+. Only the left-ward facing coefficients u^- for reconstructing the zeroth derivative are shown, along standard 2 and 3 point upwinded Finite Volume coefficients.

Appendix VI: Results for KdV and KS equations

Similar to Fig. 3(a) and 3(b) in the main text, Figures S6 and S7 show particular realizations of solutions to the KdV and KS Equations, respectively, on both the a domain of the same size as the training domain and on a 10x larger validation domain (for the 8x resample factor model). The figures show how the same initial condition is solved at different resample factors by the baseline finite-difference method and neural network, demonstrating that our method significantly outperforms the baseline. They also show that our models for KdV and KS also generalize to a much larger spatial domain than used for training.

Similar to Fig. 3(c) in the main text, Fig. S8 shows mean absolute error for coarse grained models of KdV and KS at different resample factors, averaged over 100 realizations of each model on the training spatial domain, and 10 realizations of each model on a ten-times larger spatial domain. It is only possible to compare mean absolute error for short times, because at long times bad models diverge and the mean becomes undefined. Similar to Fig. 4 in the main text, Figure S9 shows median survival time for all models on the ten-times larger domain. The ranking of models in both figures is broadly consistent with Fig. 4 .

Appendix VII: Ablation study

To understand the importance of various physical constraints in our model architecture, we performed an ablation study, comparing various modeling choices with the same neural network models. In order of increasing level of physical constraints, these include:

- "Time derivative" models that predict the time derivative $\partial u/\partial t$ directly, without any incorporation of physical prior knowledge at all.
- "Flux" models that predict the flux J at the boundary between grid cells and use that to compute the time derivative with the continuity equation, without any knowledge of the physical equation.
- "Space derivatives" models that predict spatial derivatives without any constraints, and plug those spatial derivatives into the physical equation to compute the flux.
- "Coefficients" models that predict linear coefficients used in finite difference formula for computing space derivatives.
- "Constrained coefficients" models that predict linear coefficients constrained to at least first-order accuracy.
- "Godunov constrained coefficients" models that predict constrained coefficients, and additionally use the Godunov numerical flux for the convective term in the flux.

In addition, we also experimented with finite-difference modeling (coarse-graining by subsampling) rather than the finite-volume modeling (coarse-graining by averaging) used in all of the above. Generally speaking, finite difference models can also work but they perform worse than finite volumes, as with

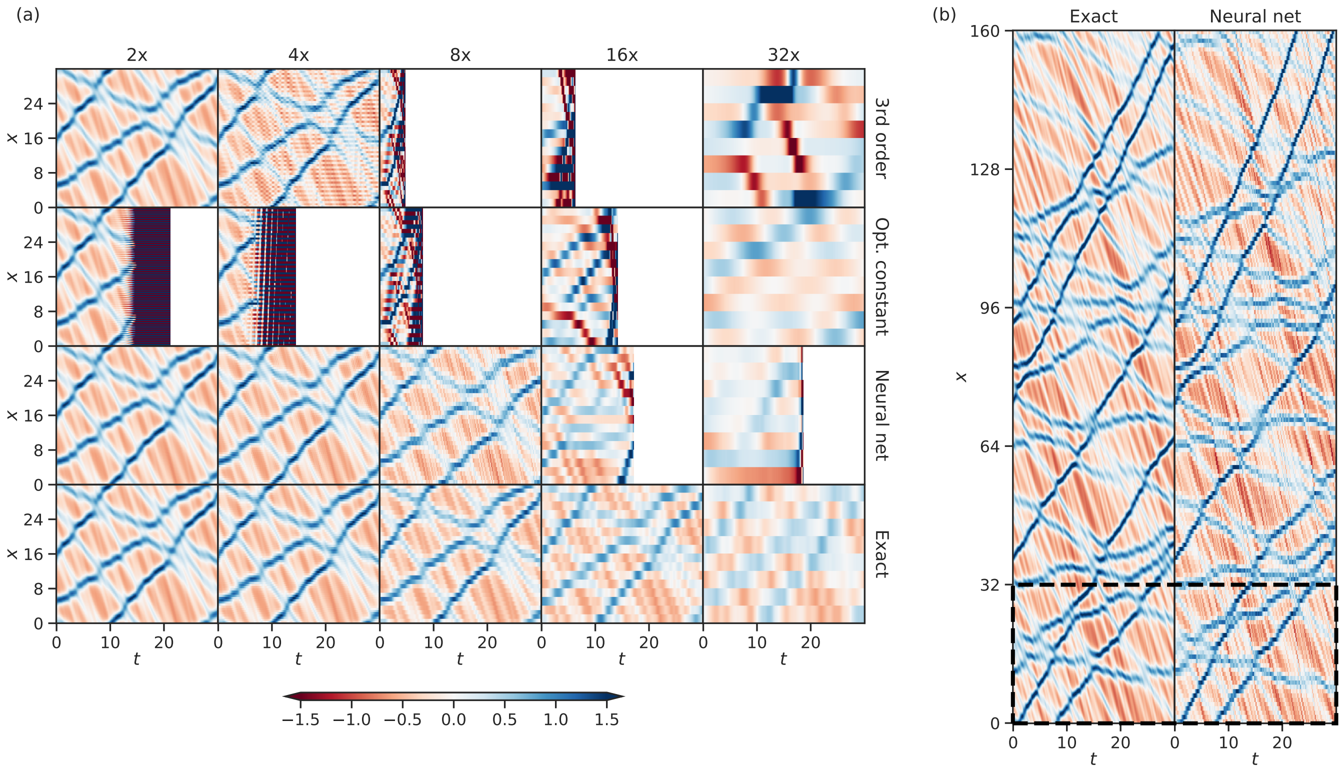

Figure S6: (a) Particular realization of the solution for the Korteweg-de Vries (KdV) equation at varying resolution solved by the baseline 1st order finite volume method (top row), optimal constant coefficients (second row), the neural network (third row) and the exact coarse-grained solution (bottom row). Blank regions indicate where the solver has diverged. Note that optimal constant coefficients exhibit lower error at small times than the baseline coefficients, even though the 2x and 4x models suffer from worse numerical stability. (b) Inference predictions for the 8x neural network model, on a ten times larger spatial domain (only partially shown). The box surrounded by the dashed line shows the spatial extent of the training domain.

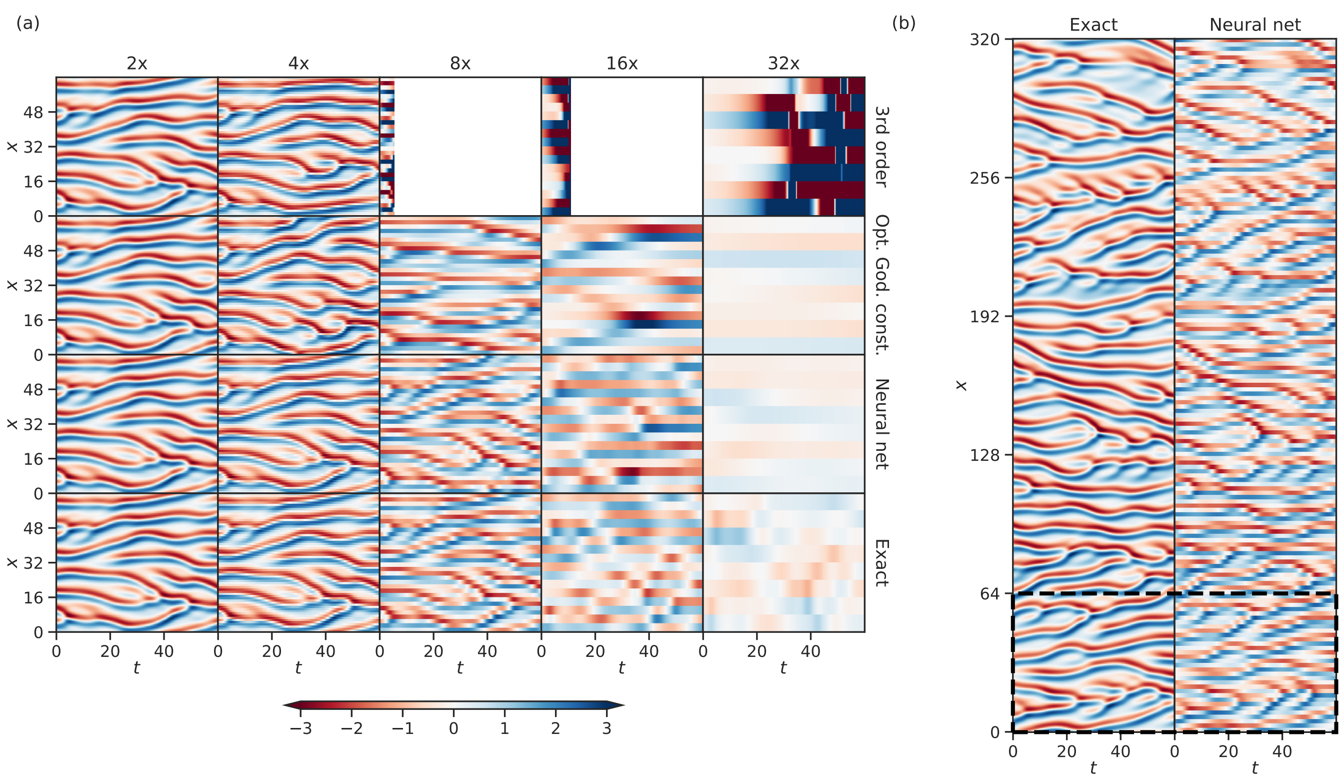

Figure S7: Same as S6, but for the Kuramoto-Shivashinsky (KS) equation.

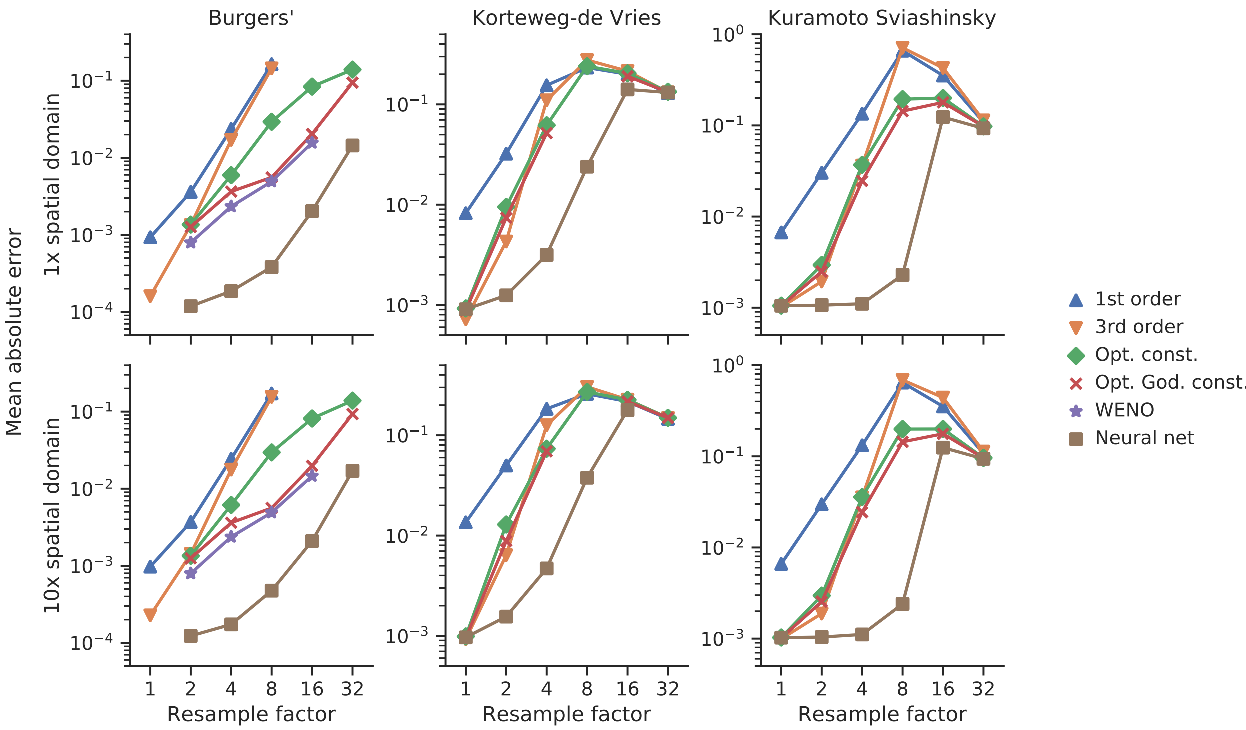

Figure S8: Mean absolute error for Burgers', KdV and KS at short times ($t \leq 15$, $t \leq 1$ and $t \leq 3$, respectively) on both the 1x spatial domain used for training and the 10x larger domain only used for inference. The 10x domain results for Burgers' equation in the bottom left panel are duplicated for comparison purposes from Fig. 3(c) of the main text.

traditional numerical methods. We do not report these results here for brevity.

The results, shown in Fig. S10, suggest a number of trends:

1. Consistent results across all equations and grid resolutions were only achieved with the "constrained coefficients" and "Godunov constrained coefficients" models. For Burgers' and KS, these models achieved good results on all training runs.

2. Physical constraints dramatically improve the performance of trained models on fine-resolution grids, but have a smaller influence on results for low-resolution (large resample factor) grids. This makes sense for two reasons: First, low resample factor is closer to the continuum limit, for which these physical constraints and the equations themselves hold exactly. Second, due to the CFL condition fine-resolution models use a smaller timestep. Therefore, when integrated for the same fixed amount of time, the fine-resolution models are repeatedly applied more times than low resolution ones, making them more sensitive to numerical stability.

3. Building in the Godunov flux does not directly improve the predictions of neural network models. Apparently Eq. (S-1) is simple enough that it is easy for a neural network to learn, as shown in Fig. 2 in the main text and Fig. S3.

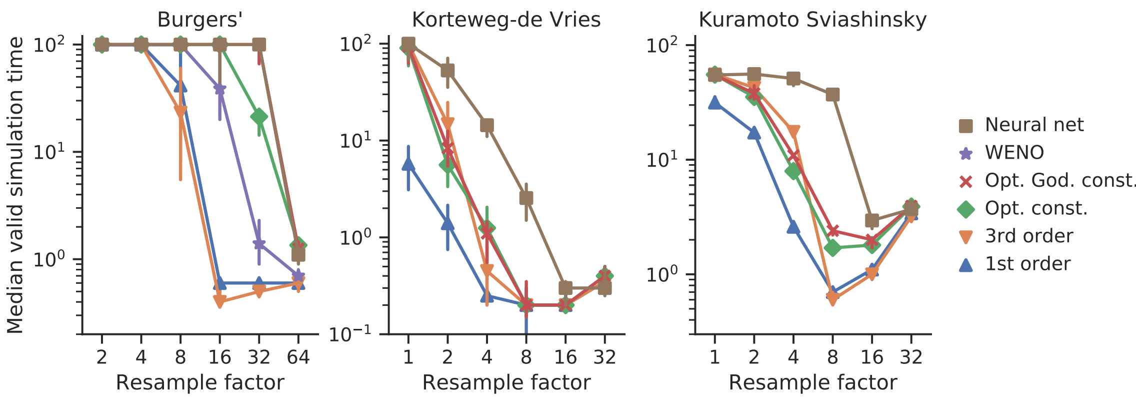

Figure S9: Survival times for Burgers', KdV and KS at short times on the 10x larger inference domain. Results for the optimized Godunov constant model for Burgers' equation are obscured in the plot by the results for the neural network model.

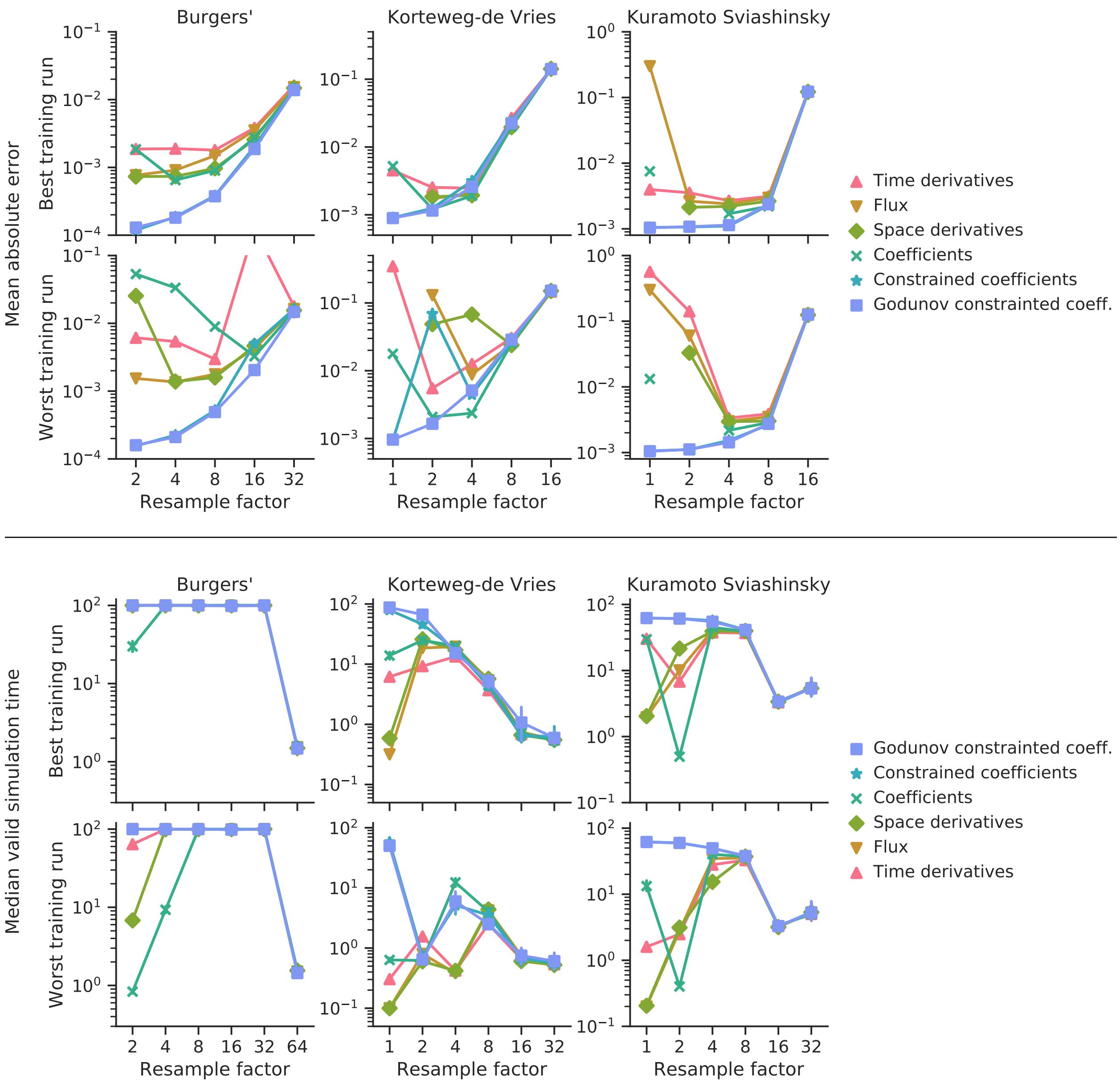

Figure S10: Mean absolute error and survival times for alternative neural network models. The "best" and "worst" training runs illustrate variability over ten random choices of initial conditions and shuffles of the training data.