HyperFNO: Improving the Generalization Behavior of Fourier Neural Operators

Francesco Alesiani* NEC Labs Europe
Makoto Takamoto, NEC Labs Europe
Mathias Niepert University of Stuttgart

Abstract

Physic-informed machine learning aims to build surrogate models for real-world physical systems governed by partial differentiable equations (PDEs). One of the more popular recently proposed approaches is the Fourier Neural Operator (FNO), which learns Green's function operator for PDEs based only on observational data. These operators are able to model PDEs for a variety of initial conditions and show the ability of multi-scale prediction. However, as we will show, this model class is not able to generalize to changes in the parameters of the PDEs, such as the viscosity coefficient or forcing term. We propose HYPERFNO, an approach combining FNOs with hypernetworks so as to improve the models' extrapolation behavior to a wider range of PDE parameters using a single model. HyperFNO learns to generate the parameters of functions operating in both the original and the frequency domain. The proposed architecture is evaluated using various simulation problems.

1 Introduction

Physical systems governed by differential equations are pervasive in the sciences and engineering. Indeed, partial differential equations (PDEs) are the most popular modeling method in numerous disciplines such as the material and climate sciences. On the other hand, the recent success of data-driven modeling with machine learning (ML) has started a trend to use ML methods, which are both differentiable and efficient at prediction time, for approximating simulations of physical systems and solving inverse problems. For instance, neural networks are now commonly used to approximate the solution of a partial differential equation or its Green's function (Avrutskiy, 2020; Karniadakis et al., 2021; Li et al., 2021; Raissi et al., 2019; Chen et al., 2018; Raissi, 2018; Raissi et al., 2018; Pfaff et al., 2020; Wang et al., 2020; Khoo et al., 2021).

Neural Operators (NOs) Li et al. (2020) and in particular, Fourier Neural Operators (FNOs) (Guibas et al., 2022; Li et al., 2021) have shown impressive results as surrogate models and have been applied in challenging scenarios such as weather forecasting (Pathak et al., 2022). Neural operators, however, work under the assumption that the governing PDE is fixed, that is, its parameters are static while the initial condition is what changes. If this assumption is not met, the accuracy of these approaches deteriorates (Mischaikow & Mrozek, 1995), also visible in Figure 3. Thus, when we are in a situation where we want to generalize to multiple physical model parametrizations, we would need to either (1) re-train the NO for each of the parameter configurations or (2) include the parameter values as input to the neural operator (Arthurs & King, 2021). Training over a large number of possible parametrizations is computationally demanding. On the other hand, increasing the number of parameters of the network increases the inference time which limits the efficiency advantage surrogate models have over numerical solvers.

*E-mail:Francesco.Alesiani@neclab.eu

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

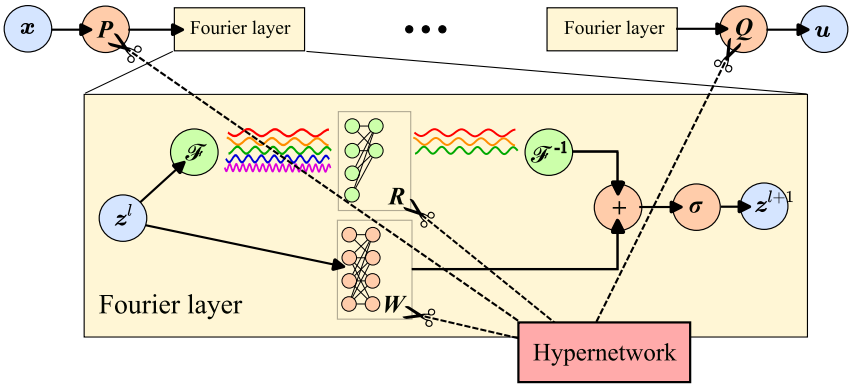


Figure 1: A HYPERFNO that implements M_θ^λ consists of a base neural operator and a hypernetwork that learns to generate parameters of subnetworks of the base model. HYPERFNO generates, for each layer of the base network, and depending on the configuration, the frequency and/or spatial weight matrices.

With this work, we formulate a meta-learning problem where each task corresponds to a parameter value of a given PDE. At inference time, we use the learned meta-model to adapt to the current task, that is, the given parameters of the PDE. To this end, we propose HYPERFNO (section 2), a method to adapt the Neural Operator over a wide range of parameter configurations, which uses Hypernetworks (Ha et al., 2016). We also propose variations of the HYPERFNO framework (section 2) with low memory requirements; while we show experimentally (section 3) why modeling the change in parameters of a PDE in the frequency domain is beneficial; Finally, since the HYPERFNO is differentiable with respect to the PDE parameters, we show how to solve inverse problems through stochastic gradient descent using HYPERFNO in section 2.

2 HYPERFNO: The Hyper Fourier Neural Operator

Problem definition A solution to a PDE is vector-valued function $\boldsymbol{u} : \mathcal{T} \times \mathcal{X} \times \Lambda \to \mathbb{R}$ on some temporal domain $\mathcal{T}$, spatial domain $\mathcal{X}$, and parameterized over Λ. For example in the heat diffusion equation, $\boldsymbol{u}$ could represent the temperature at a location $\boldsymbol{x} \in \mathcal{X}$ at a time $t \in \mathcal{T}$, where the conductivity field is defined by $\boldsymbol{\lambda} : \mathcal{X} \to \mathbb{R}_+$. The *forward operator* maps the solution at one instant of time to a future time step $F : \boldsymbol{u}(t, \boldsymbol{x}, \boldsymbol{\lambda}) \to \boldsymbol{u}(t+1, \boldsymbol{x}, \boldsymbol{\lambda})$, which is used to compute the solution of the PDE at any time, given the initial conditions. We consider the general problem of learning a class of operators, which includes the forward operator F. $M^\lambda : \mathcal{A} \times \Lambda \to \mathcal{U}$ between two infinite dimensional spaces of functions $\mathcal{A} : \mathbb{R}^d \to \mathbb{R}^p$ and $\mathcal{U} : \mathbb{R}^d \to \mathbb{R}^q$, on the space of parameters Λ, from a finite collection of observed data $\{\lambda_j, a_j, u_j\}_{i=1}^N, \lambda_j \in \Lambda, a_j \in \mathcal{A}, u_j \in \mathcal{U}$, composed of parameter-input-output triples, where $a_j \sim \mu$ and $\lambda_j \sim \rho$ are drawn from two known probability distributions μ over $\mathcal{A}$ and ρ over Λ; using a family of operators $M_\theta^\lambda : \mathcal{A} \times \Lambda \times \Theta \to \mathcal{U}$ parameterized by $\theta \in \Theta$. We then aim to minimize the expected loss $\min_\theta \mathbb{E}_{a\sim\mu,\lambda\sim\rho}\, \mathcal{L}(M_\theta^\lambda(a), M^\lambda(a))$, with $\mathcal{L}(u', u)$ measuring the difference between the true and predicted output.

Fourier Neural Operators Fourier Neural Operators (FNO) (Guibas et al., 2022; Li et al., 2021) are among the most successful Neural Operators since they model spatial and frequency domains. FNO implements a discrete version of $M_\theta : \mathcal{A} \times \Theta \to \mathcal{U}$, and is composed of initial and final projection networks parameterized by $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{Q}'$, and of a sequence of *Fourier layers*, parameterized by a tensor $\boldsymbol{W}$ and is implemented using a 1-d convolutional network for the spatial component, while parameterized by a tensor $\boldsymbol{R}$ for the frequency component. The FNO can be expressed as

$$\boldsymbol{z}^{l+1} = \sigma(\boldsymbol{W}^l \boldsymbol{z}^l + \mathscr{F}^{-1}(\boldsymbol{R}^l\, \mathscr{F}(\boldsymbol{z}^l))), \quad \boldsymbol{z}^0 = \boldsymbol{P}\boldsymbol{x}, \quad \boldsymbol{u} = \boldsymbol{Q}'\sigma(\boldsymbol{Q}\boldsymbol{z}^{L-1}), \tag{1}$$

with $\boldsymbol{z}^l$ the latent tensor at l-layer of proper dimension depending on the domain of the PDE, while $\boldsymbol{x}, \boldsymbol{u}$ are the input and output of the operator. The output projection is implemented using two consecutive fully connected (FC) layers, the input projection with one FC, and $\mathscr{F}$ is Fast Fourier Transform (FFT).

HYPERFNO: Hyper Networks for FNO Hypernetworks Ha et al. (2016) have been proposed as a meta-learning method composed of two networks: the main network ($f(\boldsymbol{\psi}, \boldsymbol{x})$), with parameters $\boldsymbol{\psi}$, and the hypernetwork[2] ($h(\boldsymbol{\theta}, \boldsymbol{\lambda})$),with parameters $\boldsymbol{\theta}$. The HYPERFNO network consists of a hypernetwork that generates a subset of the parameters of the FNO network. If we write a FNO as the function $f(\boldsymbol{\psi}, \boldsymbol{x})$, then the HYPERFNO can be written as $\boldsymbol{\psi}_{\boldsymbol{\lambda}} = h(\boldsymbol{\theta}, \boldsymbol{\lambda}),\ \ \hat{\boldsymbol{u}} = f(\boldsymbol{\psi}_{\boldsymbol{\lambda}}, \boldsymbol{x})$, where $\hat{\boldsymbol{u}}$ is the predicted solution given the PDE of parameters $\boldsymbol{\lambda}$ and initial condition $\boldsymbol{x}$, and $\boldsymbol{\psi}$ are the parameters of the FNO network generated by the hypernetwork. We train the hypernetwork end-to-end by minimizing the following expectation of a loss function $\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\lambda} \sim p(\boldsymbol{\lambda})}\, \mathcal{L}^{\text{tr}}_{\boldsymbol{\lambda}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$, where $\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{u}) \sim D^{\text{tr}}_{\boldsymbol{\lambda}}} ||\boldsymbol{u} - f(\boldsymbol{\psi}_{\boldsymbol{\lambda}}, \boldsymbol{x})||^2$, and $\boldsymbol{\psi}_{\boldsymbol{\lambda}} = h(\boldsymbol{\theta}, \boldsymbol{\lambda})$. $p(\boldsymbol{\lambda})$ is some given distribution over the PDE's parameters $\boldsymbol{\lambda}$, while $D^{\text{tr}}_{\boldsymbol{\lambda}}$ is the training dataset, conditioned by the PDE's parameter. Fig.1 shows an implementation of the HYPERFNO.

HYPERFNO hypernetwork architecture Hypernetworks are used to generate the parameters of a main neural network conditioned on the current task. Typically, the hypernetwork is a large neural network that generates the parameters of a smaller network. This approach increases the efficiency of the model at inference time. An alternative approach aims at using a hypernetwork that only marginally increases the size of the main network but still allows it to easily adapt to new tasks. We focus on the second approach and propose a special class of layers of the hypernetwork that can be used to assemble the main network. In the HYPERFNO, each layer of the FNO and its initial and final projection operators (Equation 1)are generated by the corresponding hypernetworks

$$\boldsymbol{W}^l = h_w(\boldsymbol{\theta}^l_w, \boldsymbol{\lambda}),\ \boldsymbol{R}^l = h_r(\boldsymbol{\theta}^l_r, \boldsymbol{\lambda}),\quad \boldsymbol{P} = h_p(\boldsymbol{\theta}_p, \boldsymbol{\lambda}),\ \boldsymbol{Q'} = h_{q'}(\boldsymbol{\theta}_{q'}, \boldsymbol{\lambda}),\quad \boldsymbol{Q} = h_q(\boldsymbol{\theta}_q, \boldsymbol{\lambda}) \quad (2)$$

In the following sections, we considered different implementations of the hypernetworks $h_*(\boldsymbol{\theta}, \boldsymbol{\lambda})$ of Equation 2.

Addition and Taylor Model While we could use a generic multi-layer perceptron (MLP), we present more memory-efficient architectures. From Equation 1, we scale the row and columns of the $\boldsymbol{W}^l$ tensor or only the rows of the $\boldsymbol{R}^l$ matrix. The intuition is to allow to learn a fixed direction term $\boldsymbol{R}^l_1$ and then be able to scale independently in each dimension based on the parameter λ_k. We call this version the *Addition* version.

$$h_r(\boldsymbol{\theta}^l_r, \boldsymbol{\lambda}) = \boldsymbol{R}^l_0 + (\boldsymbol{V}^l_0 \text{diag}(\boldsymbol{\lambda}) \boldsymbol{V}^{lT}_1) \odot_{\text{row,col}} \boldsymbol{R}^l_1,\quad h_w(\boldsymbol{\theta}^l_w, \boldsymbol{\lambda}) = \boldsymbol{W}^l_0 + (\boldsymbol{U}^l_0 \boldsymbol{\lambda}) \odot_{\text{row}} \boldsymbol{W}^l_1, \quad (3)$$

where $\odot_{\text{row}}$ and $\odot_{\text{row,col}}$ represent the Hadamard product only along the specified dimensions (i.e. along the rows and along both rows and columns). We can modify the previous model to reduce the number of parameters

$$h_r(\boldsymbol{\theta}^l_r, \boldsymbol{\lambda}) = \boldsymbol{R}^l_0 \odot (\boldsymbol{I} + \boldsymbol{V}^l_0 \text{diag}(\boldsymbol{\lambda}) \boldsymbol{V}^{lT}_1),\quad h_w(\boldsymbol{\theta}^l_w, \boldsymbol{\lambda}) = \boldsymbol{W}^l_0 \odot (\boldsymbol{I} + \boldsymbol{U}^l_0 \boldsymbol{\lambda}), \quad (4)$$

where $\odot$ refers to the Hadamard operator along the relevant dimensions. We call this version *Taylor*. The parameters $\boldsymbol{\lambda}$ can be encoded using additional neural networks of minimal size, $\boldsymbol{\lambda}' = g(\boldsymbol{\theta}_g, \boldsymbol{\lambda})$, with $\boldsymbol{\theta}_g$ additional HYPERFNO parameters.

Inverse Problem over PDE parameters Since HYPERFNO is able to adapt to the change in the parameters $\boldsymbol{\lambda}$ of the PDE, HYPERFNO is used for the inverse problem, where, after training HYPERFNO network over multiple PDE's parameters, we can recover the parameters of new samples by solving $\boldsymbol{\lambda}^* = \min_{\boldsymbol{\lambda}} \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{u}) \sim D^{\text{fs}}} ||\boldsymbol{u} - f(\boldsymbol{\psi}_{\boldsymbol{\lambda}}, \boldsymbol{x})||^2,\ \ \boldsymbol{\psi}_{\boldsymbol{\lambda}} = h(\boldsymbol{\theta}, \boldsymbol{\lambda})$, where $\boldsymbol{\lambda}^*$ is the optimal parameter given the trained hypernetwork $h(\boldsymbol{\theta}, \boldsymbol{\lambda})$ and the new dataset D^{fs}. The previous problem can be solved using stochastic gradient descent $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \mu \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{u}) \sim D^{\text{fs}}} ||\boldsymbol{u} - f(\boldsymbol{\psi}_{\boldsymbol{\lambda}}, \boldsymbol{x})||^2$

3 Experiments

In order to evaluate the performance of HYPERFNO, we considered the following problems (Li et al., 2021; Takamoto et al., 2022): 1) one-dimensional Burgers' equation, 2) one-dimensional reaction-diffusion equation, 3) two-dimensional Decaying Flow problem and 4) two-dimensional Compressible Fluid Dynamic (Compressible Navier Stokes - CNS). Contrary to (Li et al., 2021), we prepare datasets allowing various parameter values for instance for the diffusion coefficient.

[2]Hypernetwork term refers to the combination of the two networks or only to the generative network, the meaning is given by the context.

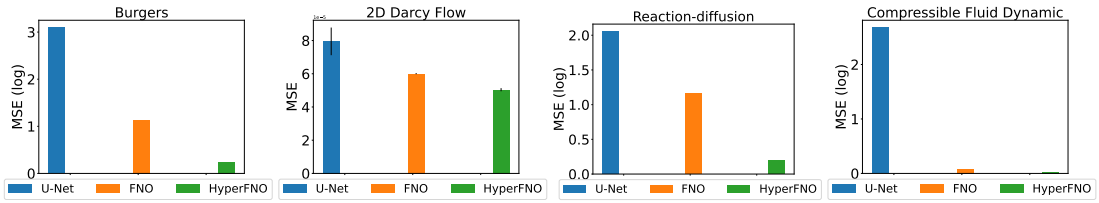
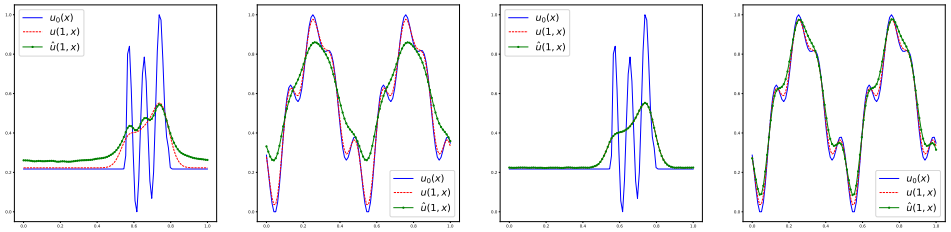


Figure 2: MSE at test time for U-Net, FNO, and HYPERFNO models; tested on (left to right) Burgers, 2D Darcy Flow, Reaction-Diffusion, and 2d+time CNS datasets; we use log scale for visualization.



(a) FNO (b) FNO (c) HYPERFNO (d) HYPERFNO

Figure 3: Visual comparison of FNO and HYPERFNO for different initial conditions and PDE parameters, test on the left (a,c), while training on the right (b,d). The x-axis is the spatial domain, while the y-axis is the solution, either predicted $\hat{u}(t=1,x)$or true $u(t=1,x)$, while we also report the initial condition $u_0(x)$ in a different color.

Zero-Shot learning In Zero-Shot learning, at training time, we have access to solutions of a PDE over different initial conditions and for a set of PDE parameters. At inference time, we use the PDE parameters of the new environment as input to the HYPERFNO to generate the parameters of the main FNO network. We then use this network to predict the solutions for new initial conditions. To evaluate the performance of HYPERFNO, we compared it in various numerical computational problems against the original FNO (Li et al., 2020) and the U-Net (Ronneberger et al., 2015).

Results Figure 2 shows the MSE error at test time for the Burgers, 2-d Darcy Flow, Reaction-Diffusion, and 2d+time compressible Navier-Stokes simulations. We trained over 10 tasks and 100 epochs, where we split the data between training and testing $60\%/40\%$ over $1'000$ samples (500 for CNS). We use a 128 dimensional grid for 1-d equations, while 64×64 for the 2-d problems. We use GeForce RTX 2080 GPU for 1D simulation, while GeForce GTX 3090 for 2D Navier Stocks simulations. In visualizing the results, we use a log-scale plot when the scale is too large. The CNS equation is trained auto-regressively, while the others are trained to predict a specific time step. In all cases, FNO has better performance with respect to U-Net, while HYPERFNO presents greater flexibility to adapt to the new environment. Figure 3 shows qualitatively that FNO is not able to predict the solution of the Burgers equation even if the task was seen during training.

4 Conclusion

We propose HYPERFNO, a method to adapt the FNO architecture over a wide range of parameters of the PDE. We show the improvement gained over different physics systems, such as the Burgers equation, the reaction-diffusion, the Darcy flow, and the compressible fluid dynamic. Meta-learning for Physics Informed Machine Learning is an important direction of research and we proposed a method in this direction that allows us to adapt NOs to new environments. The current method is limited to PDE of the hydro dynamic family. In its mathematical form, the proposed method has direct foreseen no negative societal impact.

References

Christopher J. Arthurs and Andrew P. King. Active training of physics-informed neural networks to aggregate and interpolate parametric solutions to the navier-stokes equations. *Journal of Computational Physics*, 438:110364, Aug 2021. ISSN 00219991. doi: 10.1016/j.jcp.2021.110364.

V. I. Avrutskiy. Neural networks catching up with finite differences in solving partial differential equations in higher dimensions. *Neural Computing and Applications*, 32(17):13425–13440, Sep 2020. ISSN 0941-0643, 1433-3058. doi: 10.1007/s00521-020-04743-8. arXiv: 1712.05067.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in NeurIPS*, 31, 2018.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *ICLR*, 2022.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, Jun 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00314-5.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv:2003.03485*, Mar 2020. URL `http://arxiv.org/abs/2003.03485`.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv:2010.08895*, May 2021. URL `http://arxiv.org/abs/2010.08895`.

Konstantin Mischaikow and Marian Mrozek. Chaos in the lorenz equations: a computer-assisted proof. *Bulletin of the American Mathematical Society*, 32(1):66–72, 1995.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv:2202.11214*, Feb 2022. URL `http://arxiv.org/abs/2202.11214`.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks. *Journal of Computational Physics*, 378:686–707, Feb 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045.

Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *arXiv:1801.06637*, Jan 2018. URL `http://arxiv.org/abs/1801.06637`.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv:1808.04327*, Aug 2018. URL `http://arxiv.org/abs/1808.04327`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Int. Conf. on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Makoto Takamoto, Timothy Pradita, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: A diverse and comprehensive benchmark for scientific machine learning, 2022. URL `https://darus.uni-stuttgart.de/privateurl.xhtml?token=1be27526-348a-40ed-9fd0-c62f588efc01`.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1457–1466, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code will be provided; experimental detail could not be included in the main paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We presented the main setup of the experiments, but not all experimental detail is included.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We reported only partial error information; we run experiments multiple time.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We reported some information on the execution environment.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Yes, we cited the authors of the work we based our work on.
   (b) Did you mention the license of the assets? [N/A] we will provide the license of the code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]