Factorized Fourier Neural Operators

Alasdair Tran[1], **Alexander Mathews**[1], **Lexing Xie**[1], **Cheng Soon Ong**[1,2]
[1] Australian National University
[2] Data61, CSIRO
`{alasdair.tran,alex.mathews,lexing.xie,chengsoon.ong}@anu.edu.au`

Abstract

The Fourier Neural Operator (FNO) is a learning-based method for efficiently simulating partial differential equations. We propose the Factorized Fourier Neural Operator (F-FNO) that allows much better generalization with deeper networks. With a careful combination of the Fourier factorization, a shared kernel integral operator across all layers, the Markov property, and residual connections, F-FNOs achieve a six-fold reduction in error on the most turbulent setting of the Navier-Stokes benchmark dataset. We show that our model maintains an error rate of 2% while still running an order of magnitude faster than a numerical solver, even when the problem setting is extended to include additional contexts such as viscosity and time-varying forces. This enables the same pretrained neural network to model vastly different conditions.

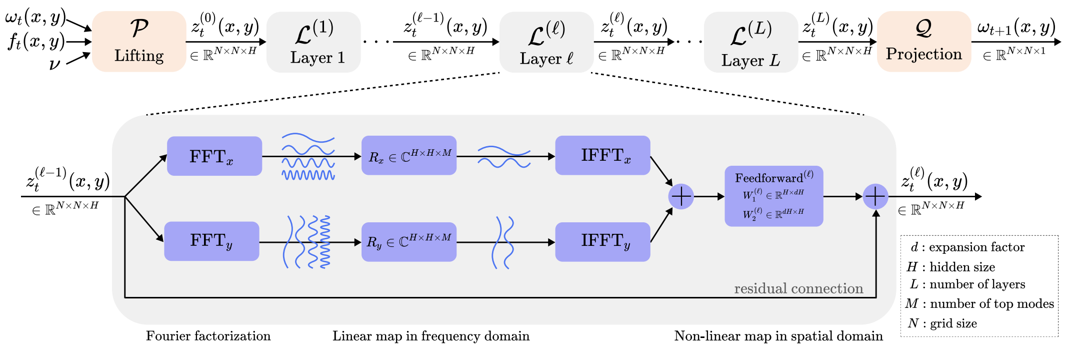
1 Introduction

From modeling population dynamics to understanding the formation of stars, partial differential equations (PDEs) permeate the world of science and engineering. For most real-world problems, the lack of closed-form solutions requires computationally expensive numerical solvers, sometimes consuming millions of core hours and terabytes of storage [2].

Deep learning architectures have come full circle, from initially replacing frequency-domain representations with learned features [1] to a resurgence of Fourier operators due to the $O(n \log n)$ time complexity of the fast Fourier transform (FFT) and its ability to represent long-range dependencies. Two notable examples are FNet [5] for representing semantic relationships in text classification and the Fourier Neural Operator (FNO) [7, 8] for flow simulation. FNOs, in particular, have been shown to approximate any continuous operator [4]. However, the original FNO architecture has not taken advantage of the separable nature of Fourier transforms in space, more parsimonious neural representations, and the neural network's versatility to solve more general settings. We address these limitations through the following three contributions:

1. We propose the Factorized Fourier Neural Operator (F-FNO) (Fig. 1), which uses a separable Fourier representation and a shared kernel integral operator across all layers to reduce the parameter count from 100M to 1M and enable the network to grow to 24 layers on a single GPU (Fig. 3a).
2. For the Navier-Stokes equations on the torus, our approach outperforms the state of the art, reducing the normalized mean squared error from 15.56% to 2.29% and maintaining an error reduction of more than 80% even when predicting 10 steps ahead. (Fig. 2).
3. F-FNOs can take additional contexts as input (Fig. 4), allowing us to solve more general Navier-Stokes problems with different parameters *without the need to retrain*. On newly created data with a range of viscosities and time-varying forces, F-FNOs maintain an error of 2% (Table 1). Code and datasets are released on GitHub.[1]

[1] `https://github.com/alasdairtran/fourierflow`



$\omega_t(x,y)$
$f_t(x,y)$
ν
$\mathcal{P}$ Lifting
$z_t^{(0)}(x,y)$ $\in \mathbb{R}^{N\times N\times H}$
$\mathcal{L}^{(1)}$ Layer 1
$z_t^{(\ell-1)}(x,y)$ $\in \mathbb{R}^{N\times N\times H}$
$\mathcal{L}^{(\ell)}$ Layer ℓ
$z_t^{(\ell)}(x,y)$ $\in \mathbb{R}^{N\times N\times H}$
$\mathcal{L}^{(L)}$ Layer L
$z_t^{(L)}(x,y)$ $\in \mathbb{R}^{N\times N\times H}$
$\mathcal{Q}$ Projection
$\omega_{t+1}(x,y)$ $\in \mathbb{R}^{N\times N\times 1}$
$z_t^{(\ell-1)}(x,y)$ $\in \mathbb{R}^{N\times N\times H}$
FFT$_x$
FFT$_y$
$R_x \in \mathbb{C}^{H\times H\times M}$
$R_y \in \mathbb{C}^{H\times H\times M}$
IFFT$_x$
IFFT$_y$
Feedforward$^{(\ell)}$ $W_1^{(\ell)} \in \mathbb{R}^{H\times dH}$ $W_2^{(\ell)} \in \mathbb{R}^{dH\times H}$
$z_t^{(\ell)}(x,y)$ $\in \mathbb{R}^{N\times N\times H}$
residual connection
Fourier factorization
Linear map in frequency domain
Non-linear map in spatial domain
d : expansion factor
H : hidden size
L : number of layers
M : number of top modes
N : grid size

Figure 1: The architecture of the Factorized Fourier Neural Operator (F-FNO). The iterative process (Eq. (1)) is shown at the top, with a zoomed-in operator layer (Eq. (4)) at the bottom.

2 The Factorized Fourier Neural Operator

Preliminaries We are interested in learning neural operators between infinite-dimensional function spaces. We consider the 2D Navier-Stokes equations as the guiding example, in which the function space $\Omega(D;\mathbb{R})$ consists of vorticity fields $\omega_t \in \Omega$ over a bounded spatial domain, i.e. $\omega_t : D \to \mathbb{R}$ with $D \subset \mathbb{R}^2$. The operator $\mathcal{G} : \Omega \to \Omega$ would then learn to map this field ω_t at some time step t to the next time step $t+1$. Motivated by the kernel formulation of the solution to linear PDEs using Green's functions, Li et al. [6] propose an iterative approach to map ω_t to ω_{t+1},

$$\omega_{t+1} = \mathcal{G}(\omega_t) = (\mathcal{Q} \circ \mathcal{L}^{(L)} \circ \mathcal{L}^{(L-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{P})(\omega_t), \tag{1}$$

where $\circ$ indicates function composition, L is the number of layers/iterations, $\mathcal{P}$ is the lifting operator that maps the input function ω_t into the first latent representation $z_t^{(0)}$, $\mathcal{L}^{(\ell)}$ is the ℓ'th non-linear operator layer, and $\mathcal{Q}$ is the projection operator that maps the last latent representation $z_t^{(L)}$ to the output function ω_{t+1}. Fig. 1 (top) contains a schematic diagram of this iterative process. Originally, Li et al. [7] formulate each operator layer as

$$\mathcal{L}^{(\ell)}\left(z_t^{(\ell)}\right) = \sigma\left(W^{(\ell)} z_t^{(\ell)} + b^{(\ell)} + \mathcal{K}^{(\ell)}(z_t^{(\ell)})\right), \tag{2}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a point-wise non-linear activation function, $W^{(\ell)} z_t^{(\ell)} + b^{(\ell)}$ is an affine point-wise map in the spatial domain, and $\mathcal{K}^{(\ell)}$ is a kernel integral operator using the Fourier transform,

$$\mathcal{K}^{(\ell)}\left(z_t^{(\ell)}\right) = \text{IFFT}\left(R^{(\ell)} \cdot \text{FFT}(z_t)\right) \tag{3}$$

Note that the frequency-domain weight matrices $\{R^{(\ell)} \mid \ell \in \{1, 2, \ldots, L\}\}$ take up most of the model size, requiring $O(LH^2M^2)$ parameters, where H is the hidden size and M is the number of top modes being kept.

F-FNO We consider a more general setting in which the operator $\mathcal{G}$ maps the vorticity field ω_t along with some contexts such as the viscosity ν and the forcing function f_t, i.e., $\omega_{t+1} = \mathcal{G}(\omega_t \mid \nu, f_t)$. Furthermore, we design the following operator layer:

$$\mathcal{L}^{(\ell)}\left(z_t^{(\ell)}\right) = z_t^{(\ell)} + \sigma\left[W_2^{(\ell)} \sigma(W_1^{(\ell)} \mathcal{K}(z_t^{(\ell)}) + b_1^{(\ell)}) + b_2^{(\ell)}\right] \tag{4}$$

Note that we apply the residual connection *after* the non-linearity to allow an easier flow of gradients. Our kernel integral operator is shared across all layers. We also factorize the Fourier transforms over the two spatial dimensions, giving us

$$\mathcal{K}\left(z_t^{(\ell)}\right) = \text{IFFT}\left(R_x \cdot \text{FFT}_x(z_t^{(\ell)})\right) + \left(R_y \cdot \text{FFT}_y(z_t^{(\ell)})\right) \tag{5}$$

Our approach uses two weight matrices, R_x and R_y, that together have only $O(H^2M)$ parameters, irrespective of the number of layers. Fig. 1 (bottom) gives the overview of an F-FNO operator layer. We will show in the next section that the factorization and weight sharing in $\mathcal{K}$, along with the residual connection in $\mathcal{L}$ and other deep learning techniques, allow F-FNOs to vastly outperform the state of the art.

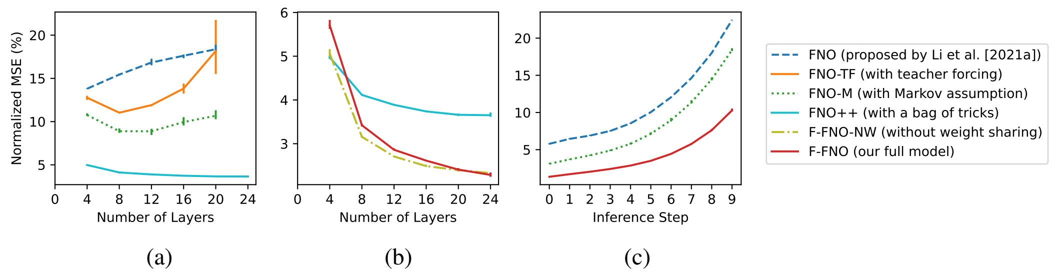


Figure 2: Performance (lower is better) of six model variants on `TorusConst`, with error bars showing the min and max values over three trials. In (a), we improve the original FNO [7] by using teacher forcing, the Markov property, among other techniques, resulting in FNO++. In (b), we show the effect of Fourier factorization and weight sharing. In (c), we show (for a four-layer network) the error as we infer more time steps into the future.

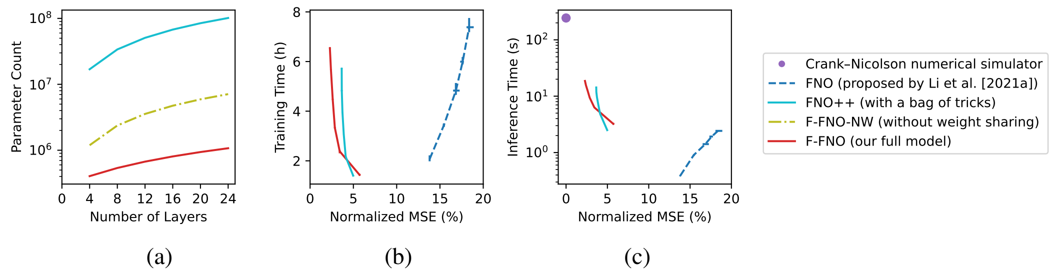


Figure 3: The resource usage of four model variants, in terms of the parameter count (a), training time (b), and inference time (c). Error bars show the min and max values over three trials.

3 Experiments on Navier-Stokes Equations

Datasets We test the F-FNO on three fluid dynamics datasets. `TorusConst` [7] is used to benchmark our model against the state of the art. `TorusV` and `TorusVF` are new datasets we generate to test the generalization of F-FNOs across Navier-Stokes tasks with different viscosities and forcing functions. The training split of each dataset consists of 1000 simulations of the 2D Navier-Stokes equations on a unit torus, and each simulation contains 20 steps of a 64×64 grid of the vorticity field. The viscosity is fixed at 10^{-5} ($\text{Re} = 2000$) in `TorusConst`, while it varies between 10^{-4} and 10^{-5} in `TorusV` and `TorusVF`. The forcing function on the torus is defined as

$$f(t,x,y) = 0.1 \sum_{p=1}^{2} \sum_{i=0}^{1} \sum_{j=0}^{1} \Big[\alpha_{pij} \sin\big(2\pi p(ix+jy)+\delta t\big) + \beta_{pij} \cos\big(2\pi p(ix+jy)+\delta t\big) \Big] \tag{6}$$

where the amplitudes α_{pij} and β_{pij} are sampled from the uniform distribution in `TorusV` and `TorusVF`. Furthermore, δ is set to 0 in `TorusV`, making the forcing function constant across time; while it is set to 0.2 in `TorusVF`, giving us a time-varying force. Finally in `TorusConst`, the forcing function is fixed at $f(x,y) = 0.1[\sin(2\pi(x+y)) + \cos(2\pi(x+y))]$ across all samples and time steps.

Training details For the experiments involving the original FNO, FNO-TF (with teaching forcing), and FNO-M (with Markov assumption), we use the same training procedure as Li et al. [7]. For all other experiments, we train for 100,000 steps, warming up the learning rate to 2.5×10^{-3} for the first 500 steps and then decaying it using the cosine function [9]. We use ReLU as our non-linear activation function, clip the gradient value at 0.1, and use the Adam optimizer [3] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. The weight decay factor is set to 10^{-4} and is decoupled from the learning rate [10]. Models are implemented in PyTorch [11] and trained on a single Titan V GPU. Finally, the loss function is the normalized mean squared error, defined as $\text{N-MSE} = \frac{1}{B}\sum_{i=1}^{B} \frac{\|\hat{\omega}_i - \omega\|_2}{\|\omega\|_2}$, where $\|\cdot\|_2$ is the 2-norm, B is the batch size, and $\hat{\omega}$ is the prediction of the ground truth ω.

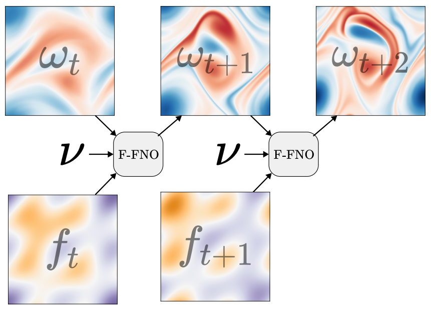


Figure 4: F-FNO solves the Navier-Stokes equations with a range of parameters without needing to retrain, by taking viscosity ν and forcing function f_t as input.

Table 1: Performance of F-FNO on `TorusV` and `TorusVF`, with ablation studies to test the contribution of each context. Each result is accompanied by the standard deviation from three trials.

	N-MSE (%)
`TorusV`	
Without context	43.8 ± 0.9
With force	4.48 ± 0.09
With force and viscosity	2.06 ± 0.02
`TorusVF`	
Without context	33.0 ± 0.4
With force	4.2 ± 0.1
With force and viscosity	2.01 ± 0.05

Results against state of the art The performance on `TorusConst` is plotted in Figs. 2 and 3, with more details in Appendix Table 3. First, note that in FNO, Li et al. [7] use the previous 10 steps as input and predict the next 10 steps incrementally (by using each predicted value as the input to the next step). We find that the teacher forcing strategy (FNO-TF, orange line), in which we always use the ground truth as input, helps in shallow networks. Enforcing the Markov condition (FNO-M, dotted green line), where only one step of history is used, further improves the performance. Including two or more steps of history makes no difference to the results.

However, all of these three models (FNO, FNO-TF, and FNO-M) do not scale with network depth, even failing to converge at 24 layers. The real improvement in deeper networks does not come until we move the residual connection outside the non-linear activation to ease the gradient flow, and couple it with other deep learning techniques such as normalizing the inputs, adding Gaussian noise, and using a cosine learning rate scheduler with warmup (FNO++, cyan line). Furthermore, if we use Fourier factorization (F-FNO-NW, yellow dashed line), the error drops by an additional 64%, from 3.65% to 2.33%, at 24 layers (Fig. 2b), while the parameter count is reduced by an order of magnitude (Fig. 3a). Sharing the weights in the Fourier domain (F-FNO, red line) helps the most with the deepest network, pushing the error down to 2.29% and reducing the parameter count by another order of magnitude to 1M. Finally, even though our models do take longer to do inference, the deepest network is still an order of magnitude faster than the Crank-Nicholson numerical simulator (Fig. 3c).

Results on more general settings The same pretrained F-FNO can handle Navier-Stokes equations with different viscosities (in `TorusV`) and time-varying forcing functions (in `TorusVF`). Fig. 4 shows an example where our model takes these two contexts as additional input channels. On both datasets, the F-FNO is able to maintain an error of 2% (Table 1). We also conduct ablation studies to show the importance of the contexts. In particular, if we remove the viscosity information, the error doubles. If we further remove the forcing function from the input, the error increases by an order of magnitude, showing the substantial influence the force has on the vorticity field.

4 Conclusion

The Fourier transform is a powerful tool to learn neural operators that can handle long-range dependencies efficiently. By factorizing the transform, sharing the kernel integral operator across layers, enforcing the Markov condition, and carefully constructing the training process, our proposed F-FNO outperforms the state of the art by 85% and handles a range of Navier-Stokes parameters without retraining. For future work, we plan to compare our method against baseline spectral solvers at different resolutions to illustrate the full Pareto frontier of accuracy vs. inference speed. We are also interested in extending the Fourier layers to irregular structures such as graphs and meshes. Finally we would like to examine equilibrium properties of generalized Fourier layers in the limit of an infinite number of layers.

Broader Impact

This work focuses on fundamental technology that is typically used as a component in simulation and control systems. Such larger systems could bring significant social benefits (e.g., weather forecast, airplane design) or harm (e.g., missiles). The advantage of making this work available outweighs the potential harm it could cause. Since this work does not pose an immediate risk, making the work publicly available could help to ensure that any negative effects are anticipated and mitigated in advance since keeping a technology secret is not a valid or effective preventative measure of harm.

Acknowledgments

This research is supported in part by the Australian Research Council Project DP180101985 and AOARD project 20IOA064. We thank NVIDIA for providing us with Titan V GPUs for experimentation.

References

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/amodei16.html.

[2] S.M. Hosseini, R. Vinuesa, P. Schlatter, A. Hanifi, and D.S. Henningson. Direct numerical simulation of the flow around a wing section at moderate reynolds number. *International Journal of Heat and Fluid Flow*, 61:117–128, 2016. ISSN 0142-727X. doi: https://doi.org/10.1016/j.ijheatfluidflow.2016.02.001. URL https://www.sciencedirect.com/science/article/pii/S0142727X16300169. SI TSFP9 special issue.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[4] Nikola B. Kovachki, S. Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *ArXiv*, abs/2107.07562, 2021.

[5] J. Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontañón. Fnet: Mixing tokens with fourier transforms. *ArXiv*, abs/2105.03824, 2021.

[6] Zong-Yi Li, Nikola B. Kovachki, K. Azizzadenesheli, Burigede Liu, K. Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *ArXiv*, abs/2003.03485, 2020.

[7] Zongyi Li, Nikola B. Kovachki, K. Azizzadenesheli, Burigede Liu, K. Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

[8] Zongyi Li, Nikola B. Kovachki, K. Azizzadenesheli, Burigede Liu, K. Bhattacharya, Andrew Stuart, and Anima Anandkumar. Markov neural operators for learning chaotic systems. *ArXiv*, abs/2106.06898, 2021.

[9] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[11] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

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] See Section 4.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the Broader Impact section after Section 4.
   (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] See Section 3.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 3.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Fig. 2 and Fig. 3.
   (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] See Fig. 3b and Fig. 3c.
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] We cited Li et al. [7] for the `TorusConst` dataset.
   (b) Did you mention the license of the assets? [Yes] See the Appendix B.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the Appendix B.
   (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]

A The Navier-Stokes Equations

Our experiments are conducted on simulations of the incompressible 2D Navier-Stokes equations on the unit torus using the vorticity form:

$$\frac{\partial}{\partial t}\omega_t(x,y) + v_t(x,y)\cdot\nabla\omega_t(x,y) = \nu\Delta\omega_t(x,y) + f_t(x,y) \tag{7}$$

$$\nabla\cdot v_t(x,y) = 0 \tag{8}$$

$$\omega_t(x,y) = \omega_0(x,y) \tag{9}$$

where $x, y \in (0,1)$ are the positions on the unit torus, $t \in (0,T]$ is the time step, ν is the viscosity, $v_t(x,y)$ is the velocity field, $\omega \coloneqq \nabla \times v$ is the vorticity field, and $f_t(x,y)$ is the forcing function. Given the forcing function

$$f(x,y) = 0.1[\sin(2\pi(x+y)) + \cos(2\pi(x+y))], \tag{10}$$

the Reynolds number is approximately

$$Re \approx \frac{\sqrt{0.1}}{\nu(2\pi)^{3/2}} \tag{11}$$

B Code and Datasets

The code, pretrained models, and datasets are licensed under the MIT License. They can be found at `https://github.com/alasdairtran/fourierflow`.

C Further Results

Detailed results, including the normalized mean squared error, along with complexity measures such as the parameter count and inference time, are shown in Table 3.

Dataset	Size	Viscosity	Forcing function varying	
			across samples	across time
`TorusConst` [7]	1,200	$\nu = 10^{-5}$	✗	✗
`TorusV`	1,400	$\nu \in [10^{-5}, 10^{-4})$	✓	✗
`TorusVF`	1,400	$\nu \in [10^{-5}, 10^{-4})$	✓	✓

Table 2: An overview of the three datasets used in our experiments, including key differences between them. Observe that our datasets, `TorusV` and `TorusVF`, contain simulation data with a more variety of viscosities and forces than `TorusConst`. Note also that Li et al. [7] do not include a validation set, splitting the data into 1,000 training examples and 200 test examples. In our datasets, we generate 200 additional simulations to be used as validation.

	No. of Layers	No. of Parameters	N-MSE (%)	Training Time (h)	Inference Time (s)
ResNet [7]	-	266,641	27.53	-	-
TF-Net [7]	-	7,451,724	22.68	-	-
U-Net [7]	-	7,451,724	19.82	-	-
FNO [7]	4	414,517	15.56	-	-
FNO (reproduced)	4	926,357	13.81 ± 0.04	2.1	0.4
	8	1,849,637	15.44 ± 0.09	3.4	0.9
	12	2,772,917	16.84 ± 0.32	4.8	1.4
	16	3,696,197	17.62 ± 0.20	6.0	1.9
	20	4,619,477	18.37 ± 0.38	7.4	2.4
FNO-TF (with teacher forcing)	4	926,357	12.76 ± 0.20	2.0	0.4
	8	1,849,637	11.03 ± 0.07	3.3	0.9
	12	2,772,917	11.91 ± 0.10	4.6	1.4
	16	3,696,197	13.82 ± 0.47	6.0	1.9
	20	4,619,477	18.17 ± 2.64	7.3	2.4
FNO-M (with Markov assumption)	4	926,177	10.76 ± 0.14	1.5	0.4
	8	1,849,457	8.89 ± 0.22	2.4	0.9
	12	2,772,737	8.88 ± 0.31	3.3	1.4
	16	3,696,017	9.95 ± 0.41	4.2	1.9
	20	4,619,297	10.68 ± 0.45	5.0	2.4
FNO++ (with bags of tricks)	4	16,919,746	4.98 ± 0.03	1.4	2.5
	8	33,830,594	4.11 ± 0.02	2.2	4.9
	12	50,741,442	3.89 ± 0.02	3.1	7.2
	16	67,652,290	3.74 ± 0.01	4.0	9.6
	20	84,563,138	3.66 ± 0.02	4.8	11.9
	24	101,473,986	3.65 ± 0.04	5.7	14.3
F-FNO-NW (without weight sharing)	4	1,191,106	5.06 ± 0.08	1.5	3.2
	8	2,373,314	3.16 ± 0.02	2.4	6.3
	12	3,555,522	2.71 ± 0.01	3.6	9.4
	16	4,737,730	2.49 ± 0.02	4.7	12.5
	20	5,919,938	2.40 ± 0.01	5.8	15.5
	24	7,102,146	2.33 ± 0.01	6.8	18.6
F-FNO (our full model)	4	404,674	5.70 ± 0.09	1.4	3.2
	8	538,306	3.42 ± 0.02	2.4	6.3
	12	671,938	2.86 ± 0.02	3.3	9.4
	16	805,570	2.61 ± 0.01	4.5	12.5
	20	939,202	2.41 ± 0.03	5.7	15.6
	24	1,072,834	2.29 ± 0.04	6.5	18.6

Table 3: Performance on `TorusConst`. When possible, the N-MSE is accompanied by the standard deviation from three trials. A hyphen indicates that the data is not available.