FACTORIZED FOURIER NEURAL OPERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose the Factorized Fourier Neural Operator (F-FNO), a learning-based approach for simulating partial differential equations (PDEs). Starting from a recently proposed Fourier representation of flow fields, the F-FNO bridges the performance gap between pure machine learning approaches to that of the best numerical or hybrid solvers. This is achieved with several insights that collectively have a significant effect – the separable spectral representations; improved residual connections; and carefully designed training strategies. On several challenging benchmark PDEs on regular grids, structured meshes, and point clouds, the F-FNO can scale to deeper networks and outperform both the FNO and the geo-FNO, reducing the error by 83% on the Kolmogorov flow, 31% on the elasticity problem, 57% on the airfoil flow problem, and 60% on the plastic forging problem. Compared to the state-of-the-art pseudo-spectral method, the F-FNO can take a step size that is an order of magnitude larger in time and achieve an order of magnitude speedup to produce the same solution quality.

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 (Hosseini et al., 2016). Of particular interest are turbulent flows, governed by the Navier-Stokes equations. Turbulence is very costly to simulate due to the need to resolve the smallest flow features – a 10-fold increase in the Reynolds number requires a 1000-fold increase in compute power to run a Direct Numerical Simulation (DNS) (Kochkov et al., 2021). This has led researchers to data-driven approaches in search of more efficient solutions.

In order to accelerate a DNS, we need to consider both the spatial and the temporal aspects. Hybrid methods such as learned interpolation and learned correction (Kochkov et al., 2021) achieve the speedup via superresolution – specifically by running a traditional solver on a lower resolution grid, coupled with a neural network to fill in or correct for the missing details. In contrast, the efficiency of pure machine learning methods like neural operators (Kovachki et al., 2021b), which this work is a part of, comes both from *superresolution* (i.e., being able to model the finer details that a numerical solver would miss on the same resolution) and from *leaping in time* (i.e., taking a step size much bigger than is allowed in numerical methods).

In the family of neural operators, the Fourier Neural Operator (FNO) (Li et al., 2021a) was recently proposed to solve PDEs purely from data. However, Lu et al. (2022) found stability issues with the FNO when testing on complex geometries and noisy data. In our experiments, we find that the original FNO and its derivative geo-FNO (Li et al., 2022) are only stable when there are a handful of network layers. We hypothesize that there might not be enough regularization in the original architecture. In our exploration of various regularization techniques, we find that it is possible to reduce the parameter count by up to an order of magnitude while still outperforming the state of the art by a significant margin. In particular, through the factorization of the Fourier transform, our model can learn higher-dimensional problems with ease, even when deeply stacked. On the 3D plastic forging problem, for instance, we are able to reduce the prediction error by 60% ($0.45 \rightarrow 0.18$) while also lowering the parameter count by 80% (57M $\rightarrow$ 11M).

Overall, we make the following three key contributions:

1. We propose the Factorized Fourier Neural Operator (F-FNO), which uses a separable Fourier representation to improve model stability and performance as we scale to deeper networks (Section 3).
2. On regular grid geometries, the F-FNO outperforms the original FNO on the Navier-Stokes simulations, reducing the error by 83% while still achieving an order of magnitude speedup over the state-of-the-art pseudo-spectral method (Section 5).
3. On irregular geometries including point clouds and structured meshes, the F-FNO outperforms the geo-FNO on both structural mechanics and fluid dynamics PDEs, reducing the error by up to 60% (Section 6).

2 Related Work

Existing works on solving PDEs with machine learning can be divided into two clusters: hybrid approaches and pure machine learning approaches. Hybrid approaches replace parts of traditional numerical solvers with learned alternatives but keep the components that impose physical constraints such as conservation laws; while pure machine learning approaches learn the time evolution of PDEs from data only.

Hybrid methods typically aim to speed up traditional numerical solvers by using lower resolution grids (Bar-Sinai et al., 2019; Um et al., 2020; Kochkov et al., 2021), or by replacing computationally expensive parts of the solver with learned alternatives Tompson et al. (2017); Obiols-Sales et al. (2020). Bar-Sinai et al. (2019) develop a data driven method for discretizing PDE solutions, allowing coarser grids to be used without sacrificing detail. Kochkov et al. (2021) design a technique specifically for the Navier-Stokes equations that uses neural network-based interpolation to calculate velocities between grid points rather than using the more traditional polynomial interpolation. Their method leads to more accurate simulations while at the same time achieving an 86-fold speed improvement over DNS. Similarly, Tompson et al. (2017) employ a numerical solver and a decomposition specific to the Navier-Stokes equations, but introduce a convolutional neural network to infer the pressure map at each time step. While these hybrid methods are effective when designed for specific equations, they are not easily adaptable to other PDE tasks.

An alternative approach which is less specialized than most hybrid methods, but also less general than pure machine learning methods, is learned correction (Um et al., 2020; Kochkov et al., 2021) which involves learning a residual term to the output of a numerical step. That is, the time derivative is now $\mathbf{u}_t = \mathbf{u}_t^* + LC(\mathbf{u}_t^*)$, where $\mathbf{u}_t^*$ is the velocity field provided by a standard numerical solver on a coarse grid, and $LC(\mathbf{u}_t^*)$ is a neural network that plays the role of super-resolution of missing details.

Pure machine learning approaches eschew the numerical solver altogether and learn the field directly, i.e., $\mathbf{u}_t = \mathcal{G}(\mathbf{u}_{t-1})$, where $\mathcal{G}$ is dubbed a neural operator. The operator can include graph neural networks Li et al. (2020a;b), low-rank decomposition Kovachki et al. (2021b), or Fourier transforms (Li et al., 2021a;b). Pure machine learning models can also incorporate physical constraints, for example, by carefully designing loss functions based on conservation laws (Wandel et al., 2020). They can even be based on existing simulation methods such as the operator designed by Wang et al. (2020) that uses learned filters in both Reynolds-averaged Navier-Stokes and Large Eddy Simulation before combining the predictions using U-Net. However, machine learning methods need not incorporate such constraints – for example, Kim et al. (2019) use a generative CNN model to represent velocity fields in a low-dimensional latent space and a feedforward neural network to advance the latent space to the next time point. Similarly, Bhattacharya et al. (2020) use PCA to map from an infinite dimensional input space into a latent space, on which a neural network operates before being transformed to the output space. Our work is most closely related to the Fourier transform-based approaches (Li et al., 2021a; 2022) which can efficiently model PDEs with zero-shot super-resolution but is not specific to the Navier-Stokes equations.

Fourier representations are popular in deep-learning due to the efficiency of convolution operators in the frequency space, the $O(n \log n)$ time complexity of the fast Fourier transform (FFT), and the ability to capture long-range dependencies. Two notable examples of deep learning models that

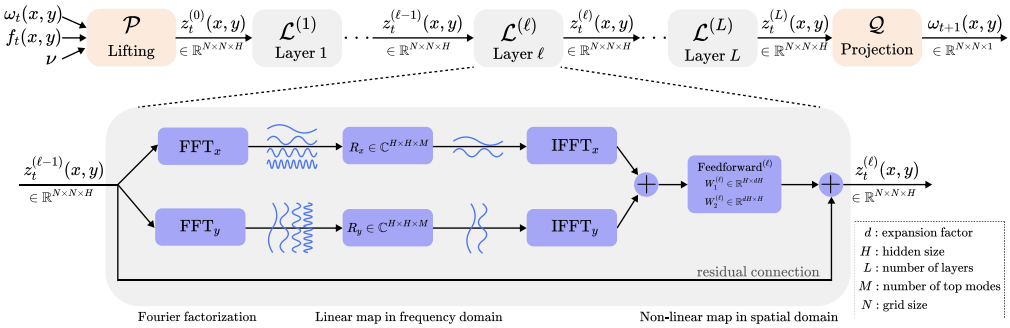


Figure 1: The architecture of the Factorized Fourier Neural Operator (F-FNO) in the context of the 2D Navier-Stokes equations. The iterative process (Eq. (3)) is shown at the top, in which the inputs (flow field ω along with contextual information such as the viscosity ν and forcing function f) are fed through a series of operator layers $\mathcal{L}$ in order to produce the field ω at time $t+1$. A zoomed-in operator layer (Eq. (7)) is shown at the bottom which shows how we process each spatial dimension independently in the Fourier space, before merging them together again in the physical space.

employ Fourier representations are FNet (Lee-Thorp et al., 2021) for encoding semantic relationships in text classification and FNO (Li et al., 2021a) for flow simulation. In learning mappings between function spaces, the FNO outperforms graph-based neural operators and other finite-dimensional operators such as U-Net. In modeling chaotic systems, the FNO has been shown to capture invariant properties of chaotic systems (Li et al., 2021b). More generally, Kovachki et al. (2021a) prove that the FNO can approximate any continuous operator.

3 The Factorized Fourier Neural Operator

Preliminaries As a guiding example, we consider the vorticity formulation of the 2D Navier-Stokes equations,

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega + f \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

where $\mathbf{u}$ is the velocity field, ω is the vorticity, and f is the external forcing function.

The task of a PDE solver is to evolve the field from one time step to the next: $\omega_t \to \omega_{t+1}$. These fields are defined over a bounded spatial domain, $\omega_t : D \to \mathbb{R}$ with $D \subset \mathbb{R}^2$, and they live in a function space, $\omega_t \in \Omega(D; \mathbb{R})$. Under this formulation, we can frame our problem as learning a neural operator between infinite-dimensional function spaces. The operator $\mathcal{G} : \Omega \to \Omega$ would then learn to map the 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. (2020b) propose an iterative approach to map ω_t to ω_{t+1},

$$\omega_{t+1} = \mathcal{G}(\omega_t) = (\mathcal{Q} \circ \mathcal{L}^{(L)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{P})(\omega_t), \tag{3}$$

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. (2021a) formulate each operator layer as

$$\mathcal{L}^{(\ell)}\left(z_t^{(\ell)}\right) = \sigma\left(W^{(\ell)} z_t^{(\ell)} + b^{(\ell)} + \mathcal{K}^{(\ell)}(z_t^{(\ell)})\right), \tag{4}$$

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 physical space, and $\mathcal{K}^{(\ell)}$ is a kernel integral operator using the Fourier transform,

$$\mathcal{K}^{(\ell)}\left(z_t^{(\ell)}\right) = \text{IFFT}\left(R^{(\ell)} \cdot \text{FFT}(z_t)\right) \tag{5}$$

The wavenumber-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 Fourier modes being kept. Furthermore, the constant value for M and the affine point-wise map allow the FNO to be resolution-independent.

Formulation of 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, that is,

$$\omega_{t+1} = \mathcal{G}(\omega_t \mid \nu, f_t) \tag{6}$$

Furthermore, we design the following operator layer:

$$\mathcal{L}^{(\ell)}\Big(z_t^{(\ell)}\Big) = z_t^{(\ell)} + \sigma\Big[W_2^{(\ell)}\sigma\big(W_1^{(\ell)}\mathcal{K}\big(z_t^{(\ell)}\big) + b_1^{(\ell)}\big) + b_2^{(\ell)}\Big] \tag{7}$$

Note that unlike the original FNO, we apply the residual connection ($z_t^{(\ell)}$ term) *after* the non-linearity to preserve more of the layer input – we find that this works better empirically. We also factorize the Fourier transforms over the two spatial dimensions, giving us

$$\mathcal{K}\Big(z_t^{(\ell)}\Big) = \text{IFFT}\Big(R_x^{(\ell)} \cdot \text{FFT}_x(z_t^{(\ell)})\Big) + \text{IFFT}\Big(R_y^{(\ell)} \cdot \text{FFT}_y(z_t^{(\ell)})\Big) \tag{8}$$

Our factorization removes an exponent from the model complexity – the F-FNO only requires $O(LH^2M)$ parameters, which is particularly useful when solving higher-dimensional problems (e.g., the 3D plastic forging problem in Section 4). It is also possible to share the weight matrices R_x and R_y between the layers, which further reduces the parameters to $O(H^2M)$. Fig. 1 (bottom) gives the overview of an F-FNO operator layer.

Learning neural operators In order to learn the F-FNO effectively, we note three important deep learning considerations that were overlooked in Li et al. (2021a)'s original implementation, the first of which is *enforcing the first-order Markov property*. We find Li et al. (2021a)'s use of the last 10 time steps as inputs to the neural operator to be unnecessary. Instead, it is sufficient to feed information only from the current step, just like a numerical solver.

Furthermore, we would like to highlight the F-FNO's *flexibility in the input representation*. The contextual information that the F-FNO takes can be anything that is relevant to the evolution of the field. For example, if we know that an external force is acting on the field, we can simply add that force field as an additional input channel. In our experiments, we will show that each dataset makes use of a different set of contexts. This flexibility also allows our model to be easily generalized to different PDEs.

Finally, unlike prior works (Li et al., 2021a; Kochkov et al., 2021), we do not unroll the model during training but instead use the *teacher forcing* technique which is often seen in time series and language modeling. In teacher forcing, we use the ground truth as the input to the neural operator.

Connection with pseudo-spectral methods Certain operations such as differentiation can be performed more efficiently by first taking the Fourier transform. Pseudo-spectral methods take advantage of this and compute the gradient $\nabla\omega$ and the Laplacian $\nabla^2\omega$ in the wavenumber space, while the divergent $\mathbf{u} \cdot \nabla$ is done in the physical space. This back and forth between the two spaces can also be observed in the F-FNO (see Eqs. (7) and (8)), although we let the network decide which features to learn in each space.

Learning on irregular geometries So far, we have formulated the F-FNO in the context of the 2D Navier-Stokes equations, where we learn an operator that maps the vorticity field from one time step to the next. But neural operators can solve more general problems. Consider the equation for a solid body in structural mechanics, $\rho^s \dfrac{\partial^2 \mathbf{u}}{\partial t^2} + \nabla \cdot \sigma = 0$, where p^s is the mass density, $\mathbf{u}$ is the displacement vector and σ is the stress tensor. The Elasticity dataset, governed by this equation and described in Section 4, consists of point clouds, not regular grids. Our neural operator would then need to learn to map a point cloud to its corresponding stress value. Since the standard Fourier transform expects a grid-based input, we would like to find a diffeomorphic deformation between the input domain (e.g., the point cloud) and the unit torus. This would allow us to continue using Eq. (3),

Table 1: An overview of the datasets and the corresponding task.

Dataset	Geometry	Dim.	Problem	Input	Output
TorusLi	regular grid	2D	Kolmogorov flow	ω_t	ω_{t+1}
TorusKochkov	regular grid	2D	Kolmogorov flow	ω_t	ω_{t+1}
TorusVis	regular grid	2D	Kolmogorov flow	ω_t and ν	ω_{t+1}
TorusVis	regular grid	2D	Kolmogorov flow	ω_t and ν and f_t	ω_{t+1}
Elasticity	point cloud	2D	hyper-elastic material	point cloud	stress
Airfoil	structured mesh	2D	transonic flow	mesh grid	Mach number
Plasticity	structured mesh	3D	plastic forging	die function	mesh grid over time

but with an extra deformation function in the input and output. Following Li et al. (2022), we use the geometric Fourier Transform in the first layer, and the inverse geometric Fourier Transform in the last layer. Both of these transforms involve defining a coordinate map, parameterized by a small neural network and learned end-to-end, that deforms the physical space of irregular geometry into a regular computational space which the standard Fourier transform can understand.

4 Datasets and evaluation settings

PDEs on regular grids The four Torus datasets on regular grids (TorusLi, TorusKochkov, TorusVis, and TorusVisForce, summarized in Table 1) are simulations based on Kolmogorov flows which have been extensively studied in the literature (Chandler & Kerswell, 2013). In particular, they model turbulent flows on the surface of a 3D torus (i.e., a 2D grid with periodic boundary conditions). TorusLi is publicly released by Li et al. (2021a) and is used to benchmark our model against the original FNO. The ground truths are assumed to be simulations generated by the pseudo-spectral Crank-Nicholson second-order method on 64x64 grids. All trajectories have a constant viscosity $\nu = 10^{-5}$ (Re $= 2000$), use the same constant forcing function, $f(x, y) = 0.1[\sin(2\pi(x+y))+\cos(2\pi(x+y))]$, and differ only in the initial field.

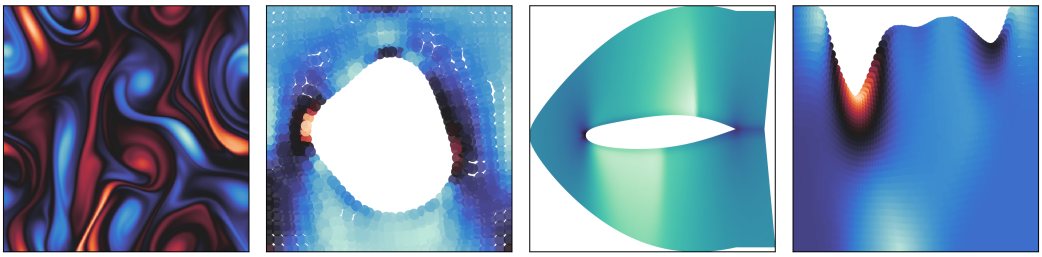
Using the same Crank-Nicolson numerical solver, we generate two further datasets, called TorusVis and TorusVisForce, to test the generalization of the F-FNO across Navier-Stokes tasks with different viscosities and forcing functions. In particular, for each trajectory, we vary the viscosity between 10^{-4} and 10^{-5}, and set the forcing function to

$$f(t, x, y) = 0.1 \sum_{p=1}^{2} \sum_{i=0}^{1} \sum_{j=0}^{1} \Big[\alpha_{pij} \sin\big(2\pi p(ix + jy) + \delta t\big) + \beta_{pij} \cos\big(2\pi p(ix + jy) + \delta t\big)\Big], \tag{9}$$

where the amplitudes α_{pij} and β_{pij} are sampled from the standard uniform distribution. Furthermore, δ is set to 0 in TorusVis, making the forcing function constant across time; while it is set to 0.2 in TorusVisForce, giving us a time-varying force.

Finally, we regenerate TorusKochkov (Fig. 2a) using the same settings provided by Kochkov et al. (2021) but with different initial conditions from the original paper (since the authors did not release the full dataset). Here the ground truths are obtained from simulations on 2048x2048 grids using the pseudo-spectral Carpenter-Kennedy fourth-order method. The full-scale simulations are then downsampled to smaller grid sizes, allowing us to study the Pareto frontier of the speed vs accuracy space (see Fig. 4a). TorusKochkov uses a fixed viscosity of 0.001 and a constant forcing function $\mathbf{f} = 4\cos(4y)\hat{x} - 0.1\mathbf{u}$, but on the bigger domain of $[0, 2\pi]$. Furthermore, we generate only 32 training trajectories to test how well the F-FNO can learn on a low-data regime.

PDEs on irregular geometries The Elasticity, Airfoil, and Plasticity datasets (final three rows in Table 1) are taken from Li et al. (2022). Elasticity is a point cloud dataset modeling the incompressible Rivlin-Saunders material (Pascon, 2019). Each sample is a unit cell with a void in the center of arbitrary shape (Fig. 2b). The task is to map each cloud point to its stress value. Airfoil models the transonic flow over an airfoil, shown as the white center in Fig. 2c. The neural operator would then learn to map each mesh location to its Mach number. Finally, the Plasticity data models the plastic forging problem, in which a die, parameterized by an arbitrary function and traveling at a constant speed, hits a block material from above (Fig. 2d). Here the task is to map the shape of the die to



(a) TorusKochkov (b) Elasticity (c) Airfoil (d) Plasticity

Figure 2: The F-FNO can handle a variety of PDEs on widely varying geometries including: (a) regular grids, (b) point clouds, (c)(d) structured meshes.

the 101×31 structured mesh over 20 time steps. Note that Plasticity expects a 3D output, with two spatial dimensions and one time dimension.

Training details For experiments involving the original FNO, FNO-TF (with teaching forcing), and FNO-M (with the Markov assumption), we use the same training procedure as Li et al. (2021a). For our own models, we train for 100,000 steps on the regular grid datasets and for 200 epochs for the irregular geometry datasets, warming up the learning rate to 2.5×10^{-3} for the first 500 steps and then decaying it using the cosine function (Loshchilov & Hutter, 2017). We use ReLU as our non-linear activation function, clip the gradient value at 0.1, and use the Adam optimizer (Kingma & Ba, 2015) 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 (Loshchilov & Hutter, 2019). In each operator layer on the torus datasets, we always throw away half of the Fourier modes (e.g., on a 64x64 grid, we keep only the top 16 modes). Models are implemented in PyTorch (Paszke et al., 2017) and trained on a single Titan V GPU.

Evaluation metrics We use the normalized mean squared error as the loss function, 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.

In addition to comparing the N-MSE directly, for TorusKochkov, we also compute the vorticity correlation, defined as

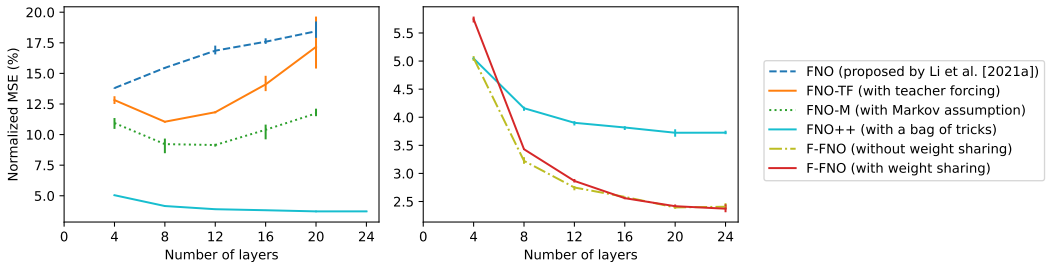
$$\rho(\omega, \hat{\omega}) = \sum_i \sum_j \frac{\omega_{ij}}{\|\omega\|_2} \frac{\hat{\omega}_{ij}}{\|\hat{\omega}\|_2},$$

and from which we measure the time until this correlation drops below 95%. To be consistent with prior work, we use the N-MSE to compare the F-FNO against the FNO and geo-FNO Li et al. (2021a; 2022), and the vorticity correlation to compare against Kochkov et al. (2021)'s work.

5 Results for Naiver-Stokes on a torus

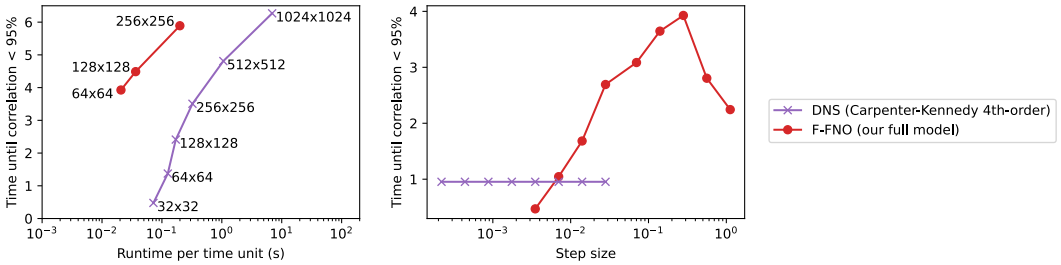
Comparison against FNO The performance on TorusLi is plotted in Fig. 3, with the raw numbers shown in Table A.3. We note that our method F-FNO is substantially more accurate than the FNO regardless of network depth, when judged by N-MSE. The F-FNO uses fewer parameters than the FNO, has a similar training time, but generally has a longer inference time. Even so, the inference time for the F-FNO is still up to two orders of magnitude shorter than for the Crank-Nicolson numerical solver.

In contrast to our method, Li et al. (2021a) do not use teacher forcing during training. Instead they use the previous 10 steps as input to 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 from the previous time step as input during training, leads to



(a) (b)

Figure 3: Performance (lower is better) of six model variants on TorusLi, with error bars showing the min and max values over three trials. In (a), we improve the original FNO (Li et al., 2021a) by using teacher forcing, the Markov property, among other techniques, which results in FNO++. In (b), we show the effect of Fourier factorization and weight sharing.



(a) (b)

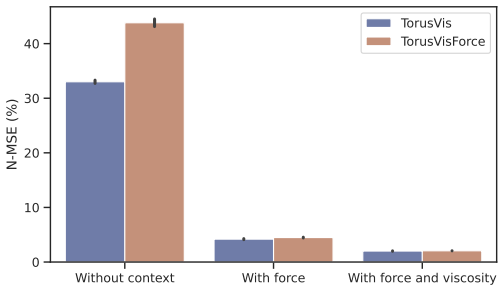
Figure 4: Performance of the F-FNO on TorusKochkov. In (a), we plot the time until the correlation with the ground truths in the test set drops below 95% on the y-axis, against the time it takes to run one second of simulation on the x-axis. In (b), we show how, on the validation set of TorusKochkov, given a fixed spatial resolution of 64x64, changing the step size has no effect on the numerical solver; however there is an optimal step size for the F-FNO at around 0.2.

a smaller N-MSE when the number of layers is less than 24. Furthermore, enforcing the first-order Markov property (FNO-M, dotted green line), where only one step of history is used, further improves the performance over FNO and FNO-TF. Including two or more steps of history does not improve the results.

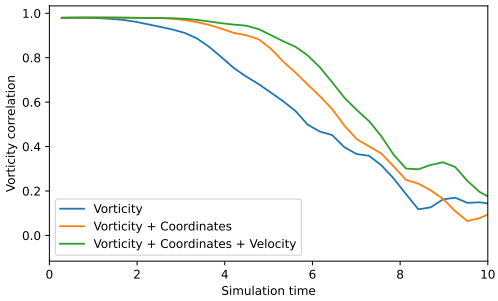
The models FNO, FNO-TF, and FNO-M do not scale with network depth, as seen by the increase in the N-MSE with network depth. These models even diverge during training when 24 layers are used. FNO++ does not suffer from this problem, as a result of a careful combination of: normalizing the inputs, adding Gaussian noise to the training inputs, using a cosine learning rate schedule, and placing residual connections after the activation functions. In particular, we find that adding a small amount of Gaussian noise to the normalized inputs helps to stabilize training. Without the noise, the validation loss at the early stage of training can explode.

Finally, if we use Fourier factorization (F-FNO, yellow dashed line), the error drops by an additional 35% ($3.73\% \rightarrow 2.41\%$) at 24 layers (Fig. 3b), while the parameter count is reduced by an order of magnitude. Sharing the weights in the Fourier domain (F-FNO, red line) makes little difference to the performance especially at deep layers, but it does reduce the parameter count by another order of magnitude to 1M (see Fig. A.1 and Table A.3).

Trade-off between speed and accuracy From Fig. 4a, we observe that our method F-FNO only needs 64x64 input grids to reach a similar performance to a 256x256 grid solved with DNS. At the same time, the F-FNO also achieves an order of magnitude speedup. While the highly specialized hybrid method introduced by Kochkov et al. (2021) can achieve a speedup closer to two orders of magnitude over DNS, the F-FNO takes a much more flexible approach and thus can be more easily adapted to other PDEs and geometries.



(a) Each group of bars shows F-FNO with a different set of input features. On both TorusVis and TorusVisForce, having the force field as an additional input reduces the error by an order of magnitude. Including the viscosity further halves the error. The error bars are the standard deviation from three trials.



(b) Effect of having the coordinates and velocity as additional input channels on TorusKochkov. Each of these features allows F-FNO's predictions to correlate with the reference solution for longer.

Figure 5: F-FNO can handle flexible input representations.

The improved accuracy of the F-FNO over DNS when both methods are using the same spatial resolution can be seen graphically in Fig. A.3. In this example, the F-FNO on a 128x128 grid produces a vorticity field which is visually much closer to the ground truth than DNS running on the same grid size. This is also clearly supported by comparing the time until correlation falls below 95% in Fig. 4a.

Optimal step size The F-FNO includes a step size parameter which specifies how many seconds of simulation time one application of the operator will advance. A large step size sees the model predicting far into the future possibly leading to large errors, while a small step size means small errors have many more steps to compound. We thus try different step sizes in Fig. 4b.

In numerical solvers, there is a close relationship between the step size in the time dimension and the spatial resolution. Specifically, the Courant-Friedrichs-Lewy (CFL) condition provides an optimal step size given a space discretization: $\Delta t = C_{\max} \Delta x / \|\mathbf{u}\|_{\max}$. This means that if we double the grid size, the solver should take a step that is twice as small (we follow this approach to obtain DNS's Pareto frontier in Fig. 4a). Furthermore, having step sizes smaller than what is specified by the CFL condition would not provide any further benefit unless we also reduce the distance between two grid points (see purple line in Fig. 4b). On the other hand, a step size that is too big (e.g., bigger than 0.05 on a 64x64 grid) will lead to stability issues in the numerical solver.

For the F-FNO, we find that we can take a step size that is at least an order of magnitude bigger than the stable step size for a numerical solver. This is the key contribution to the efficiency of neural methods. Furthermore, there is a sweet spot for the step size – around 0.2 on TorusKochkov– and unlike its numerical counterpart, we find that there is no need to reduce the step size as we train the F-FNO on a higher spatial resolution.

Flexible input representations F-FNO can be trained to handle Navier-Stokes equations with viscosities (in TorusVis) and time-varying forcing functions (in TorusVisForce) provided at inference time. Our model, when given both the force and viscosity, in addition to the vorticity, is able to achieve an error of 2% (Fig. 5a). If we remove the viscosity information, the error doubles. Removing the forcing function from the input further increases the error by an order of magnitude. This shows that the force has a substantial impact on the future vorticity field, and that the F-FNO can use information about the forcing function to make accurate predictions.

More generally, different datasets benefit from having different input features – Table A.7 shows the minimum set of features to reach optimal performance on each of them. We also find that having redundant features does not significantly hurt the model, so there is no need to do aggressive feature pruning in practice.

Table 2: Performance (N-MSE, lower is better) on point clouds (Elasticity) and structured meshes (Airfoil and Plasticity) between our F-FNO and the previous state-of-the-art geo-FNO (Li et al., 2022). Cells with a dash correspond to models which do not converge. The N-MSE is accompanied by the standard deviation from three trials. More detailed results are shown in Tables A.4 to A.6.

No. of layers	Elasticity		Airfoil		Plasticity	
	geo-FNO	F-FNO	geo-FNO	F-FNO	geo-FNO	F-FNO
4 layer	2.5 ± 0.1	3.16 ± 1.29	1.9 ± 0.4	0.79 ± 0.02	0.74 ± 0.01	0.48 ± 0.02
8 layer	3.3 ± 1.3	2.05 ± 0.01	1.4 ± 0.5	0.64 ± 0.01	0.57 ± 0.04	0.32 ± 0.01
12 layer	16.8 ± 0.7	1.96 ± 0.02	4.1 ± 4.4	0.62 ± 0.03	0.45 ± 0.03	0.25 ± 0.01
16 layer	16.3 ± 0.4	1.86 ± 0.02	-	0.61 ± 0.01	-	0.22 ± 0.00
20 layer	16.0 ± 0.7	1.84 ± 0.02	-	0.57 ± 0.01	-	0.20 ± 0.02
24 layer	15.9 ± 0.5	1.74 ± 0.03	-	0.58 ± 0.04	-	0.18 ± 0.00

Our experiments with different input representations also reveal an interesting performance gain from the *double encoding of information*. For example, even though there is a one-to-one map between the vorticity and velocity fields, have both as inputs improves the F-FNO's performance on TorusKochkov (Fig. 5b).

Furthermore, all datasets benefit from the coordinate encoding – i.e., having the (x, y) coordinates as two additional input channels – even if the positional information is already contained in the absolute position of grid points (indices of the input array). We hypothesize that these two positional representations are used by different parts of the F-FNO. The Fourier transform uses the absolute position of the grid points and thus the Fourier layer should have no need for the (x, y) positional features. However, the feedforward layer in the physical space is a pointwise operator and thus needs to rely on the raw coordinate values, since it would otherwise be independent of the absolute position of grid points.

6 Results for PDEs on point clouds and meshes

As shown in Table 2, the geo-FNO (Li et al., 2022), similar to the original FNO, also suffers from network scaling. It appears to be stuck in a local minimum beyond 8 layers in the Elasticity problem and it completely fails to converge beyond 12 layers in Airfoil and Plasticity. Plasticity is the only task in which the geo-FNO gets better as we go from 4 to 12 layers ($0.74 \rightarrow 0.45$). In addition to the poor scaling with network depth, we also find during our experiments that the geo-FNO can perform worse as we increase the hidden size H. This indicates that there might not be enough regularization in the model as we increase the model complexity.

Our F-FNO, on the other hand, continues to gain performance with deeper networks and bigger hidden size, reducing the prediction error by 31% on the Elasticity point clouds ($2.51 \rightarrow 1.74$) and by 57% on the 2D transonic flow over airfoil problem ($1.35 \rightarrow 0.58$). Our Fourier factorization particularly shines in the plastic forging problem, in which the neural operator needs to output a 3D array, i.e., the stress value on a 2D mesh over 20 time steps. As shown in Table A.6, our 24-layer F-FNO with 11M parameters outperforms the 12-layer geo-FNO with 57M parameters by 60% ($0.45 \rightarrow 0.18$).

7 Conclusion

The Fourier transform is a powerful tool to learn neural operators that can handle long-range spatial dependencies. By factorizing the transform, using better residual connections, and improving the training setup, our proposed F-FNO outperforms the state of the art on PDEs on a variety of geometries and domains. For future work, we are interested in examining equilibrium properties of generalized Fourier operators with an infinite number of layers, exploring other transforms such as the Discrete Cosine Transform (Poli et al., 2022), and checking if the universal approximation property (Kovachki et al., 2021a) still holds under Fourier factorization.

REFERENCES

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. ISSN 0027-8424. doi: 10.1073/pnas.1814058116. URL https://www.pnas.org/content/116/31/15344.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *arXiv preprint arXiv:2005.03180*, 2020.

Gary J. Chandler and Rich R. Kerswell. Invariant recurrent solutions embedded in a turbulent two-dimensional kolmogorov flow. *Journal of Fluid Mechanics*, 722:554–595, 2013. doi: 10.1017/jfm.2013.122.

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.

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pp. 59–70. Wiley Online Library, 2019.

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

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021. ISSN 0027-8424. doi: 10.1073/pnas.2101784118. URL https://www.pnas.org/content/118/21/e2101784118.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021a. URL http://jmlr.org/papers/v22/21-0806.html.

Nikola B. Kovachki, Zong-Yi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Animashree Anandkumar. Neural operator: Learning maps between function spaces. *ArXiv*, abs/2108.08481, 2021b.

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

Zong-Yi Li, Daniel Z. Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *ArXiv*, abs/2207.05209, 2022.

Zongyi Li, Nikola B. Kovachki, K. Azizzadenesheli, Burigede Liu, K. Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *ArXiv*, abs/2006.09535, 2020a.

Zongyi 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, 2020b.

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*, 2021a. URL https://openreview.net/forum?id=c8P9NQVtmnO.

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, 2021b.

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.

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

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 2022.

Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. Cfdnet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pp. 1–12, 2020.

João Paulo Pascon. Large deformation analysis of plane-stress hyperelastic problems via triangular membrane finite elements. *International Journal of Advanced Structural Engineering*, 2019.

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.

Michael Poli, Stefano Massaroli, Federico Berto, Jinkyoo Park, Tri Dao, Christopher Re, and Stefano Ermon. Transform once: Efficient operator learning in frequency domain. In *ICML 2022 2nd AI for Science Workshop*, 2022. URL https://openreview.net/forum?id=x1fNT5yj41N.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pp. 3424–3433. PMLR, 2017.

Kiwon Um, Robert Brand, Yun Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. *Advances in Neural Information Processing Systems*, 2020.

Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning incompressible fluid dynamics from scratch–towards fast, differentiable fluid models that generalize. *arXiv preprint arXiv:2006.08762*, 2020.

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.

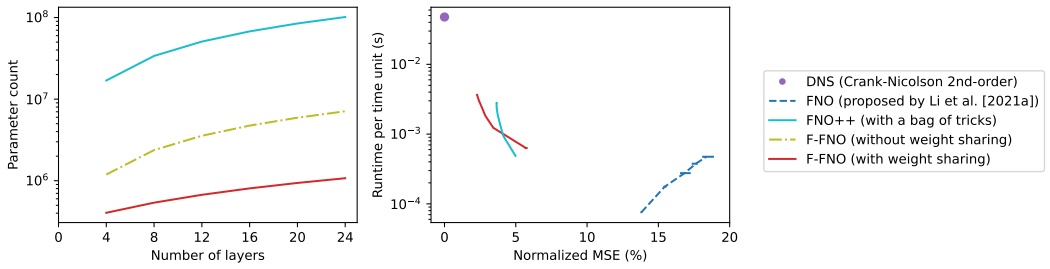
A APPENDIX

Table A.1: An overview of the four fluid dynamics datasets on regular grids. Our newly generated datasets, TorusVis and TorusVisForce, contain simulation data with a more variety of viscosities and forces than TorusLi (Li et al., 2021a) and TorusKochkov (Kochkov et al., 2021). Note that Li et al. (2021a) did not generate a validation set.

Dataset	Train / valid / test split	Trajectory length	Domain	Viscosity	Force varying across	
					samples	time
TorusLi	1000 / 0 / 200	20	$[0, 1]$	$\nu = 10^{-5}$		
TorusKochkov	32 / 4 / 4	34	$[0, 2\pi]$	$\nu = 10^{-3}$		
TorusVis	1000 / 200 / 200	20	$[0, 1]$	$\nu \in [10^{-5}, 10^{-4})$	✓	
TorusVisForce	1000 / 200 / 200	20	$[0, 1]$	$\nu \in [10^{-5}, 10^{-4})$	✓	✓

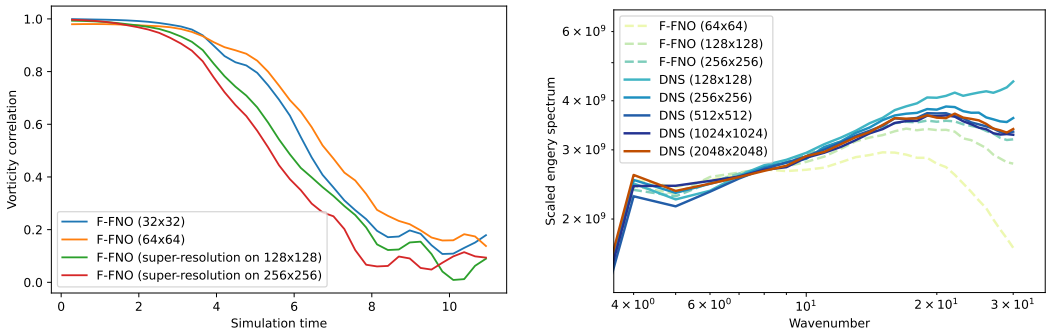
Table A.2: An overview of the three PDE datasets on irregular geometries. These datasets were generated by Li et al. (2022).

Dataset	Train	Valid	Test	Governing equation	Problem dimension
Elasticity	1000	200	200	Equation of a solid body	point cloud on 2D unit cell
Airfoil	1000	200	200	Euler's equation	2D structured mesh
Plasticity	827	80	80	Equation of a solid body	2D structured mesh over 1D time



(a) (b)

Figure A.1: The resource usage of four model variants, in terms of (a) the parameter count and (b) inference time (the time it takes to run one second of simulation). Error bars, when applicable, show the min and max values over three trials. In (b), as we move along a line, we increase the number of layers. We observe that only our model variants (F-FNO) have the desired slope, that is, as we use more resources (increasing the inference time), we obtain better predictions.



(a) Zero-shot super-resolution performance of F-FNO. We train the model on 32x32 and 64x64 grids of TorusKochkov, and evaluate on the larger 128x128 and 256x256 grids. We observe some degradation in the correlation with the ground truths on unseen grid sizes.

(b) Energy spectra of F-FNO and DNS on various grid sizes. The spectra are computed by averaging the kinetic energy for each wavenumber between $t = 12$ and $t = 34$, when the predictions from all methods have decorrelated with the ground truths.

Figure A.2: F-FNO can perform zero-shot superresolution and capture the energy spectrum of DNS.

Zero-shot super-resolution In Fig. A.2a, we train the F-FNO once on 32x32 and 64x64 grids from TorusKochkov, and then perform inference and evaluation on 128x128 and 256x256 grids. This extends the super-resolution setting presented by Li et al. (2021a) as they only worked on simple PDEs such as the 1D Burger's equation and the 2D Darcy flow. We find that although the F-FNO can do zero-shot super-resolution – unlike a traditional CNN which by design cannot even accept inputs of variable size – its performance does degrade on grid sizes not seen during training. This is seen by the lower vorticity correlation of the super-resolution F-FNO settings in Fig. A.2a. We posit that the super-resolution performance could be improved by training on a variety of grid sizes (e.g., by downsampling each training example to a random size). We leave such exploration for future work.

Capturing the energy spectrum In addition to having a high vorticity correlation, a good model should also produce predictions with an energy spectrum similar to the most accurate numerical methods. Given the Fourier transform of a velocity field $\hat{\mathbf{u}} = \text{FFT}(\mathbf{u})$, we can compute, for each wavenumber k, the corresponding kinetic energy as $E(k) = \frac{1}{2}\|\hat{\mathbf{u}}_k\|^2$. Fig. A.2b shows the energy spectrum of both the F-FNO and DNS at different resolutions. These multiple DNS resolutions are included both as a reference solution in the case of DNS 2048x2048, and to demonstrate that increasing the resolution of DNS further is not likely to substantially change the energy spectrum.

We observe that compared to DNS on 2048x2048, the F-FNO trained on 64x64 grids produces an energy spectrum that has substantially lower energy at high wavenumbers. This is expected as at this spatial resolution we only select the top 16 modes in each Fourier layer. Even so, the F-FNO can still capture the long term trend much better than running DNS on a grid four times its size (see Fig. 4a). As we select more Fourier modes on bigger grids (top 32 modes on 128x128 grids and top 64 modes on 256x256 grids), the energy spectrum produced converges towards that of the reference solution (DNS on 2048x2048). This gives some indication that the F-FNO is able to accurately predict both high and low frequency details.

Table A.3: Detailed performance on TorusLi. These results are used to generate Fig. 3 in the main paper. We run three trials for each experiment, each with a different random seed. We report the mean N-MSE from the three trials, along with the min and max value. A dash indicates that the data is not available.

	No. of layers	No. of parameters	N-MSE (%)			Training time (h)
			Mean	Min	Max	
ResNet (Li et al., 2021a)	-	266,641	27.53	-	-	
TF-Net (Li et al., 2021a)	-	7,451,724	22.68	-	-	
U-Net (Li et al., 2021a)	-	7,451,724	19.82	-	-	
FNO (Li et al., 2021a)	4	414,517	15.56	-	-	
FNO (reproduced)	4	926,357	13.80	13.75	13.83	2
	8	1,849,637	15.45	15.40	15.50	3
	12	2,772,917	16.86	16.55	17.27	5
	16	3,696,197	17.59	17.41	17.87	6
	20	4,619,477	18.44	17.91	19.24	7
FNO-TF (with teacher forcing)	4	926,357	12.82	12.50	13.13	2
	8	1,849,637	11.05	10.98	11.10	3
	12	2,772,917	11.83	11.74	11.91	5
	16	3,696,197	14.10	13.55	14.80	6
	20	4,619,477	17.17	15.40	19.64	7
FNO-M (with Markov assumption)	4	926,177	10.94	10.47	11.35	1
	8	1,849,457	9.22	8.48	9.67	2
	12	2,772,737	9.15	9.02	9.25	3
	16	3,696,017	10.39	9.61	10.81	4
	20	4,619,297	11.73	11.50	12.12	4
FNO++ (with bags of tricks)	4	16,919,746	5.05	5.02	5.06	1
	8	33,830,594	4.16	4.11	4.19	2
	12	50,741,442	3.90	3.85	3.93	3
	16	67,652,290	3.82	3.77	3.84	4
	20	84,563,138	3.72	3.65	3.79	5
	24	101,473,986	3.73	3.70	3.76	5
F-FNO (without weight sharing)	4	1,191,106	5.05	5.00	5.09	1
	8	2,373,314	3.22	3.17	3.29	2
	12	3,555,522	2.75	2.70	2.77	3
	16	4,737,730	2.58	2.57	2.59	4
	20	5,919,938	2.39	2.37	2.42	5
	24	7,102,146	2.41	2.37	2.47	6
F-FNO (with weight sharing)	4	404,674	5.74	5.69	5.79	1
	8	538,306	3.43	3.42	3.44	2
	12	671,938	2.87	2.84	2.90	3
	16	805,570	2.56	2.54	2.57	4
	20	939,202	2.42	2.38	2.45	5
	24	1,072,834	2.37	2.31	2.45	6

Table A.4: Detailed performance on Airfoil. These results are more detailed version of Table 2 in the main paper. We run three trials for each experiment, each with a different random seed. We report the mean N-MSE from the three trials, along with the min and max value.

	No. of layers	No. of parameters	N-MSE (%)			Training time (h)
			Mean	Min	Max	
geo-FNO (reproduced)	4	2,368,033	1.87	1.40	2.27	4
	8	4,731,553	1.35	1.02	2.00	5
	12	7,095,073	4.11	0.92	10.29	4
F-FNO (with weight sharing)	4	535,810	0.98	0.90	1.03	0.4
	8	669,442	0.72	0.70	0.75	0.7
	12	803,074	0.68	0.66	0.70	1
	16	936,706	0.67	0.63	0.70	1
	20	1,070,338	0.64	0.63	0.66	2
	24	1,203,970	0.66	0.60	0.70	2
F-FNO (without weight sharing)	4	1,715,458	0.79	0.76	0.82	3
	8	3,421,954	0.64	0.63	0.65	4
	12	5,128,450	0.62	0.59	0.67	5
	16	6,834,946	0.61	0.59	0.62	5
	20	8,541,442	0.57	0.56	0.58	4
	24	10,247,938	0.58	0.56	0.64	4

Table A.5: Detailed performance on Elasticity. These results are more detailed version of Table 2 in the main paper. We run three trials for each experiment, each with a different random seed. We report the mean N-MSE from the three trials, along with the min and max value. Note that for a given layer, our F-FNO (whether with weight sharing or without) has slightly more parameters than the geo-FNO. This is due to the F-FNO using a bigger hidden size H. We find that on the geo-FNO, increasing its hidden size does not necessarily translate to a better performance.

	No. of layers	No. of parameters	N-MSE (%)			Training time (h)
			Mean	Min	Max	
geo-FNO (reproduced)	4	1,546,403	2.51	2.43	2.59	0.4
	8	2,730,659	3.30	2.32	5.10	0.5
	12	3,914,915	16.76	16.17	17.72	0.7
F-FNO (with weight sharing)	4	2,681,475	3.55	2.36	5.84	0.2
	8	2,765,187	2.23	2.18	2.29	0.2
	12	2,848,899	2.12	2.10	2.16	0.3
	16	2,932,611	2.08	2.06	2.10	0.4
	20	3,016,323	2.04	1.99	2.07	0.5
	24	3,100,035	1.97	1.94	2.01	0.6
F-FNO (without weight sharing)	4	3,205,763	3.16	2.23	4.98	1
	8	4,338,051	2.05	2.04	2.06	1
	12	5,470,339	1.96	1.93	1.98	2
	16	6,602,627	1.86	1.83	1.88	2
	20	7,734,915	1.84	1.82	1.86	2
	24	8,867,203	1.74	1.70	1.78	2

Table A.6: Detailed performance on Plasticity. These results are more detailed version of Table 2 in the main paper. We run three trials for each experiment, each with a different random seed. We report the mean N-MSE from the three trials, along with the min and max value.

	No. of layers	No. of parameters	N-MSE (%)			Training time (h)
			Mean	Min	Max	
geo-FNO (reproduced)	4	18,883,492	0.74	0.73	0.75	2
	8	37,762,084	0.57	0.55	0.63	4
	12	56,640,676	0.45	0.41	0.49	5
F-FNO (with weight sharing)	4	568,968	0.58	0.57	0.60	4
	8	702,600	0.50	0.46	0.52	8
	12	836,232	0.44	0.42	0.48	12
	16	969,864	0.40	0.36	0.44	16
	20	1,103,496	0.34	0.31	0.37	19
	24	1,237,128	0.30	0.28	0.35	21
F-FNO (without weight sharing)	4	1,846,920	0.48	0.47	0.51	4
	8	3,684,488	0.32	0.31	0.34	8
	12	5,522,056	0.25	0.24	0.26	12
	16	7,359,624	0.22	0.21	0.22	16
	20	9,197,192	0.20	0.18	0.22	20
	24	11,034,760	0.18	0.17	0.18	24

Table A.7: The F-FNO is flexible in its input representation. We find that different datasets benefit from having different features. Shown here is the optimal input combination for each dataset on the torus.

Dataset	Vorticity	Velocity	Coordinates	Viscosity	Forcing
TorusLi	✓		✓		
TorusKochkov	✓	✓	✓		
TorusVis	✓		✓	✓	
TorusVisForce	✓		✓	✓	✓

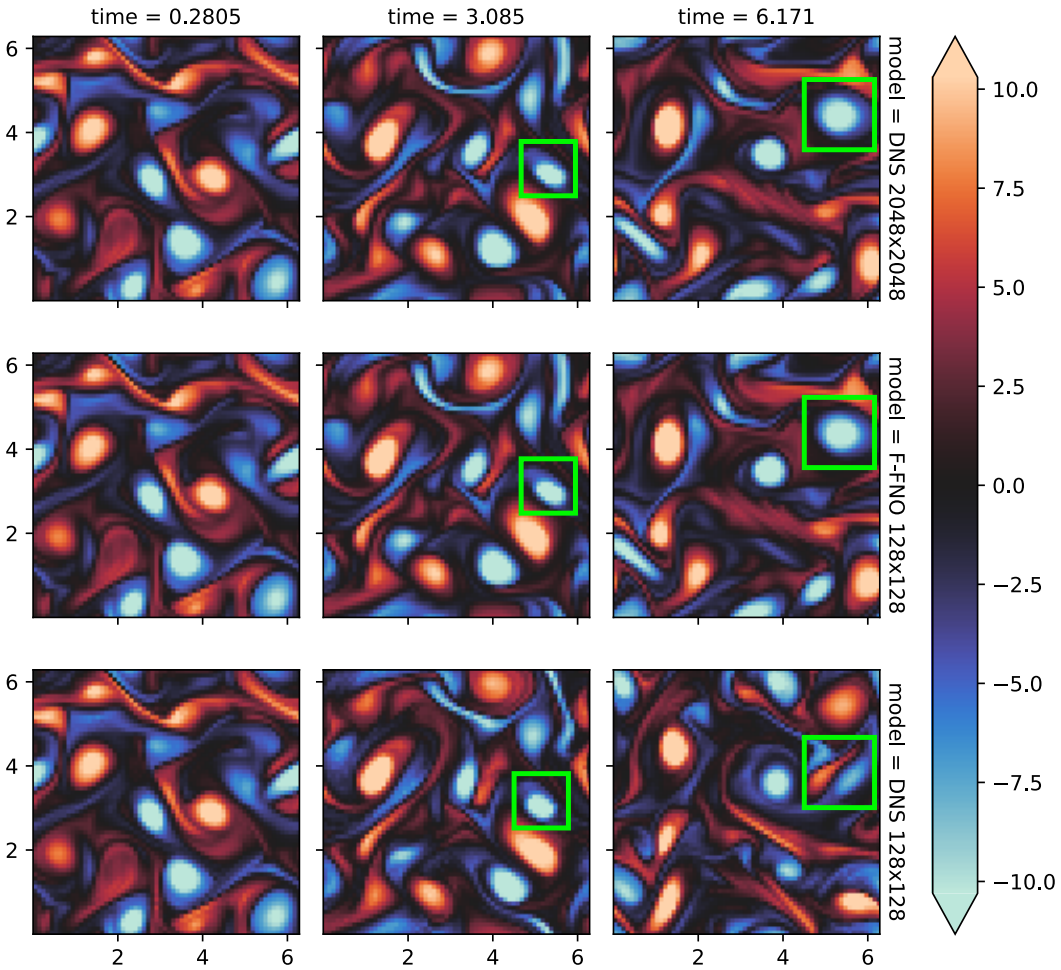


Figure A.3: The vorticity fields predicted by the F-FNO trained on 128x128 grids (middle row) correlates with the ground truths (top row) for longer than if we run DNS on the same spatial resolution (bottom row). This is especially evident after 6 seconds of simulation time (compare the green boxes).