Prediction of Fluid Flows Using a Fourier Neural Operator

Gustav Burman, Karl Lundgren, Erik Norlin, and Mattias Wiklund
Chalmers University of Technology
(Dated: July 11, 2023)

In this examination, a Fourier neural operator (FNO) was constructed with the purpose of predicting two-dimensional fluid flows described by Navier−Stokes equations. By construction, the FNO is spatio-temporally resolution invariant. This implies that the model, despite being trained on low-resolution flows, can predict flows with higher high-resolution. To test this property of the FNO, it was trained on data with a spatial resolution of 32×32 and time step of 1 s, and evaluated on data with higher spatial and temporal resolution. From this study, the resolution invariance in space could be confirmed, with the FNO having a mean absolute error (MAE) of 0.0054 to 0.0087 on all test data sets, ranging from 64×64 to 1024×1024. However, the test of the resolution invariance in time provided worse results, contradictory to the theory. To perform a comparative analysis for the spatial resolution invariance, high-resolution data was downsampled, run through the FNO and upsampled, using bicubic interpolation. This resulted in mean absolute errors between around 85 % and 285 % higher for the bicubic interpolation, indicating the advantages of using the FNO. Future research could for example include developing the loss function to better capture the physics, compare to other partial differential equation solvers and try to predict more turbulent flows.

I. INTRODUCTION

In this section, fluid dynamics and the Fourier neural operator are introduced, as well as the aim of the project.

A. Background

Fluid dynamics is the study of motion of liquids and gases and has been investigated for a long time to better understand, and accurately model, the behaviour and properties of fluids. Modelling fluids in engineering allows to design better mechanical systems, optimize production flows, better understand how pollutant and heat spread through the environment, and more. Thus, modelling fluid dynamics accurately and efficiently is of great importance.

To simulate fluid flows of partial differential equations (PDE), traditional methods are most often turned to, such as the finite difference method (FDM), the finite volume method (FVM) and the finite element method (FEM). These methods are rigorous, reliable and as accurate as one chooses to make them. However, these methods are computationally expensive, and simulations can take unreasonable long time. Also, a new simulation has to be run for every new flow to analyse, which accumulates even more time. To find more efficient ways to simulate fluid dynamics accurately are therefore of interest.

With the rise of advancements of deep neural networks in the recent years, creative ways of making neural networks learn PDEs have been developed and become potential candidates as supplementary methods in addition to the traditional ones to improve efficiency. One such example is the physics-informed neural network (PINN) introduced by M. Raissi et al. (2019) [1], which aims to learn the PDE that is desired to be simulated. One therefore needs simulated samples from a traditional method as a training data set for a PINN, and once the PINN is trained, it is able to predict the flow of the PDE it is trained on. This allows one to predict new flows without using the expensive traditional methods. The main drawback of a PINN, however, is that it is trained on only one instance of a given PDE. This makes it neither spatially nor temporally resolution invariant, meaning that once it is trained, it can only predict a flow of the resolution it was trained on since trainable parameters of a PINN are mapped directly to each point on the spatial-temporal grid of the flow [1].

A different approach to make a neural network learn a PDE was introduced by Zongyi Li et al. (2021) [2], which is to use a physics-informed neural operator (PINO) that, similarly to a PINN, learns the PDE it is trained on but instead learns many instances of the PDE. This is done by using Fourier layers [3], instead of feed forward layers that a PINN uses.

A Fourier layer places trainable parameters in the Fourier domain. This leads the model to learn global frequencies of the data, which are completely independent of the chosen resolution. As a consequence, the PINO can learn global structures of the data, and is thus able to perform so called zero-shot super resolution, meaning it can be used for any arbitrary resolution no matter what resolution it was trained on. This implies that one can train the model on a low-resolution data set and evaluate it on a high-resolution data set. One can make a slight simplification to a PINO by making it a Fourier neural operator (FNO), which is also resolution invariant but does not inherently learn the PDE itself [2], [3].

B. Aim of the Project

The objective in this project was to investigate the resolution invariance of an FNO predicting fluid flows in two spatial dimensions given an initial sequence of discretiza-

tions described by the Navier−Stokes equations. To this end, an FNO was implemented and trained on a spatial resolution of 32×32, with the physical time 1 s between each frame. The FNO was then evaluated on a set of different resolutions, varying both spatially and temporally. Lastly, the spatial resolution invariance of the FNO was evaluated by comparing a high-resolution prediction with an upsampling of a low-resolution prediction using bicubic interpolation.

II. THEORY

A Fourier neural operator was implemented to predict fluid dynamics. These two concepts are explained here.

A. Fluid Dynamics

The data used to train and evaluate the FNO was retrieved from a simulation of the Navier−Stokes equation for a viscous, incompressible fluid in two dimensions. The domain is a unit torus ($x, y \in (0, 1)$), with periodic boundary conditions. The incompressibility implies that material density is constant and, equivalently, that the divergence of the flow velocity $\vec{u}$ is zero, i.e., $\nabla \cdot \vec{u} = 0$. In general, an incompressible flow is described by the Navier−Stokes equation [3],

$$\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} = \frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = \nu \nabla^2 \vec{u} + \frac{1}{\rho}\nabla p + \widetilde{f}, \tag{1}$$

where $\mathrm{D}/\mathrm{D}t$ is the material derivative, $\vec{u}$ is the flow velocity, ν is the kinematic viscosity of the fluid, ρ is the density, p is the pressure and $\widetilde{f}$ represents external forces.

For some applications, it can be preferable to work with vorticity of the flow instead of the velocity. The vorticity $\vec{\omega}$ is defined as the curl of the flow velocity, i.e., $\vec{\omega} = \nabla \times \vec{u}$. Physically, it can be seen as the local spinning motion of the fluid, as seen from a point moving along the flow. In a two-dimensional flow, where the flow is independent of the z-coordinate in a Cartesian coordinate system, the vorticity is always perpendicular to the flow. The vorticity is therefore $\vec{\omega} = \omega \hat{z}$ and can for simplicity be considered as a scalar field.

To write Equation (1) on vorticity form, the curl operator is applied on both sides. This gives

$$\nabla \times \frac{\partial \vec{u}}{\partial t} + \nabla \times (\vec{u} \cdot \nabla)\vec{u} = \nu \nabla \times \nabla^2 \vec{u} + \frac{1}{\rho}\nabla \times \nabla p + \nabla \times \widetilde{f}. \tag{2}$$

Using the following vector calculus identifies,

$$\begin{cases} (\vec{u} \cdot \nabla)\vec{u} = 1/2 \nabla(\vec{u} \cdot \vec{u}) - \vec{u} \times \vec{\omega} \\ \nabla^2 \vec{u} = \nabla(\nabla \cdot \vec{u}) - \nabla \times \vec{\omega} \\ \nabla \times \nabla \phi = 0, \end{cases} \tag{3}$$

where ϕ is a general scalar field, the equation can be simplified to

$$\frac{\partial \vec{\omega}}{\partial t} - \nabla \times (\vec{u} \times \vec{\omega}) = -\nu \nabla \times \nabla \times \vec{\omega} + f, \tag{4}$$

where $f = \nabla \times \widetilde{f}$. Using $\nabla \cdot \vec{u} = 0$ and $\nabla \cdot \vec{\omega} = 0$, together with,

$$\begin{cases} \nabla \times (\vec{u} \times \vec{\omega}) = (\vec{\omega} \cdot \nabla)\vec{u} - (\vec{u} \cdot \nabla)\vec{\omega} \\ \nabla \times (\nabla \times \vec{\omega}) = -\nabla^2 \vec{\omega}, \end{cases} \tag{5}$$

this simplifies to

$$\frac{\partial \vec{\omega}}{\partial t} - (\vec{\omega} \cdot \nabla)\vec{u} + (\vec{u} \cdot \nabla)\vec{\omega} = \nu \nabla^2 \vec{\omega} + f. \tag{6}$$

This is the general vorticity equation for a viscous, incompressible flow [3]. For a two-dimensional planar flow, $(\vec{\omega} \cdot \nabla)\vec{u} = 0$ and, as stated earlier, the vorticity can be seen as a scalar field. This implies $(\vec{u} \cdot \nabla)\omega = \vec{u} \cdot \nabla\omega$. Finally, this results in

$$\frac{\mathrm{D}\omega}{\mathrm{D}t} = \frac{\partial \omega}{\partial t} + \vec{u} \cdot \nabla\omega = \nu \nabla^2 \omega + f. \tag{7}$$

Physically, the material derivative describes the rate of change of the vorticity for the moving fluid. The change can either depend on either lack of stability of movement ($\partial\omega/\partial t$) or convection ($\vec{u} \cdot \nabla\omega$). The first term on the right-hand side accounts for diffusion due to viscosity effects, and the forcing function accounts for external forces and boundary conditions.

B. Fourier Neural Operator

A Fourier neural operator consists of three distinct parts. One dense layer P that lifts the input to a higher dimension, one or several Fourier layers $L_{\mathcal{F},1}, L_{\mathcal{F},2}, ..., L_{\mathcal{F},n}$, and finally another dense layer Q that drops the dimensionality to the desired output.

Figure 1 shows a schematic of a Fourier layer. The layer consists of two parts. One global integral operator (top row in the figure) which consists of a Fourier transform of the data ($\mathcal{F}$), a linear transform (L_1) in Fourier space and an inverse Fourier transform ($\mathcal{F}^{-1}$) back to real space; and one linear transform in real space (L_2). The sum of these two transformations is then fed to a nonlinear activation function working locally. This way, the operator learns both global structures from the Fourier domain and local structures in real space.

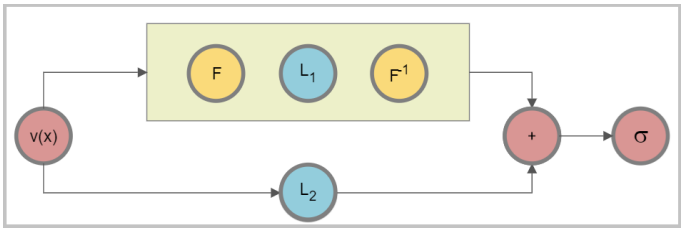

FIG. 1: Starting from input v, apply a Fourier transform, a linear transform, and the inverse Fourier transform. Add this to a linear transform in the real space and feed the resulting sum to a local activation function.

When implementing a Fourier layer, it is usually possible to discard the higher modes in the Fourier domain for computational efficiency and to avoid overfitting, as some higher modes actually might just be noise. The linear transform L_2 has a two-fold role. It helps recovering the dropped modes, and it helps regulating non-periodic boundary conditions, in case such are applied [3].

The overall structure of the operator is

$$u = (Q \circ L_{\mathcal{F},n} \circ ... \circ L_{\mathcal{F},1} \circ P)v, \tag{8}$$

where u is the output of the operator and v is the input data. The power of the operator comes from the Fourier layers, which combine linear global operators and local non-linear activation functions.

III. METHOD

The resolution invariance of the FNO was evaluated in both the spatial and temporal domain. In the spatial domain, a comparison was then done to a bicubic interpolation.

A. Generating the Data Sets

In this project, a data set for a two-dimensional incompressible fluid flow consisting of 5,000 unique flows was generated for training. The initial conditions for the vorticity were generated using Gaussian random fields and the forcing function was set to $f(x, y) = 0.1(\sin(2\pi(x + y)) + \cos(2\pi(x + y)))$. For generating the time-series, a Crank−Nicolson pseudo-spectral method was applied with a time step of 10^{-4} s [4], [5]. Note that the solution was recorded every $t = 1$ s, effectively making the temporal resolution of the training data 1 frame per second. The data was then split into 80 % training, 15 % validation and 5 % test data.

TABLE I: The different data sets used. The spatial resolution is given by number of pixels and the temporal resolution by the sample rate [Hz]. The model was trained on data set 1, with spatial resolution 32×32 with a sample rate of 1 Hz.

Data set	Spatial	Temporal [Hz]	No. flows
1	32	1	5,000
2	64	1	40
3	128	1	20
4	256	1	10
5	512	1	10
6	1024	1	10
7	32	2	40
8	32	10	40
9	32	20	40

To evaluate the resolution invariance of the network, a total of eight smaller data sets with varying spatial and temporal resolutions were also generated. These data sets consisted of 10−40 unique flows, with spatial resolutions 32×32, 64×64, 128×128, 256×256, 512×512 and 1024×1024. The temporal resolution was also varied, with a fixed spatial resolution, taking values of 1, 2, 10 and 20 frames per second. A summary of the evaluated data sets can be seen in Table I.

The input data fed to the FNO can be visualized as a spatio-temporal cuboid, as can be seen in Figure 2. Each point in the cuboid represents a 13-dimensional array, consisting of vorticity $\omega(x, y, t)$ for the first 10 seconds $\forall\,(x, y)$ and the coordinate (x, y, t). This 13-dimensional array was used as the input to the network, while the Fourier transform was taken along the x, y and t dimensions of the cuboid. For the output data of the FNO, a similar visualization can be applied. This time, however, each point is represented by the solution $\omega(x, y, t)$ in each coordinate, rather than a 13-dimensional array.

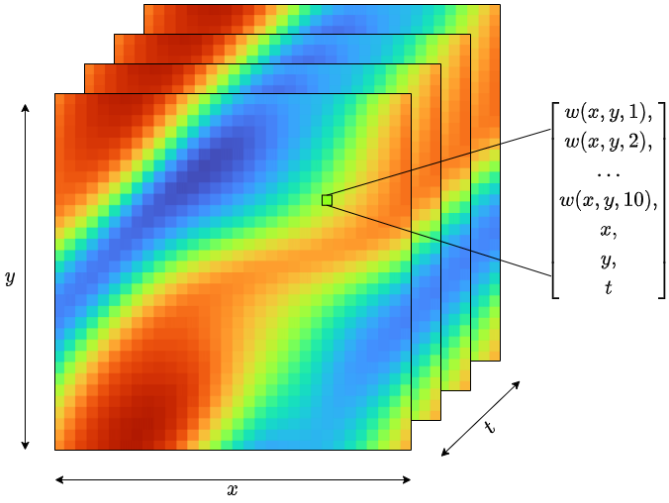

FIG. 2: The input data as a cuboid with two spatial dimensions and one temporal dimension. Each point in the cuboid is represented by an array consisting of the vorticity $\omega(x, y, t)$ for the first 10 seconds for each spatial point, together with the coordinate (x, y, t).

B. Implementing the FNO

An FNO model was implemented in PyTorch Lightning as this is a convenient package for building, training, and evaluating neural network architectures. The Fourier layers of the model were implemented as instances of a custom layer built from the ground up since the Fourier layer is not yet part of of the PyTorch API.

The implemented architecture of the FNO consisted of four Fourier layers, which was deemed sufficient for flow prediction of this task. Convolutional layers with kernel sizes of one were added after the inverse Fourier transformation as dense layers, which helps reconstructing the higher frequencies and improves performance of the net-

work. The input to the FNO was of four dimensions, including the time sequence t, as Figure 2 shows. This means that the network learns the entire time sequence in one go, in contrast to traditional recurrent neural networks (RNNs). Most importantly, this allows the FNO to perform a three-dimensional Fourier transform (along the x, y and t axes), which allows the network to become resolution invariant both spatially and temporally. The higher modes in Fourier space were dropped to increase performance, since the network has the capacity to reconstruct these modes anyway.

Algorithm 1 shows a description of how the FNO handles forward propagation of the architecture just described. Fourier(x) represents the Fourier transform followed by a linear transformation and lastly an inverse transform (upper signal in Figure 1), MLP(x) represents a multi-layer perceptron with two dense layers and $W(x)$ is another dense layer, and GELU(x) is the Gaussian error linear unit, used as an activation function after the first three Fourier layers. $p(x)$ is a Linear input layer which takes in the 13-dimensional input vector, and $q(x)$ is an output MLP layer with a single output $\omega(x, y, t)$ for each coordinate.

Algorithm 1: Forward propagation

$x = p(x)$
for *i = 1, 2, 3* **do**
  $x_1 = \text{Fourier}_i(x)$
  $x_1 = \text{MLP}_i(x_1)$
  $x_2 = W_i(x)$
  $x = x_1 + x_2$
  $x = \text{GELU}(x)$
end
$x_1 = \text{Fourier}_4(x)$
$x_1 = \text{MLP}_4(x_1)$
$x_2 = W_4(x)$
$x = x_1 + x_2$
$x = q(x)$
return x

The mean absolute error (MAE), see Equation (9), was used as loss function for both training and evaluation of different resolutions. The error was calculated as the average error of every point in the data cuboid. Here $\hat{y}$ is the output of the operator at a certain point, y is the ground truth of the simulation at the same point and S is the entire simulation containing n individual points.

$$\text{Error} = \frac{1}{n} \sum_{S} |\hat{y} - y| \,. \tag{9}$$

For a description of the hyperparameters used in the training, see Appendix A.

IV. RESULTS

The model was trained on data with spatial resolution 32×32 and sample rate 1 Hz. To evaluate spatial resolution invariance, the FNO was evaluated on spatial resolution 32×32, 64×64, 128×128, 256×256, 512×512 and 1024×1024, respectively, with a fixed temporal resolution of 1 Hz as sample rate.

Figure 3 shows evaluation for the case of 1024×1024. As can be seen, the network can approximate all points in the higher resolution quite well even though it was trained on a grid with resolution 32×32. This demonstrates the resolution invariance of the FNO.

The analysis of the spatial resolution invariance can be seen in Figure 5. The error margin corresponds to a 95 % confidence interval. As can be seen, the FNO performs similarly on all different spatial resolutions, showing that the FNO is spatially resolution invariant. For a comparative analysis, the FNO was used to predict a flow on the same resolution it was trained on (32×32), and then the resulting flow was upsampled using bicubic interpolation. This was done to infer the same prediction error into both methods as a bicubic interpolation only performs upsampling. An overview of this comparison can be seen in Figure 6 in Appendix B.

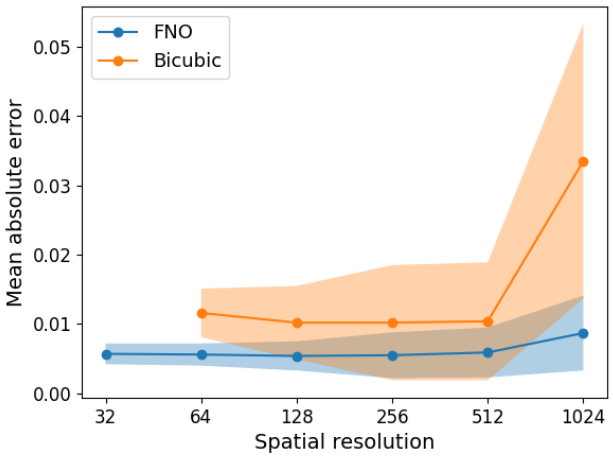

FIG. 5: The blue graph displays the MAE obtained from using the FNO model, trained on a spatial resolution 32×32 and temporal resolution 1 Hz, to predict flows with different resolutions. The orange graph displays the MAE for predicting a downsampled (to 32×32) flow with the FNO, followed by an upsampling using bicubic interpolation. Spatial resolutions varying from 32×32 to 1024×1024 were tested, with a constant temporal resolution of 1 Hz. The error bands correspond to 95 % confidence intervals. Note that for 32×32, no up- and downsampling was possible, since this was the lowest resolution tested.

As can be seen, the MAE for the bicubic interpolation was roughly twice as high up to 512×512, and nearly four times as high for 1024×1024. This result is of course dependant on which method is used for predicting the flow for the lower resolution, but still gives some indication

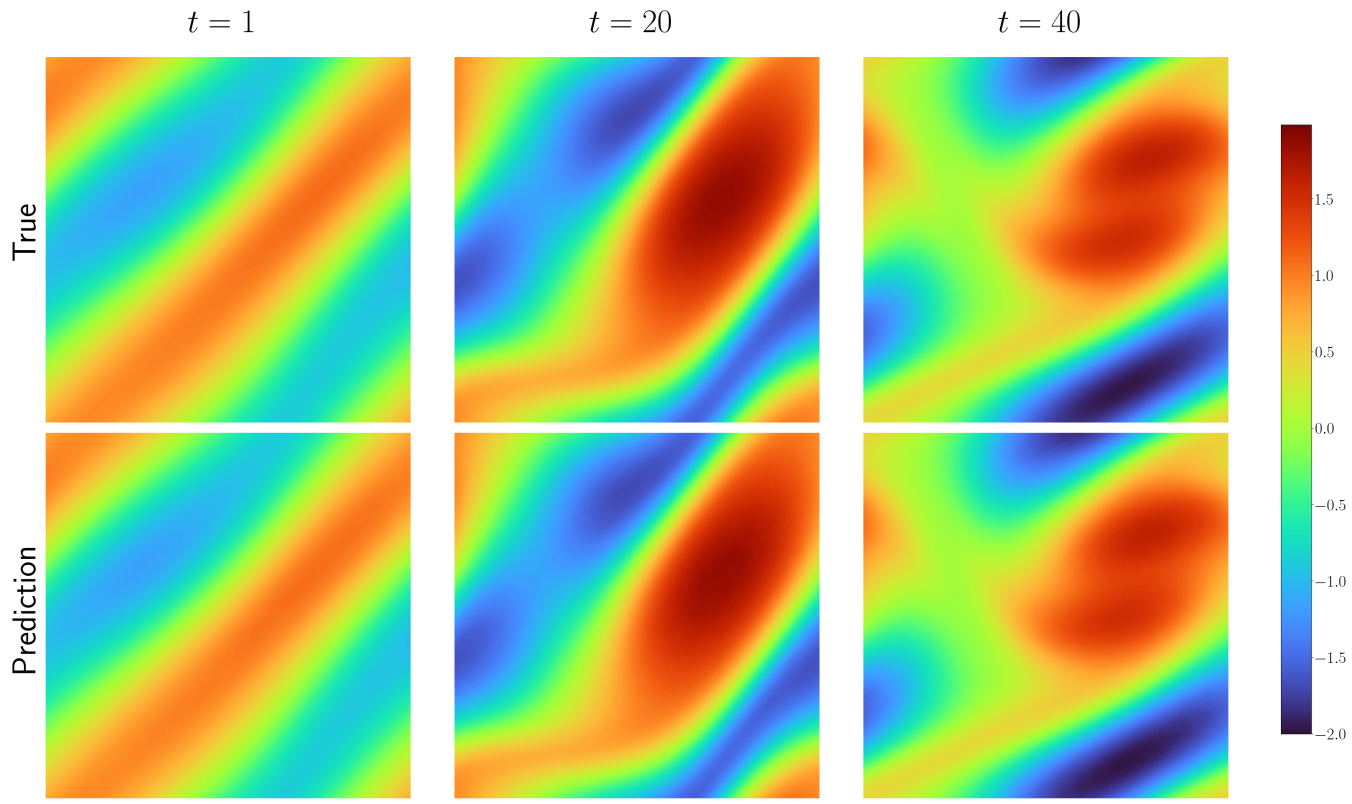

FIG. 3: The FNO shows here that it can predict fluid flow with zero-shot super resolution of spatial dimensions 1024×1024, despite being trained on spatial dimensions 32×32. The flow stretches over 50 seconds where the sequence of the first 10 seconds is the input to the FNO. t here corresponds to the predicted time into the future in seconds, and the colour bar shows the vorticity values. Note that the boundary conditions are periodic, this should not be confused with that the flows continue into one another.

that an FNO can be quite useful in predicting high resolution flows. Figure 4 shows how the absolute error of a flow predicted and upscaled by the FNO compares to a bicubic interpolation from spatial resolution 32×32 to 1024×1024 at fixed time step. It is clear from this that the FNO is indeed better at upsampling than bicubic interpolation. For numerical values of these results, see Appendix C.

TABLE II: The mean absolute error obtained from evaluating the model on different temporal resolutions. Sampling frequencies of 2 Hz, 10 Hz, and 20 Hz were tested, but all of them gave a significantly higher error than sampling with 1 Hz. All data sets were generated with a spatial resolution of 32×32.

Temporal res. [Hz]	MAE
1	0.0057 ± 0.0015
2	0.1318 ± 0.0404
10	0.3616 ± 0.0946
20	0.3934 ± 0.1022

Moreover, the temporal resolution invariance was evaluated on a fixed spatial resolution of 32×32 and with sample rate of 2 Hz, 10 Hz, and 20 Hz, respectively. As is shown in Table II, all of the increased sample rates gave a much higher error than using a sample rate of 1 Hz. This result is not expected from theory as the network treats the temporal dimension the same as spatial dimensions, and is likely because of a mistake in the implementation. This is discussed more in the upcoming section.

V. DISCUSSION

In this section, thorough discussions is carried out about the results, the implementation as well as limitations and possible improvements.

A. Results

The Reynolds number of the flows that the FNO was trained and evaluated on was $\mathrm{Re} \approx 588$, which is considered to be laminar flow. The FNO was initially trained on turbulent flow, but it turned out that it was difficult for it to learn that particular data set with a sample rate of only 1 Hz. The reason for this could be that the flow

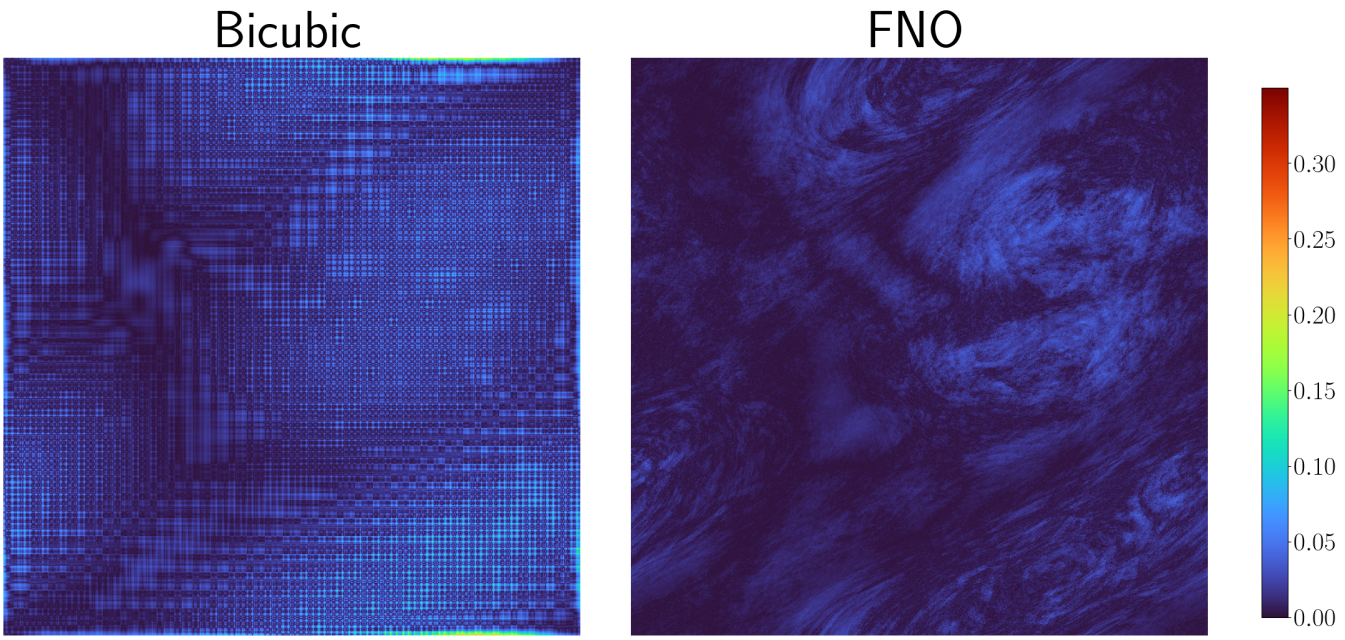

FIG. 4: The absolute error of a flow at the predicted frame $t = 40$ s between a bicubic interpolation and the true flow (left), as well as the absolute error between the FNO and the true flow (right). The error of the FNO is overall lower than the error of the bicubic interpolation, and shows a more defined resolution. Note that the largest errors (red) can be seen in some of the corners of the bicubic comparison.

changes too much between every frame, which makes it too difficult for the FNO to learn. The FNO could potentially learn turbulent flow better if it is instead trained on turbulent flows with a significantly faster sampling rate, thus better capturing smaller changes in the flow.

It also not surprising that the FNO has a harder time learning turbulent flow because the dynamics of Navier−Stokes equation are infamously hard to predict. R.G. Deissler calculated the Lyapunov exponent of a low Reynolds number (Re = 13.3) Navier−Stokes equation to $\lambda = 2.7$ [6]. And for higher Reynolds numbers, the dynamics only become harder to predict. It is therefore not surprising that the FNO had a more difficult time learning turbulent flows. Despite these difficulties, it is still an achievement that the FNO manages to predict resolution invariant flows of a Reynolds number of Re ≈ 588.

The FNO does not rely on a pixel-to-pixel mapping. Rather it learns a mapping between two function spaces, mapping the input data directly to the solution of the PDE on a fixed domain. This makes it much more versatile than traditional methods, since it can solve any new instance of the PDE without retraining, only requiring the domain to be constant. This highlights the advantages of using an FNO for applications where a high or variable resolution is desirable.

The FNO were supposed to be able to do zero-shot super resolution temporally as well as spatially. However, when the temporal resolution was analysed, the network generated a much higher error. For implementing the higher temporal resolution, a data set with a higher sampling rate than the network was trained on was used. The input was taken as the vorticity at the first 10 physical seconds (not the first 10 data points in the data set) to match the input data in the training set, while the output was expanded to match the larger frequency. An attempt using the first 10 data points in the higher resolution test set (corresponding to shorter physical time than the training set) was also done, but this generated even worse results. This error could be a consequence of how the time dimension was padded in the forward step of the network, but due to a lack of time, this was not investigated. Another possibly is problems with the linear transform L_2 (see Figure 1), regulating non-periodic boundary conditions. The resolution invariance in the temporal dimension could therefore not be confirmed.

When comparing the higher resolution prediction to upsampling a lower resolution prediction, it would be desirable to use other non-resolution invariant methods as a benchmark. Networks such as ResNet [7] or U-net [8] could have been analysed. Unfortunately, none of these models were available for the project, and due to time limitations, there was no possibility to create an implementation. Instead, the FNO was used as a proxy to determine the error caused by upsampling the result after the prediction was made.

Regarding the bicubic interpolation, it could be observed that the MAE was relatively constant at around 0.01 for all resolutions, which is around 100 % higher error compared to the FNO. For 1024×1024, the MAE increased to 0.03. This gives an indication of the ad-

vantages of using FNO with regards to spatial resolution invariance. It was somewhat surprising that the error for the bicubic was relatively constant for the lower resolutions. This could possibly be because the majority of the error was due to the prediction, rather than the upsampling, but it should be investigated further.

As stated above, the bicubic upsampling method used might not be the most accurate one for comparing the resolution invariance. When running high-resolution data through the FNO trained on 32×32, the MAE error could, as stated above, be due to both prediction and upsampling. The same sources of error applies when calculating the MAE for the bicubic upsampling. This time, however, the prediction is done first followed by an upsampling, rather than simultaneously. For a meaningful comparison between the two, it is therefore assumed that the prediction error and the upsampling error is independent of each other; the covariance between the two is zero.

B. Implementation of the FNO

There are multiple factors which had an impact on the results. The architecture of the model and the setting of different parameters is one such example. One important aspect is the number of frequency modes k kept after the Fourier transformation. Choosing fewer modes would likely result in too much information being lost, which could lead to underfitting in the training. On the other hand, choosing a larger number could lead to overfitting since the FNO could potentially learn the noise from the higher modes. Even though $k = 8$ seemed to provide satisfactory results, a more careful tuning of this parameter could have been performed. Parameter sweeps for the learning rate, the number of epochs and the number of hidden neurons could also have been performed with the purpose of optimizing the model.

For the learning rate, different techniques were tested to obtain an appropriate balance between exploration and exploitation. Initially, the cosine annealing method was applied, in which the learning rate, starting at its maximum value, is varied as a cosine function. In the final implementation, no restarts were used in the cosine annealing, meaning that only half a period was considered and effectively making it a regular learning rate decay. This provided slightly better results. The reason for this could be due to the early-stopping validation used causing the training to stop too early. Different combination of validation techniques and learning rate techniques could be investigated further.

The architecture of the model also has an impact on the performance and could also be analysed further. This includes the number of Fourier layers. Too many layers could potentially lead to overfitting, but could on the other hand result in better performance with respect to the resolution invariance. The Gaussian error linear unit (GELU) was consequently used as activation function. It is reasonable to use an activation function which captures the non-linearity of the dynamics. However, other alternatives could be considered, such as the hyperbolic tangent function.

Moreover, a different loss function could have been used in the training and evaluation. In this case, the mean absolute error was used as a metric, but mean squared error could for example have been used instead. Since MAE is a linear metric, it is more robust to outliers in the data, but the absolute value could potentially lead to problems when calculating the gradient.

PyTorch Lightning was used as the machine learning framework in this project. The disadvantage of using this framework is that the Fourier layers is not part it at the time of this examination. For a simpler and better optimized implementation, NVIDIA Modulus could have been used instead. However, this framework is in a late alpha or early beta stage. For the purpose of understanding the complete structure of the Fourier neural operator, it was rewarding to build it from the ground.

C. Other Improvements and Limitations

A potential improvement to the neural operator would be to turn it into a PINO by adding information about the PDE to the loss term. This would help the operator learn the underlying physics of the problem instead of just learning from data [2].

In recent years and months, several other variations of Fourier neural operators have been developed. This includes the U-FNO, where a U-Net is included in Fourier Layers, with higher accuracy and efficiency as a result. Other examples include the TFNO, which uses tensorization to reduce the number of parameters, or IFNO, which incrementally adds frequency modes during the training. The performance of these methods would be interesting to study [9], [10], [11].

The FNO implemented in this project only works on a pre-specified timescale. If instead it would be desirable to predict a flow for an indeterminate time into the future, an FNO using a recurrent structure could be used. This operator would only predict one fixed time step at a time and use the past couple of time steps as input. This means that the network can keep iterating as long as is desired, but the network loses its temporal resolution invariance.

The analysis was limited by the memory requirements of the input data. To predict a flow on a resolution of 1024×1024 for 40 time steps requires matrix multiplication with a floating point matrix of dimension $1024 \times 1024 \times 40 \times 20$. Anything larger than this would cause memory overflow, and was thus omitted from the analysis.

In new examinations, other fluid flows than Navier-Stokes could have been analysed as well. For example, Zongyi Li et al. [3] also performed simulations on Burgers' equation and Darcy flow. A Navier-Stokes equation

with different physical properties could also have been examined, such as a compressible flow in which $\nabla \cdot \vec{u} \neq 0$.

Moreover, physical parameters of the flow that the model was trained on could have been varied, to observe predictions of other nuances. Especially the viscosity could be interesting to vary, since this is directly related to the turbulence of the flow. A lower viscosity implies a lower resistance for the fluid to move, and thus a higher chance of turbulent motion. It would be interesting to train the FNO on flows with different viscosities, with the purpose of accurately predicting flows with different degrees of turbulence. Difficulty annealing could for example be applied, where the training data initially has higher viscosity, corresponding to a more laminar flow, which is then gradually decreased towards a more turbulent flow.

In order to perform a more comprehensive analysis, the trained model could also have been compared to more classical partial differential equation solvers, such as the FDM, FVM and FEM. A comparative analysis with regards to both accuracy and computational speed could then have been performed.

VI. CONCLUSIONS

This study confirms that the FNO architecture is not only able to learn and predict flows, it is also able to perform spatial zero-shot super resolution of flows described by the infamous Navier−Stokes equation. It turns out that the FNO is even better at upsampling spatial resolution than interpolating with bicubic interpolation. A large advantage that comes with this is that it can be trained on low-resolution data, and still be used to accurately predict flows with much higher resolution. This makes it computationally cheaper to train than methods that use pixel-to-pixel mapping, especially for applications where a high resolution is desirable.

Unfortunately, the FNO could not perform temporal zero-shot super resolution. This would be a primary topic to investigate for further improvements. Furthermore, it would be interesting to implement the PDE in the loss term to make the FNO learn the PDE itself and thus turning it into a PINO. Lastly, varying the viscosity to have the model learn more turbulent flows would be another further potential investigation.

Simulations of Navier−Stokes equations can provide great insight into the complex dynamics of fluids, with several applications in science and engineering. Over the years, different techniques have been applied, ranging from classical numerical methods to machine learning algorithms. Given vast applicability of fluid dynamics, the development of more efficient and accurate partial differential equation solvers, such as the Fourier neural operator analysed in this study, will always be of great interest.

VII. CONTRIBUTIONS

All group members were working together on writing and understanding the implementation of the FNO. All members were part of writing the report.

Specifically, G. Burman wrote part of the result, discussion, the theory on Fourier neural operators and was part of generating data for the analysis. K. Lundgren was the initiator of the project, took the main responsibility for large parts of the code and implementation, and wrote parts of the discussion. He also set up a GPU-accelerated server with unlimited compute time on private hardware to host the project, enabling all group members to connect remotely. E. Norlin was part of generating results, plots and animations, wrote the background, and parts of implementation, results, discussion and conclusions. M. Wiklund wrote the theory on Fluid dynamics, the abstract, parts of the method (data set), results, discussion and conclusion, and analysed data.

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[2] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, "Physics-informed neural operator for learning partial differential equations," *arXiv preprint arXiv:2111.03794*, 2021.

[3] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

[4] Y. He, "Two-level method based on finite element and crank-nicolson extrapolation for the time-dependent navier-stokes equations," *SIAM Journal on Numerical Analysis*, vol. 41, no. 4, pp. 1263–1285, 2003.

[5] B. Fornberg, *A practical guide to pseudospectral methods*. Cambridge university press, 1998.

[6] R. G. Deissler, "Is Navier–Stokes turbulence chaotic?," *The Physics of Fluids*, vol. 29, pp. 1453–1457, 05 1986.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[8] X.-X. Yin, L. Sun, Y. Fu, R. Lu, and Y. Zhang, "U-net-based medical image segmentation," *Journal of Healthcare Engineering*, vol. 2022, 2022.

[9] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, "U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow," *Advances in Water Resources*, vol. 163, p. 104180, 2022.

[10] J. Kossaifi, N. B. Kovachki, K. Azizzadenesheli, and A. Anandkumar, "Multi-grid tensorized fourier neural operator for high resolution pdes," *ICLR*, 2023.

[11] J. Zhao, R. J. George, Y. Zhang, Z. Li, and A. Anandkumar, "Incremental fourier neural operator," *arXiv preprint arXiv:2211.15188*, 2022.

Appendix A: Hyperparameters

Hyperparameters used for training were a learning rate of 0.001, 500 number of training epochs with early-stopping validation and 12 epochs as patience. The modes kept after Fourier transformation were the lowest 8, and the mean absolute error (MAE) was used as the loss function.

Appendix B: Bicubic interpolation

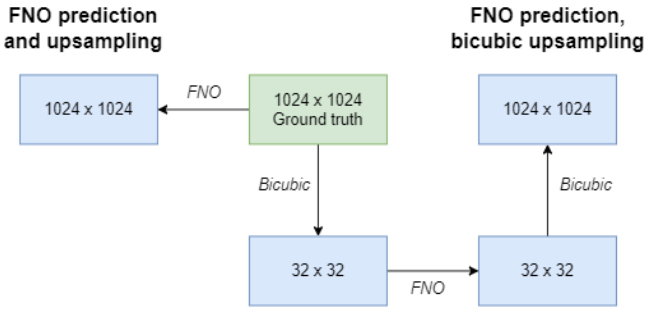

FIG. 6: The bicubic interpolation procedure used when comparing to the spatial resolution invariance of the FNO. Note that 1024×10241 is used an example here. This was performed for 64×64, 128×128, 256×256 and 512×512 as well. A high-resolution flow was downsampled using bicubic interpolation, predicted with the FNO and then upampled using bicubic interpolation again.

Figure 6 displays the procedure used when comparing the spatial resolution invariance of the FNO to bicubic interpolation. A high-resolution flow was downsampled, predicted with the FNO and then upampled. From this comparison, it could be seen how well the FNO performs simultaneous prediction and upsampling, compared to a prediction, followed by an bicubic upsampling.

Appendix C: Results

TABLE III: The mean absolute error obtained from evaluating on different data sets. A spatial resolution varying from 32×32 to 1024×10241 were tested. All data sets were tested for a model trained on a 32×32 data set with temporal resolution 1 s. The error is given as the mean absolute error with 95 % confidence intervals as error margin. This error was calculated for both a prediction with the FNO on the data set, as well as making a prediction with the FNO on a downsampled (to 32×32) version of the data set and then upsampling the prediction using bicubic interpolation.

Spatial res.	FNO MAE	Bicubic MAE
32	0.0057 ± 0.0015	–
64	0.0056 ± 0.0016	0.0116 ± 0.0035
128	0.0054 ± 0.0021	0.0102 ± 0.0053
256	0.0055 ± 0.0033	0.0102 ± 0.0083
512	0.0059 ± 0.0036	0.0104 ± 0.0085
1024	0.0087 ± 0.0054	0.0335 ± 0.0198

Table III displays the the numerical results from the test of the spatial resolution invariance. The MAE for both the FNO and the bicubic upsampling are shown, with an error margin given by 95 % confidence intervals. There results corresponds to the ones seen in Figure 5.