AMATH 590 - Final Paper: Flow Reconstruction, Time Stepping, and Sub-Grid Stress studies on a Turbulent Fluid

Deepak Venkatanarasimhan(1940141)

June 4, 2021

Abstract

Turbulent fluid velocity varies significantly and irregularly in both position and time. It is a rich field of study in physics, given its complexity, ubiquity, and practical applications. The variation in velocity and pressure fields is captured by the Navier Stokes equation, which is a second-order system of differential equations in 3 spatial coordinates and time. The data used comes from a numerical solution to the Navier Stokes. This paper uses neural networks as a function approximator to model three functional maps: i) Ability to reconstruct a fluid's high-dimensional state from a coarser grid representation, ii) Model the time-evolution of the fluid's state in time, iii) Model the relationship between coarse and fine-scale velocities using the Large Eddy Simulation(LES) framework. These experiments are done primarily to make it computationally cheaper to work with fluids with a large state. This is possible for fluids that have a low-dimensional representation that captures most of the energy of the system, on which they evolve [1].

1 Introduction and Overview

Partial Differential Equations (PDEs) play a central role in understanding fundamental physical phenomena. In the field of fluid mechanics, one studies Navier Stokes. The key variables in the Navier Stokes for incompressible fluids are the velocity vector field $U = (u, v, w)$ along the three coordinate directions and the accompanying pressure p. The equation is constructed using the principles of mass and momentum conversation. Fluids are also characterized by a non-dimensionless quantity called Reynold's number, which is the ratio of fluid velocity to kinematic viscosity. As Re is increased, a transition occurs from laminar to turbulent flows [2]. At low Reynolds numbers, viscous forces are dominant, and we have smooth and laminar flows. At high Reynolds numbers, the fluid velocity creates flow instabilities such as eddies and is characterized as turbulent flow. Turbulence can transport and mix a fluid effectively. Given the ubiquity of turbulent flows, understanding them is imperative to being able to control them. For instance, a commonly encountered issue due to turbulence is a high drag on aircraft wings. An associated control example is designing an actuator that reduces drag.

The required dataset is downloaded from Johns Hopkins Turbulence Database. The data is forced isotropic turbulence on a 1024^3 spatial grid with periodic boundary conditions. The time dimension is discretized by 1024 too [3]. Isotropic turbulence is the property by which a field is statistically invariant under rotations or reflections of the coordinate axis. This happens if the domain is not near the boundary of a flow or any other singularities. As is evident, one expects to have computing problems (RAM, download speed, etc.) downloading or working off the entire file at full resolution. Hence, functional maps that can reconstruct the state from a low-dimensional/ coarse grid are useful. LES too avoids the computational effort devoted to fine-grid simulations(99%, per [4]) by modeling the dissipation range and using solved (via DNS) velocities at crude resolution. The dissipation range is the lowest in the energy cascade that is dominated by viscous action. It receives kinetic energy from turbulence at large scales of motion. Small-scale motions at sufficiently large Reynold's numbers are isotropic, as they are far removed from the chaotic eddies at large scales. Hence, statistics of small-scale motions are, in a sense, universal. Smagorinsky is a well-known model that links the filtered or resolved rate of strain (computed at low resolution) with the residual stress (that requires input from the fine scales) [5]. This helps close the LES model.

In this paper, I perform three experiments on the isotropic turbulence dataset. The first task is flow reconstruction in line with the concepts discussed in [6]. I try to reconstruct a $512 * 512$ image(of a velocity field) from a $16 * 16$ low-definition version. The results are in line with the paper and great in-sample. The second task is constructing a flow map to evolve a reasonable resolution image of a velocity field across time. I use both multiscale flow maps and single-step flow maps. The key motivation for multiscale flow maps is that dynamics can be different at coarse and fine scales, for instance, the continuum model and molecular model in fluids [4]. The multiscale flow maps prevent error accumulation from small-step models by resetting the prediction over longer time periods. The multiscale work done is in line with [7], and the results, appear promising. Finally, I use matrix operators for first derivative of a field on a plane as specified in [1] to estimate the residual stress tensor τ_{ij} using the static Smagorinsky model. I compare this with deep-learning versions as specified in [8]. I obtain one iteration of results that is inline with my expectations. For all the above examples, I use the feed-forward network architecture. For the residual stress estimation, I use a CNN that is said to train faster and perform better on 2D image data. The CNN prediction results however don't confirm this theory, possibly due to lack of training data. I perform my analysis using Google's Colab. All code is available in Appendix A.

The next section discusses Neural Networks theory for both the Flow reconstruction and Flow maps case, and the LES model in greater detail.

2 Theoretical Background

Deep Learning is better known for its applications across Computer Vision, and Natural Language Processing [9]. Applications beyond these domains stem from Deep Learning's ability to approximate functions via a layered transformation process. For instance, it is useful in physics-based systems such as Fluid Mechanics [10]. It has been used to perform data transformations to a low-dimensional manifold such as POD [11], and estimate closure models in a data-driven manner without using equations [10]. Closure models typically discussed are either for mean-flow equations such as RANS [12], or to close LES equations as discussed in [8] and replicated in this paper. One key advantage of using Neural Networks as opposed to other models is that it can replicate the POD step in one of its layers without having to do it explicitly ([6],[11]). The most straightforward architecture often used is a feed-forward network. An example of a feed-forward network used in the flow reconstruction project is represented by Equation 1.

$$xhd = \sigma(W_3(R(W_2R(W_1(xld)))) \tag{1}$$

Here, xld is the low-dimensional fluid velocity image of size $16 * 16$ and xhd is the high-dimensional image of size $512 * 512$. By feeding multiple pairs of the low and high-dimensional images, the neural network learns a map from xld to xhd, i.e., a map from a vector of size $16 * 16$ to $512 * 512$. This can finally be used to reconstruct xhd from xld. The key hyperparameters in setting up a neural network are the number of layers between the input and output, the size of each of these layers, and the activation function. For instance, in the example in Equation 1, there are three layers between the input and the output. Also, the first two layers have a RELU activation function depicted by R, and the final layer has a linear activation function depicted by σ. The size of the layers decides the number of weights required to make each transformation happen, and the activation function adds a further function to be executed to compute the vector input to the next level. To understand hyperparameter tuning, activation functions, and measurement better, please read [13]. To walk through the setup, please refer to my code in Appendix A.

$$x(t + \Delta t) = \sigma(W_4(R(W_3R(W_2\Sigma(W_1(x(t))))))) \tag{2}$$

For time-stepping, we have a similar setup as in Equation 1. The inputs and outputs are separated by a timestep as shown in Equation 2. Here, the size of the input and output vectors is the same, and the neural network is trained over multiple timesteps. For the multiscale version, we train different neural networks for fine time steps and coarse time steps. While setting up predictions, we fill the matrix in the order of the timestep size, starting with the coarse timesteps. This ensures that error accumulation in the fine timesteps is corrected or reset by the coarse timesteps. For more details on implementation, please either refer to the Algorithm section or my code in Appendix A.

For the LES case, we start with the filtered Navier Stokes equation on a 2D grid as shown in Equation 3, 4. Anything with an overline, such as $\overline{u_i}$, $\overline{u_j}$, $\overline{p}$ are filtered velocity and pressure quantities, which implies that they are measured on a coarser grid. In the below equations index i indicates the i^{th} component of the quantity, for instance u_i is the i^{th} component of velocity.

$$\frac{\partial \overline{u_i}}{\partial x_i} = 0 \tag{3}$$

$$\frac{\partial \overline{u_i}}{\partial t} + \frac{\partial \overline{u_i u_j}}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial}{\partial x_j}\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right) \tag{4}$$

The filtering on LES is different from the mean process on RANS. Please see below, the overall velocity u is given by Equation 5 [4]. Here, $\overline{u}$ is measured on a coarse grid and is a random variable (not mean). For instance, it is measured for each point on a $32 * 32$ grid in my code. u' is the residual or sub-grid scale component. The filtered velocity contains most of the kinetic energy and represents the motion of large eddies, while u' represents the dissipation process.

$$u = \overline{u} + u' \tag{5}$$

Also, as is clear from Equation 4, $\overline{u_i u_j}$ is on the fine grid and not known due to truncation of small eddies by the spatial filtering operation [8]. This basically needs to be resolved in terms of $\overline{u_i}$ and $\overline{u_j}$. This is given by Equation 6, where τ_{ij} are the sub-grid or residual stresses. The true value of this can be calculated from DNS simulation of u on a $256 * 256$ grid and $\overline{u}$ on a coarse $32 * 32$ grid.

$$\tau_{ij} = \overline{u_i u_j} - \overline{u_i u_j} \tag{6}$$

The true value is compared against three deep learning estimations of τ_{ij} and one computed using the static smagorinsky model given by Equation 7. The filtered rate of strain S_{ij} is given by Equation 8, 9. The trace of τ is given by $\tau_{11} + \tau_{22}$.

$$\tau_{ij} = -2(C_s\Delta)^2 \|S\| S_{ij} + \frac{1}{2}(\tau_{11} + \tau_{22}) \tag{7}$$

$$S_{ij} = \frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i} \tag{8}$$

$$\|S\| = \sqrt{2 S_{ij} S_{ij}} \tag{9}$$

The dynamic Smagorinsky model([14]) is better at finding an optimal value of C_s for a position in time and space. However, for this analysis I stick with a single value of $C_s \approx .17$, as the dynamic value in the reference paper varies between .16 and .18 [8]. For Δ, I use the value of dx for the coarse grid of resolution $32 * 32$. For deep learning estimations of τ_{ij}, I build two feed-forward networks. For the first one termed $M1$, I use a parsimonious input of just u, v. For the second network, I use the velocities, first and second spatial derivatives $(u_x, v_x, u_y, v_y, u_{xx}, v_{xx}, u_{yy}, v_{yy})$ of the velocities as input, and hence it more closely models Equation 4. To compute the derivatives I create a sparse matrix derivative operator as detailed in [1]. To understand the same, please refer my code at Appendix A. I also train a Convolutional Neural Net on the same input variables, but performance as shared in the results section is not up to the mark.

Outside of the work by [8], there have been other research papers that deal with the use of NNs to parameterize LES closure terms in isotropic flows, channel flows, and other configurations ([15],[16],[17],[18],[19],[20]). Their results make for interesting study.

Reconstruction and flow-map work explore the transformation of a fluid with a high-dimensional state to a lower-dimensional manifold on which one can study evolution. In the LES work, one designs an algorithm that solves the fluid dynamics equations on a coarser grid and replaces the components affected by the resolution loss with better-performing learned alternatives. The learned alternatives (deep learning parameterization) typically have better performance than modeled alternatives (smagorinsky). In the next section, I walk through all implementation steps in the analysis.

3 Algorithm and Implementation

As discussed in prior sections, in this paper, I discuss the three study types listed below. A good way to read the below description of steps for each analysis is to look at it along with the code. My code is in Appendix A.

1. Flow field Reconstruction based on [6]. Implementation details are at Algorithm 1
2. Flow maps that capture time evolution of a fluid based on [7]. Please see Algorithm 2
3. Predict sub-grid stresses from filtered velocity fields that is based on [8]. Please see Algorithm 3

Algorithm 1: Fluid State Reconstruction from low-dimensional snapshot

Download datasets of both required resolutions - high-dimensional (512*512) and low-dimensional (16*16)
Setup Input and Output matrices of w (Velocity along the z-direction) along the x-y plane
We end up with 512 total pairs of w (Velocity along the z-direction) along the x-y plane, at different values of z
Split the dataset into training and test. Further split training into Development and Validation
Demean the input and output datasets
Setup a feed-forward configuration with 2 hidden layers as shown in the code
Train a neural network with the Development and Validation samples
Publish both in-sample and out-of-sample results

Algorithm 2: Flow Maps - Single and Multiscale

Download appropriate resolution data (256*256) at multiple points in time, i.e., 64 snapshots in time
Pickup the variable u (velocity along the x-direction) along the y-z plane. Study its time evolution
Setup input and output matrices, where the output is one timestep forward
Train a feed-forward neural network after setting up development, validation, and testing samples
Predict the trajectory for a given starting point within the sample and outside the sample
For multiscale we have 128 snapshots in time and a lower resolution of (128*128)
Train three neural networks for fine, medium, and coarse scale
Predict the trajectory first for the coarse time step
Now make predictions for the medium timestep where there are no overlaps
Finally fill up missing points in time with the fine timestep predictor
Compare quality of performance between single scale and multiscale predictors

Algorithm 3: Estimate sub-grid stresses from filtered velocity field

Load a fine grid dataset (256*256) and a coarse grid dataset (32*32)
Compute true subgrid stress τ_{ij}
Setup derivative operators for periodic boundary condition vectors
Compute smagorinsky estimation of subgrid stress using filtered stresses
Compute derivative features of filtered velocity vectors over the coarse grid
Setup three neural networks: i) without derivative features, ii) with derivative features, iii) CNN with derivative features. The output is the subgrid stress tensor
Compare probability distribution of estimated τ from smagorinsky and deep learning vs. the true stresses. Publish the results.

One important choice to be made before implementing any of the above experiments is the resolution of input data and variables required. The choice may need a tradeoff between accuracy and available computational resources because very high-resolution data may have high query running times. Another key task on which I have not spent much time is hyperparameters. Large networks may capture functional maps better but may require a lot of resources to run. For instance, in the flow-map case, because the dimensionality of input and output is large (full-scale), the number of neurons in hidden layers can scale up running time a lot. CNNs are said to have an advantage in terms of running time and number of parameters for data of the same dimensions. In the next section, I focus on results from each of the above experiments.

4 Computational Results

This section discusses results from i) Flow reconstruction, ii) Flow maps, and iii) Sub-grid stress computation. I download both a low-definition and a high-definition version of fluid velocity in the x-y plane for the flow reconstruction experiment. Please see Figure 1 for an example snapshot.

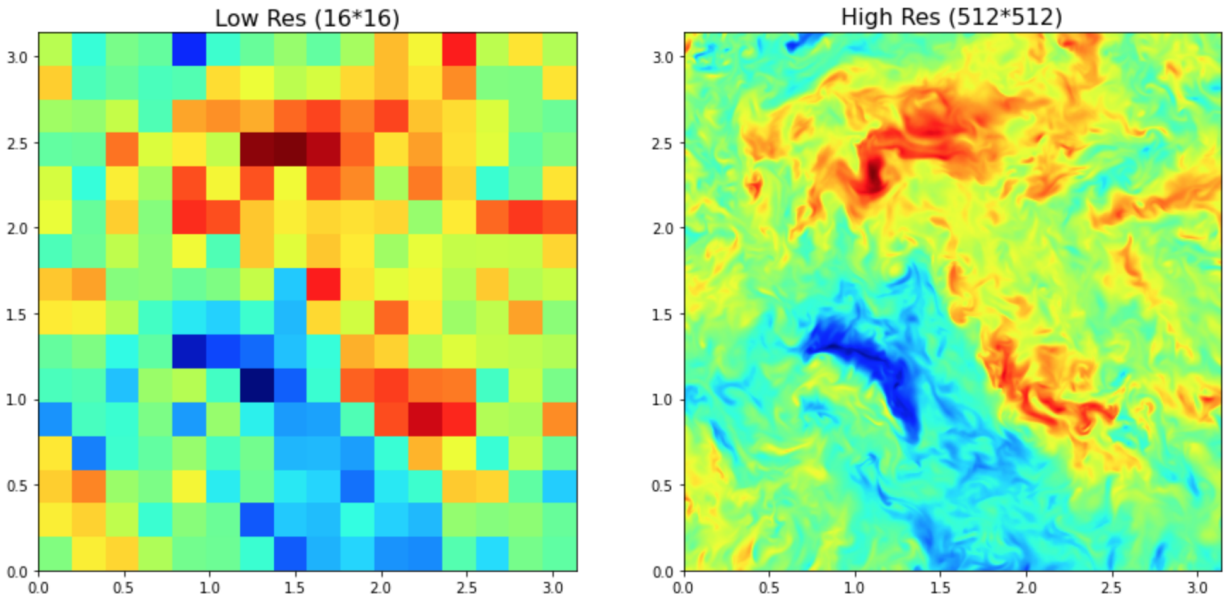

Figure 1: Low-dimensional and High-dimensional version of w(velocity in the z-direction) in the x-y plane

I train a feed-forward neural network that maps from the low-dimensional version to the high-dimensional version, over 410 samples, i.e., different z at the same point in time. Please see Figures 2 and 3 for an example of in-sample and out-of-sample reconstruction. The out-of-sample reconstruction is from the test set. To get a sense of validation loss, please refer directly to the code in Appendix A. I also compare the Mean-absolute Error(MAE) of the in-sample and out-of-sample reconstruction in Table 1. As is clear from the image, in-sample performance is better. The takeaway is one can reconstruct the state well if one has trained the NN on enough snapshots of data, and the training data covers the state we are testing.

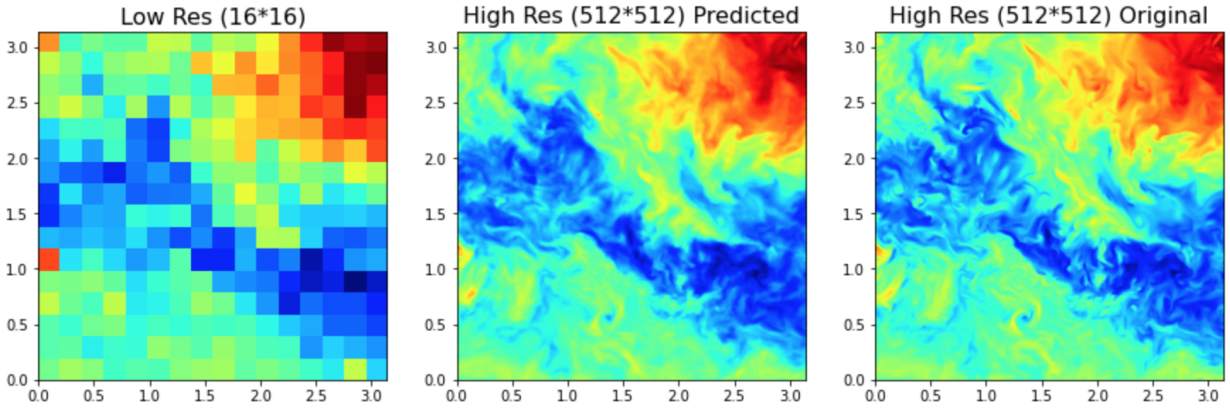

Figure 2: In-sample reconstruction

For the flow-map case, we construct the input and output data such that they are separated by one timestep. Please see Figure 4 for the single-step flow map results, and Figure 5 for the multi-scale flow map predicted

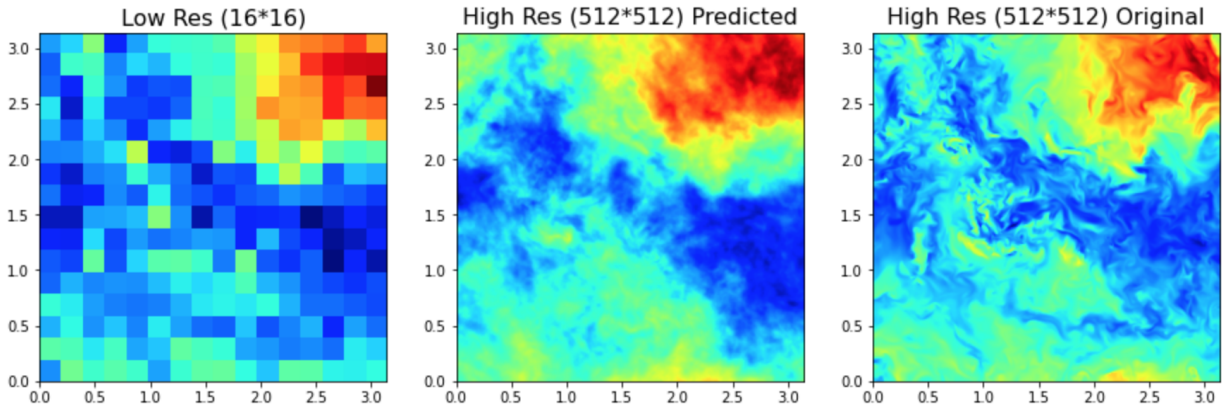

Figure 3: Out-of-sample reconstruction

Table 1: Flow Reconstruction: MAE

Metric	Mean MAE across Predictions	SD of MAE across Predictions
In Sample MAE	0.0308	0.0055
Out of Sample MAE	0.2903	0.0341

time-evolution of u(velocity in x-direction) in the y-z plane. Provided there are different dynamics at different scales, multi-step should be able to predict time evolution better. Comparison of performance between the single-step and multi-step predictors is provided in Table 2. Both the performance readings compare predicted time-evolution (with the same starting point as the original) to the original. From the results, it appears that multiscale outperforms the single-scale predictor. Please note that for Figures 3 and 4 published below, the time scales are different. So $t = 21$ on Figure 3 is not the same time-instant as $t = 21$ on Figure 4.

Table 2: Single-scale and Multi-scale Flow Maps: MAE

Metric	Mean MAE across Predictions	SD of MAE across Predictions
Single-scale MAE	0.1644	0.1466
Multi-scale MAE	0.0355	0.0302

Table 3: τ_{ij} Predictions vs Original: MAE

Metric	Mean MAE across Predictions	SD of MAE across Predictions
τ_{11} Model 1 MAE	0.0380	0.0182
τ_{11} Model 2 MAE	0.0363	0.0154
τ_{11} Model 3 (CNN) MAE	0.2111	0.0182
τ_{11} Smagorinsky MAE	0.6721	0.0008
τ_{12} Model 1 MAE	0.0377	0.0180
τ_{12} Model 2 MAE	0.0357	0.0156
τ_{12} Model 3 (CNN) MAE	0.1930	0.0167
τ_{12} Smagorinsky MAE	0.6058	0.0002
τ_{22} Model 1 MAE	0.0511	0.0225
τ_{22} Model 2 MAE	0.0478	0.0225
τ_{22} Model 3 (CNN) MAE	0.2688	0.0354
τ_{22} Smagorinsky MAE	0.8144	0.0016

Finally, for the sub-grid stresses of a 2D turbulent fluid, please see Figure 6, 7 for a probability distribution of $\tau_{11}, \tau_{12}, \tau_{22}$ from Samgorinsky, Model 1, Model 2, and Model 3 predictions vis-a-vis the true stresses. Model 1 uses only the velocity fields u and v as inputs, while Model 2 uses the velocity fields, and their gradients and second derivatives (u_x, u_y, v_x, v_y, u_{xx},u_{yy}, v_{xx}, v_{yy}) too as inputs. Performance comparisons are available in Table 3.

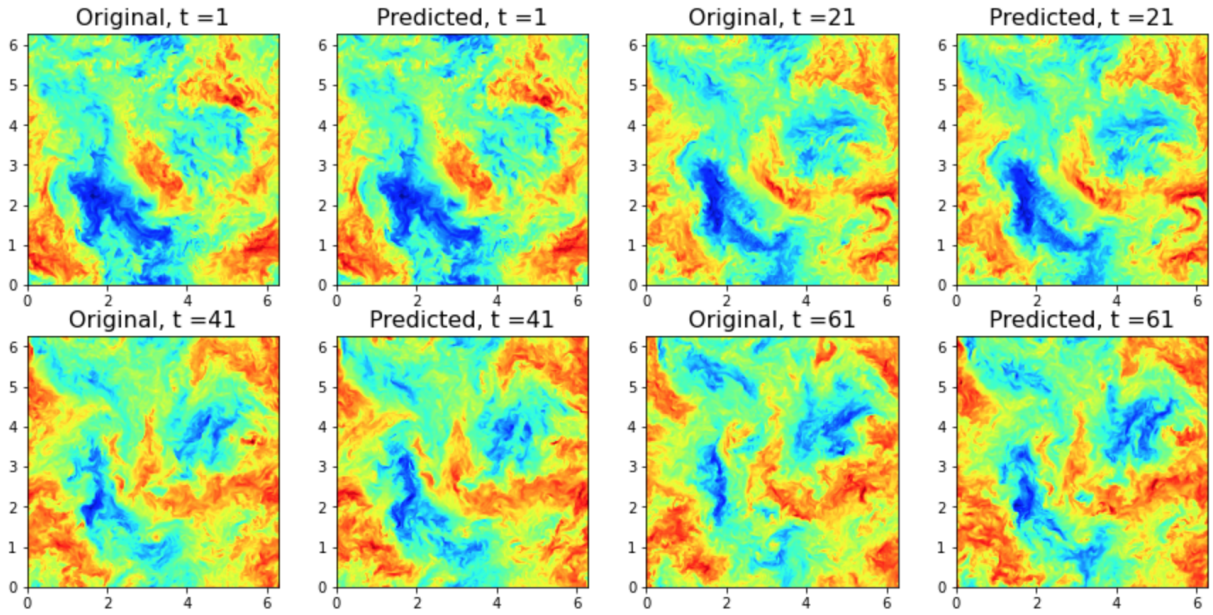

Figure 4: Single-step flow map predicted time evolution

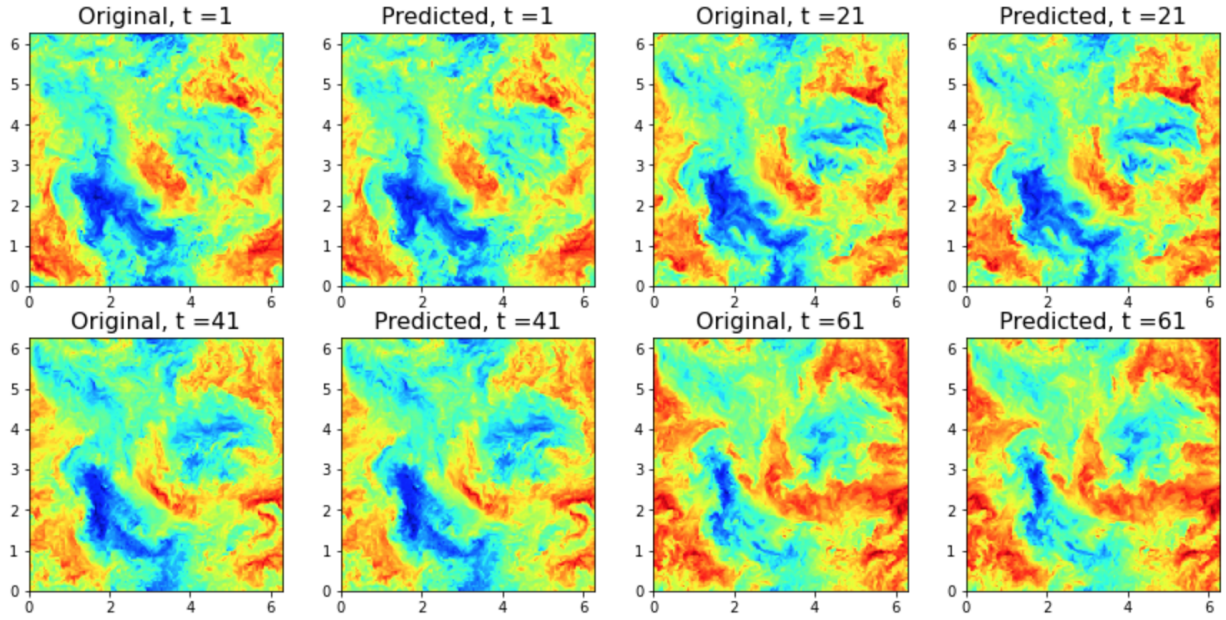

Figure 5: Multi-scale map predicted time evolution (time scales are different to the single-step case: For instance, snapshot at $t = 21$ on the multi-step case is not the same time instant on the single-step case)

The performance of both Model 2 and Model 1 are good and comparable, with Model 2 marginally better. It appears that the neural network can simulate the additional derivative variables used in Model 2 without explicitly requiring them. I wasn't able to get the CNN model to perform, possibly because of insufficient data. The smagorinsky estimation's comparison to true sub-grid stresses is expected to be poor, per the paper [8]. More study is needed to better Models 2 and 3. For all of the performance measures published above, I take the Mean and SD of the MAE across multiple testing snapshots/ images. MAE for a snapshot or an image is NN prediction vs. original or truth. MAE is the mean of absolute error between predicted

and original across pixels of a snapshot.

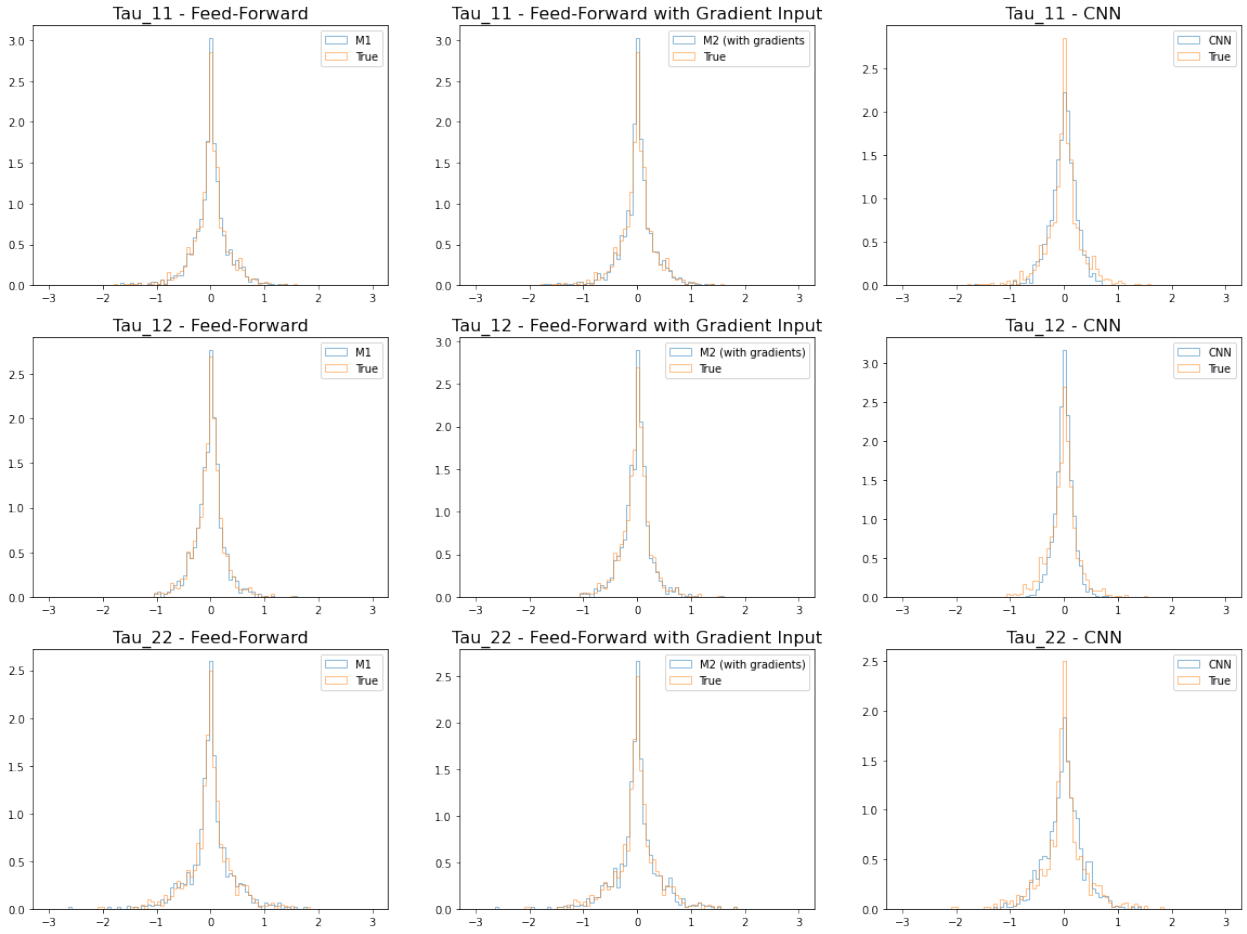

Figure 6: PDFs of τ predictions (multiple NN models) compared to ground truth

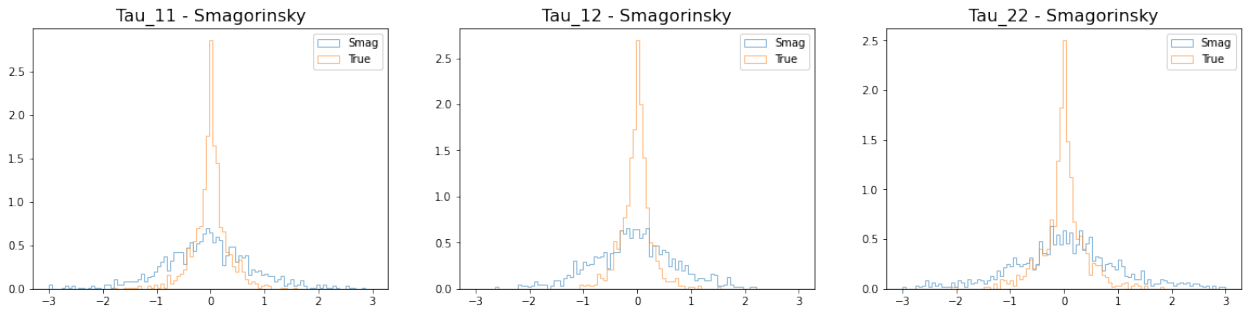

Figure 7: PDFs of τ predictions (Smagorinsky) compared to ground truth

5 Discussion

Classical methods in computational fluid dynamics use numerical solutions (DNS) to Navier Stokes and physics models such as Smagorinsky to develop a complete picture of the fluid's flow at integral, inertia, and sub-grid scales. In this paper, we look at how data-driven techniques leveraging neural networks as a

function approximator can be used to study fluid-structure in space and time in an equation-free manner and physics-based deep learning parameterization as in the case of sub-grid stress modeling. Deep Learning, if used right, can improve the computational performance of solutions and enable the use of sparse sensing through its super-resolution abilities.

In this paper, we look at a relatively simple/ isotropic case of turbulence. One can look at varied other practical flows such as flow in a cavity, near a wall et al., and other complex boundary conditions. One can also explore deep learning networks' ability to determine the structure of the fluid and use the same to design optimal actuation methods that lead to control outcomes such as reducing drag.

Specifically, on the results - they are along expected lines for the flow reconstruction case and the flow map case. The flow reconstruction neural net has good within-sample performance but doesn't extrapolate very well. The multi-scale flow map does better in controlling error compared to the single-scale flow map. In predicting sub-grid residual stresses, we get good results for a first iteration, i.e., the feed-forward networks do a good job predicting sub-grid stresses. The reference paper [8] gets better results for CNN and the feed-forward network that uses velocity spatial derivative features. CNNs are said to perform better than feed-forward networks when the data is in the form of snapshots, and are better at super-resolution tasks, turbulence closure modeling, et al. While I couldn't replicate this possibly for not having enough training data - my work covers the broad theoretical underpinnings of the paper. More study and work is needed to improve models 2 and 3 that capture the relationship between the sub-grid stress tensor and velocity vectors.

References

[1] Nathan Kutz. *Data-Driven Modeling & Scientific Computation*. Oxford University Press, 2013.

[2] Peter J Olver. *Introduction to Partial Differential Equations*. Springer, 2014.

[3] M. Van et. al. *The Johns Hopkins Turbulence Databases (JHTDB), Readme File*. 2008.

[4] Stephen B Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[5] J Smagorinsky. *General Circulation Experiments with the Primitive Equations*. 1963.

[6] Steve Brunton et. al. *Shallow Neural Networks for Fluid Flow Reconstruction with Limited Sensors*. 2020.

[7] Nathan Kutz et. al. *Hierarchical Deep Learning of Multiscale Differential Equation Time-Steppers*. 2020.

[8] Suraj Parmar et. al. *Apriori analysis on Deep Learning of subgrid-scale parameterizations for Kraichnan Turbulence*. 2019.

[9] Yoshua Bengio et. al. *Deep Learning*. 2015.

[10] Steve Brunton et. al. *Machine Learning for Fluid Mechanics*. 2020.

[11] Petros Koumoutsakos et. al. *Neural Network Modeling for Near Wall Turbulent Flow*. 2002.

[12] Karthik Duraisamy et. al. *Turbulence Modeling in the Age of Data*. 2018.

[13] Yoshua Bengio et. al. *Deep Learning*. MIT Press, 2016.

[14] Massimo Germano et. al. *A dynamic subgrid-scale eddy viscosity model*. 1991.

[15] Andrea Beck et. al. *Deep Neural Networks for Data-Driven LES Closure Models*. 2019.

[16] R Maulik et. al. *Sub-grid modelling for two-dimensional turbulence using neural networks*. 2018.

[17] Jonathan F. MacArt et. al. *Embedded training of neural-network sub-grid-scale turbulence models*. 2021.

[18] F. Sarghini et. al. *Neural networks based subgrid scale modeling in large eddy simulations*. 2001.

[19] Anikesh Pal. *Deep Learning parameterization of subgrid scales in wall-bounded turbulent flows*. 2019.

[20] Chenyue Xie et. al. *Artificial neural network mixed model for large eddy simulation of compressible isotropic turbulence*. 2019.

Appendix A Link to Code

I performed the data pull operations directly on the scientific server provided by Johns Hopkins, as they are closer in terms of network hops to where the data is stored. The analysis was performed on Google Colab. At times I scaled down the resolution of data I was working with to manage file size and training time of neural networks (RAM limitations). Parallelizing may help us perform more compute-heavy tasks.

Table 4: Data pull and Analysis Code

Description	**Link to code**
Download Data Example	get data.ipynb
Download Data - Big Cutout	get data.ipynb
Read .h5 files	Readh5.ipynb
Explore Turbulence Data	ExploreTurbulenceData.ipynb
Plot Velocity Data	PlotVel.ipynb
Flow Reconstruction	Reconstruction.ipynb
Flow Maps	FlowMaps.ipynb
Flow Maps Multi-scale	MultiscaleFlowMaps.ipynb
Sub-grid scale Research	Sub grid Scale Analysis.ipynb