Research Project
Turbulence Modeling Using Deep Learning

Student: Amir Israel
Advisers: Dr. Michael Karp, Dr. Yuval Levy

02/08/22

Contents

1 Deep Learning Basics **1**

1.1 Fully Connected Neural Network . . . 2
1.1.1 Perceptron . . . 2
1.1.2 Activation Function . . . 3
1.1.3 Backpropagation . . . 3
1.2 Convolutional neural network (CNN) . . . 3
1.2.1 Time Convolutional network (TCN) . . . 4
1.3 Recurrent neural network (RNN) . . . 5
1.3.1 LSTM (long short term memory) . . . 5
1.4 Deep autoencoders . . . 5
1.5 Generative adversarial network (GAN) . . . 6
1.6 Gene expression programming (GEP) . . . 6

2 Turbulence modeling basics **8**

2.1 Direct numerical simulation (DNS) . . . 8
2.1.1 Length And Time Scales . . . 8
2.1.2 Comparison To LES . . . 9
2.1.3 Fractional Step Method . . . 9
2.2 Large Eddy Simulation (LES) . . . 10
2.2.1 Sub-Grid Models . . . 10
2.2.2 WRLES (Wall Resolved Large Eddy Simulation) . . . 11
2.2.3 WMLES (Wall Modeled Large Eddy Simulation) . . . 11
2.2.3.1 Wall Stress Modeling . . . 11
2.2.3.2 Equilibrium Wall Models . . . 12
2.2.3.3 Non-Equilibrium Wall Models . . . 12
2.2.3.4 Separation . . . 13
2.2.3.5 The Off-Wall Matching Location . . . 13
2.2.3.6 Virtual Wall Boundary Conditions . . . 14
2.3 Reynolds-Averaged Navier-Stokes (RANS) . . . 14
2.3.1 Linear Eddy Viscosity Models (LEVM) . . . 14
2.3.1.1 Algebraic models . . . 15
2.3.1.2 The $k-\epsilon$ Model . . . 15
2.3.1.3 The $k-\omega$ Model . . . 16
2.3.1.4 The $k-\omega$ SST (Shear Stress Transport) Model . . . 16
2.3.2 Nonlinear Eddy Viscosity Models (NEVM) . . . 16
2.3.3 Reynolds Stress Model (RSM) . . . 17

3 Latest Advancements In The Field Of Deep Learning Application In Turbulence Modeling **18**
3.1 Reynolds Averaged Turbulence Modeling Using Deep Neural Networks With Embedded Invariance By Ling et al. (2016) . . . 18
3.1.1 Description . . . 18
3.1.2 Results . . . 19
3.2 Machine Learning-Accelerated Fluid Dynamics By Kochkov et al. (2021) . . . 20
3.2.1 Description . . . 20
3.2.2 Results . . . 21
3.2.2.1 Accelerating DNS . . . 21
3.2.2.2 Accelerating LES . . . 22
3.3 RANS Turbulence Model Development Using CFD-Driven Machine Learning By Zhao et al. (2020) . . . 22
3.3.1 Description . . . 22
3.3.1.1 Frozen Training . . . 23
3.3.1.2 CFD Driven Training . . . 23
3.3.2 Results . . . 24
3.4 Deep learning to replace, improve, or aid CFD analysis in built environment applications: A review By Calzolari and Liu (2021) . . . 26
3.4.1 Constant Tuning Methods . . . 26
3.4.2 Turbulence Models Enhancements . . . 26

4 Future work **27**

Abstract

In the past few years the field of Machine Learning (ML) in general, and Deep Learning (DL) in particular has been getting a lot of attention, both because the continued growth of computational power, and the generation of large data sets.

Fluid mechanics enjoys large amount of data sets from experiments and simulations, *e.g* "Johns Hopkins Turbulence Databases" and many more. Moreover, the fact that fluid flow often contain reoccurring patterns that can be studied and predicted makes fluid mechanics an excellent fit for the use of ML.

Various Computational Fluid Dynamics (CFD) methods had been developed to simulate the behavior of complex flow.

However, accounting for all of the spatial and temporal scales in turbulent flows is extremely challenging and computationally expensive. Therefore, DL methods have been developed to improve the prediction capability and computational efficiency of turbulent flow simulations.

The type of problems that are encountered in fluid mechanics greatly differ from the common DL problems. The accuracy of turbulent flow numerical solutions is essential for satisfactory prediction. This is because of the chaotic and unsteady nature of turbulent flows.

This paper describes various methods used to overcome the above mentioned challenges and return meaningful results, one of the main features that is shared by these methods is the use of prior physical knowledge to improve and stabilize the DL algorithms.

The first chapter describes the basics of deep learning. The basics of turbulence modeling are in chapter 2.

Chapter 3 discusses some application of deep learning in turbulence modeling, it is demonstrated that DL can improve both the computational cost and accuracy for variety of CFD simulations.

Chapter 1

Deep Learning Basics

DL is a subset of ML that is based on artificial neural networks (NNs). A deep learning model is built by NNs with many layers.
NNs are built using a collection of connected nodes, that pass signals to one another (where the "signal" is usually a real number). The connections between the nodes are called edges, and they usually have a weight that adjusts with the learning process (and thus increase or decrease the strength of the signal at a connection - see Figure 1.1) by using nonlinear optimization methods, such as Backpropagation.

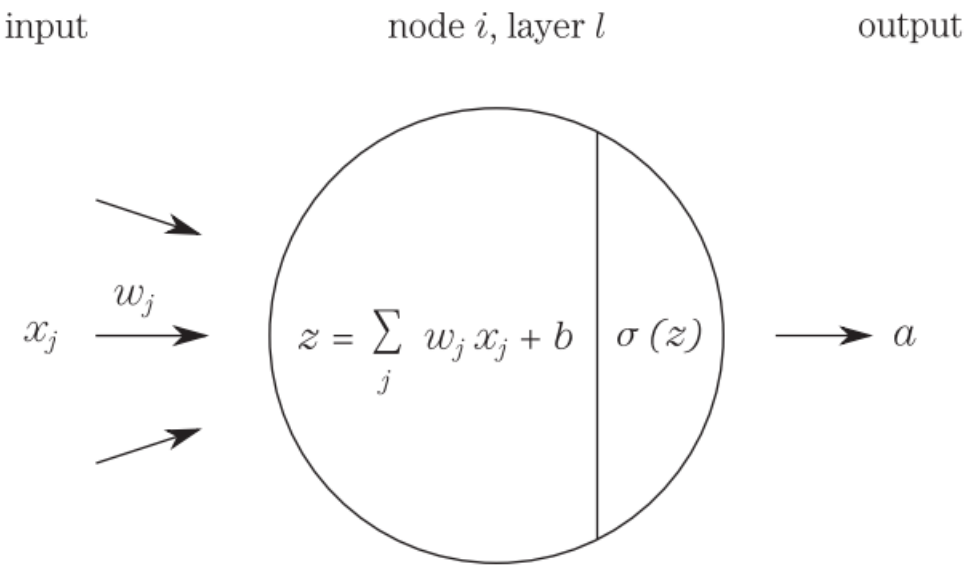

Figure 1.1: Basic structure of a node

These NNs can be combined into different structures that reflect knowledge about the problem and data.
ML in general is divided into 3 groups - supervised, semi-supervised or unsupervised.
Supervised learning is a type of learning where the input is mapped to an output based on examples of input-output pairs, where unsupervised learning is a learning type where patterns are obtained from untagged data.
The data that is used to train any ML model is divided into 3 parts:

- Training data - this data is used for the learning process, the model adjusts itself with it.

- Validation data - this data is used in the end of the training process to evaluate the model and choose proper hyperparameters.
- Test data - this data is used to test the model after it is built, to make sure that the model can resolve properly unseen data.

DL can be deployed using various architectures as mentioned in the following sections.

1.1 Fully Connected Neural Network

Fully connected neural network (or Multi Layer Perceptron - MLP) is the simplest class of NNs. This architecture is characterized by all the nodes in a specific layer connecting to the nodes in the next layer, the only different between Deep MLP and MLP is the amount of layers used (see Figure 1.2).

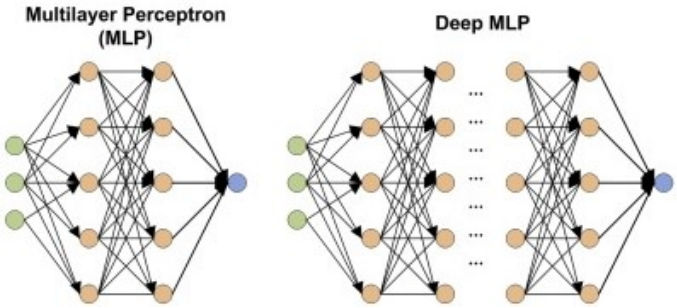

Figure 1.2: Schematics of MLP and deep MLP nodes architecture

1.1.1 Perceptron

The perceptron algorithm is a binary classifier (a function which can decide to which class a specific input belongs). It is very important since it constitutes the basis for NNs.
The perceptron algorithm can only divide properly linearly separable patterns (patterns that can be divided by an hyperplane, see Figure 1.3), It works by finding a value of w (the weights, and the normal to the hyperplane) so that:

$$f(x) = \begin{cases} 1 & if\ w \cdot x + b > 0 \\ 0 & otherwise \end{cases}$$

The value of w is found through a learning process.

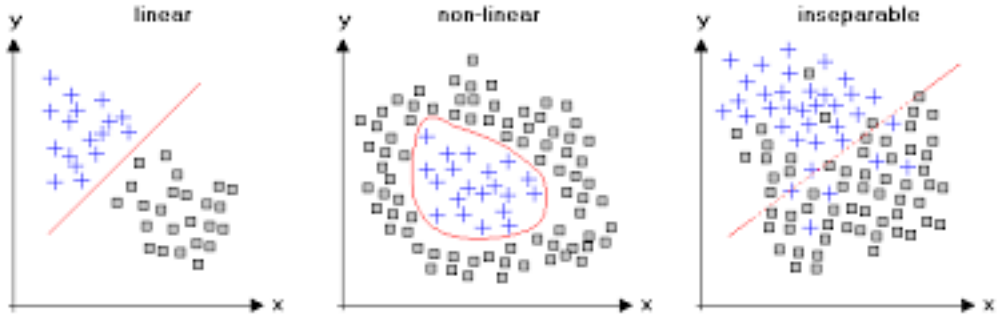

Figure 1.3: Visualization of linearly separable, non-linearly separable data and inseparable data

1.1.2 Activation Function

In MLP, each node output passes through an activation function, which is usually non-linear. The reasoning for it is the ability to represent non-linear patterns in the data, without them the entire MLP can be reduced to just input-output model. Examples of some activation functions are:

$$y\left(v_i\right)=\tanh\left(v_i\right),\ \ y\left(v_i\right)=\left(1+e^{-v_i}\right)^{-1},\ \ y\left(v_i\right)=\max\left(0,x\right)$$

Where the latter is also known as Rectified Linear Unit (ReLU) and is usually used.
Thus, the full equation of MLP is:

$$y_L\left(W_L y_{L-1}\left(W_{L-1}y_{L-2}\left(...W_2 y_1\left(W_1 x+b_1\right)+b_2\right)...\right)\right)$$

1.1.3 Backpropagation

Backpropagation is an algorithm used to compute the gradient of the loss function with respect to the weights. The algorithm works with the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to the first, and so, the change in each weight is:

$$\Delta w_{ij}\left(n\right)=-\eta\frac{\partial\varepsilon\left(n\right)}{\partial v_j\left(n\right)}y_i\left(n\right)$$

Where y_i is the output of the previous node, $0<\eta<1$ is the learning rate, and:

$$\varepsilon\left(n\right)=\frac{1}{2}\sum_j e_j^2\left(n\right)$$

Where e is a single node error and defined as:

$$e_j\left(n\right)=d_j\left(n\right)-y_j\left(n\right)$$

Where d_j is the target value.

1.2 Convolutional neural network (CNN)

Convolutional neural network (CNN) is a class of NNs that is used usually to analyze image data due to its architecture: It takes into account the spatial proximity of data points, equivalent to translation, and able to handle large amounts of data easily.

Unlike MLP, CNN weights are filters the image passes through (see Figure 1.4), known as kernels.

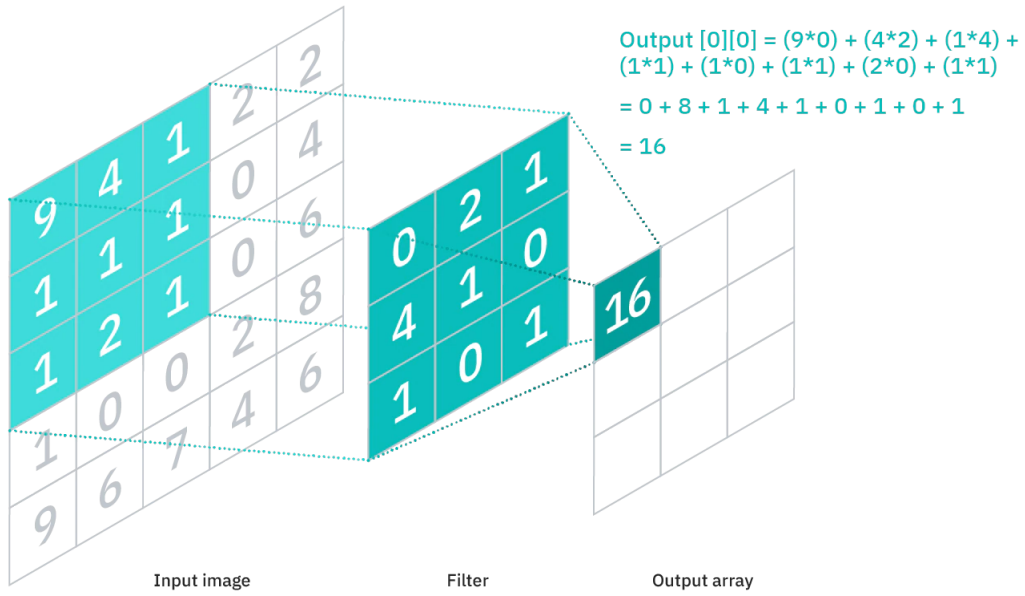

Figure 1.4: A visualization of an image passed through a kernel (filter)

The amount of kernels, size of each kernel, stride (distance between application of the kernels), of each kernel dilation (spacing between the points on which the kernel applies) are hyperparameters that change between NNs.

1.2.1 Time Convolutional network (TCN)

A specific kind of CNN is TCN, this architecture is similar to a regular CNN but with a one dimension (usually with changing dilation) convolution for the time dimension, thus, as shown in Figure 1.5 this architecture is able to model sequential data, and consider the data development and location in time (more on that, in the next section).

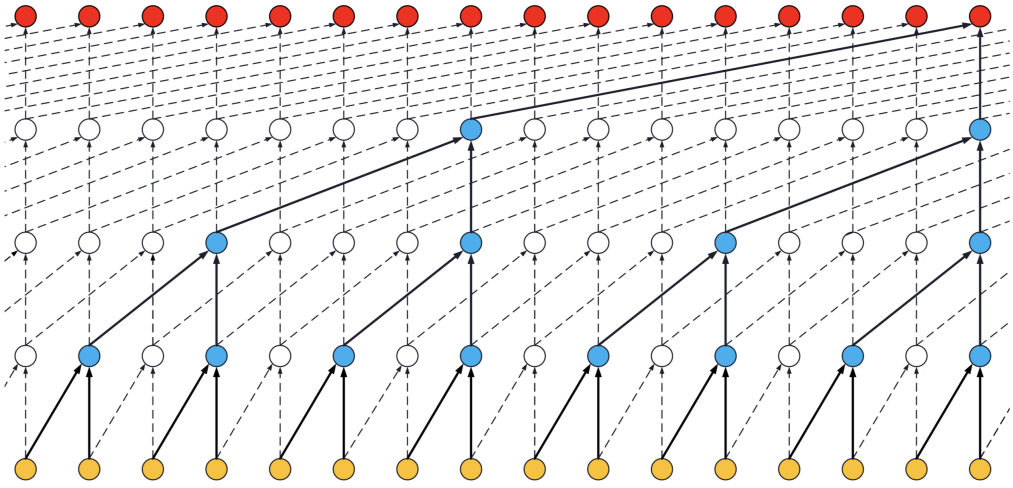

Figure 1.5: Schematic of TCN - each of the yellow nodes represents a sample of data in time (in case of a video, for example, each node would be a 2d image)

1.3 Recurrent neural network (RNN)

Recurrent neural network (RNN) is a class of NNs that operate on sequences of data, and their weights are obtained by Backpropagation through time. Since their architecture takes into account the inherent order of the data they are attractive for fluid mechanics.
The RNN model works with cells, each cell uses both the input in a specific time x_t and the output of the previous point in time y_{t-1}, the problem with this simple cell structure is that the weight for the past data is the same for each cell, causing the past data to diminish in importance quickly or explode, to deal with this problem, LSTM (long short term memory) was created.

1.3.1 LSTM (long short term memory)

LSTM is similar to a classic RNN but with slightly different cell type, as shown in Figure 1.6 the LSTM cell type has an added "forget gate" beside the classic input and output gate, and it helps to regulate the flow of information into and out of the cell.

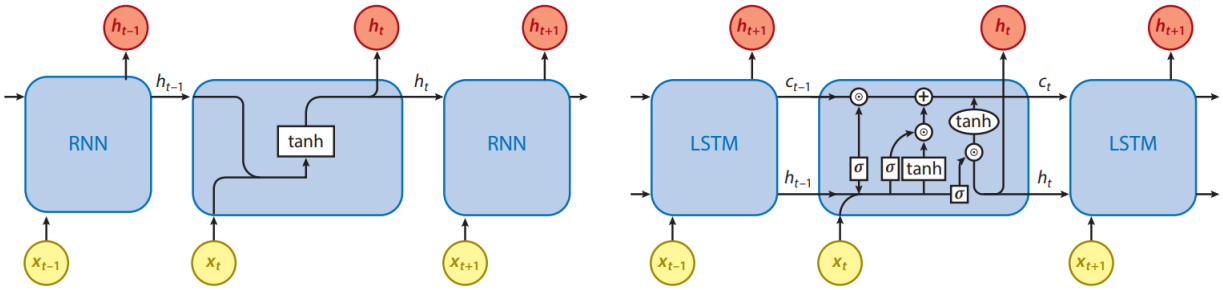

Figure 1.6: Schematic of Classic RNN (in the left) in comparison to LSTM (in the right), the "forget" gate marked as c.

1.4 Deep autoencoders

Identifying lower dimensional representations for high dimensional data from experiments and large scale simulations has always been a cornerstone of flow modeling. Principal Components Analysis (PCA) is a type of analysis that revolves around preforming change of basis on the data using the eigenvectors of the data's covariance matrix.
PCA can be represented as a 1 layer Encoder Decoder:

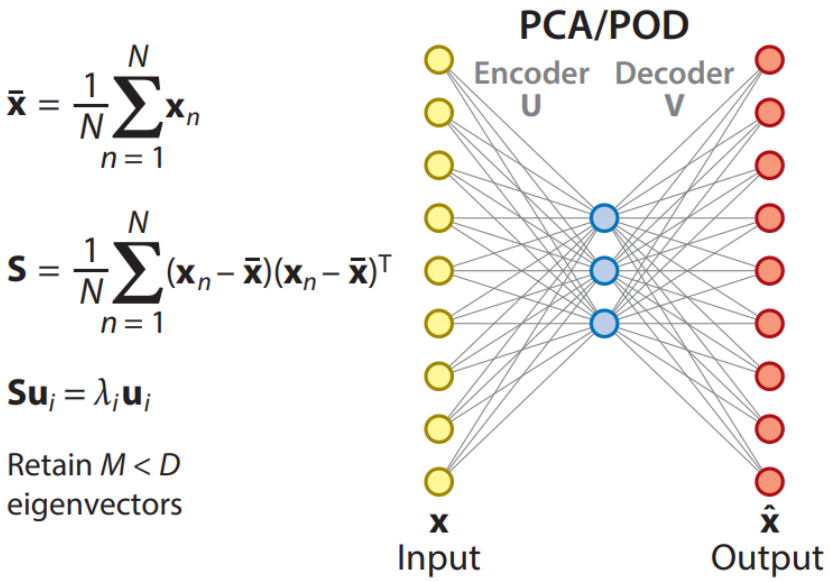

Figure 1.7: Schematic of PCA/POD as a one layer autoencoders

Using Deep autoencoders instead of the classic PCA is essential for finding non-linear patterns in the data since the added layers gives the model better predicting power.
Therefore, Deep autoencoders can be used instead of PCA in the field of fluid mechanics by representing non-linear patterns in the flow.

1.5 Generative adversarial network (GAN)

GANs are learning algorithms that result in a generative model, which is a model that can generate data by using and mimicking the data it was trained on. the way it works is by having one NN that produces candidate data examples and another that evaluates the results.

1.6 Gene expression programming (GEP)

Gene expression programming (GEP) is an evolutionary algorithm that generates models. These models are created with tree structures that can adapt.
The models evolve each iteration by using the best performing models from the last iteration and creating similar models for the current iteration.
The tree model is represented in code as a string, for example, the string $Q * b * * + baQba$ will result in the following tree structure:

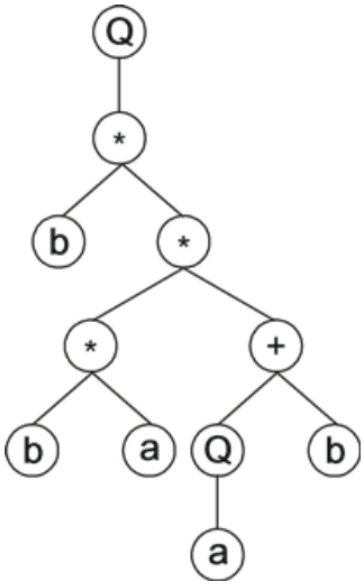

Figure 1.8: The tree structure matching the string $Q * b * * + baQba$

And models the expression:

$$\sqrt{b\left(ba\left(\sqrt{a}+b\right)\right)}$$

The selection of the population of models for the next iteration is done by Roulette-wheel selection with the fitness function of each model, meaning, the probability of model i to be chosen for a spot in the population of models in the next iteration is:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Where f_i is the fitness of model i.
The model evolution is activated by changing one or more letters in the string that creates the model, therefore inserting small changes, also each node has a weight (just as we have seen so far) these weights have small changes inserted too.

Chapter 2

Turbulence modeling basics

CFD in very diverse and can be used for many different flows, from simple ones like potential and laminar flows up to multi-phase flows, in this part we'll be focusing on single-phase turbulent flows, for this kind of flows 3 types of models were developed, DNS, LES and RANS:

2.1 Direct numerical simulation (DNS)

The most computationally expensive model type, this model type captures all eddies in the flow and thus provides the most accurate prediction possible for the flow. The lengthscale of the smallest eddies (known as Kolmogorov microscales) are found at $Re \sim 1$, since the equilibrium between viscous forces and advection forces allows dissipation of the eddies to heat.

2.1.1 Length And Time Scales

The Kolmogorov microscales are given by:

$$\eta = \left(\frac{\nu^3}{\varepsilon}\right)^{\frac{1}{4}}$$

Where ν is the kinematic viscosity and ε is the kinetic energy dissipation.
To satisfy these resolution requirements, the number of points N along a given mesh direction with increments h, must be:

$$Nh > L$$

Where:

$$h \leq \eta$$

From dimensional analysis:

$$\varepsilon \approx \frac{u'^3}{L}$$

Where u' is the root mean square of the velocity.
From all those results we obtain:

$$N^3 \geq Re^{9/4}$$

Where:

$$Re = \frac{u'L}{\nu}$$

Since the very large memory necessary for DNS, only explicit methods can be used for the integration of the solution in time, therefore, in general, the CFL should be kept below 1:

$$C = \frac{u'\Delta t}{h} < 1$$

The total time of the simulation is usually proportional to the turbulence time scale τ:

$$\tau = \frac{L}{u'}$$

Combining those we obtain:

$$\frac{L}{\eta} \sim Re^{3/4}$$

2.1.2 Comparison To LES

To demonstrate how computationally expensive DNS is a summary of papers comparing the cost of DNS to some of the other methods (such as WRLES and WMLES) is presented in Figure 2.1:

	Grid-point exponent a, where $N \sim Re_{L_x}^a$			Time-step exponent b, where $N_t \sim Re_{L_x}^b$			Overall cost exponent c, where $NN_t \sim Re_{L_x}^c$		
	DNS	WRLES	WMLES	DNS	WRLES	WMLES	DNS	WRLES	WMLES
Chapman[1]	...	1.8	0.4	...	...	...	...	2.4*	0.53*
Choi and moin[2]	2.64	1.86	1.0	...	...	...	3.52*	2.48*	1.33*
Present	2.05	1.86	1.0	6/7	6/7	1/7	2.91	2.72	1.14

Figure 2.1: The computational cost of DNS, WRLES, WMLES according to: Chapman (1979) (top row) Choi and Moin (2012) (middle row) Yang and Griffin (2021) (bottom row)

2.1.3 Fractional Step Method

For incompressible flow DNS, the fractional step method is usually chosen, in this method every time step we make sure to fix the pressure field to embed incompressibility to the solution, for second order explicit Adams-Bashforth scheme for the convective terms and second order implicit Crank-Nicholson for the viscous terms we get:

$$\frac{u_i^{n+0.5} - u_i^n}{\Delta t} = \frac{1}{2}\left(3H_i^n - H_i^{n-1}\right) + \frac{1}{2}\frac{1}{Re}\left(\frac{\delta^2}{\delta x_1^2} + \frac{\delta^2}{\delta x_2^2} + \frac{\delta^2}{\delta x_3^2}\right)\left(u_i^{n+0.5} + u_i^n\right)$$

$$\frac{u_i^{n+1} - u_i^{n+0.5}}{\Delta t} = -G\left(\phi^{n+1}\right)$$

With:

$$D\left(u_i^{n+1}\right) = 0$$

Where $H_i = -\frac{\delta}{\delta x_j}u_i u_j$, G is discrete gradient operator, D is discrete divergence operator, and $p = \phi + \frac{\Delta t}{2Re}\nabla^2\phi$.

2.2 Large Eddy Simulation (LES)

A computationally expensive model type, but not the most expensive. The goal of this model is to capture the largest eddies of the flow (most meaningful for almost every engineering purpose), without the smallest eddies, which are modeled by a subgrid model.
The reasoning behind it is that the smallest eddies demand a very fine grid and very small time steps to be captured, which makes the simulation prohibitively expensive.
In general, LES models attempt to capture eddies that contain 80% of TKE, of course in order to capture more eddies - a finer grid is needed, as shown in Figure 2.2. but in different locations in our grid different grid density is needed, close to the wall for example - we expect smaller eddies, so the eddies sizes are needed to be predicted.

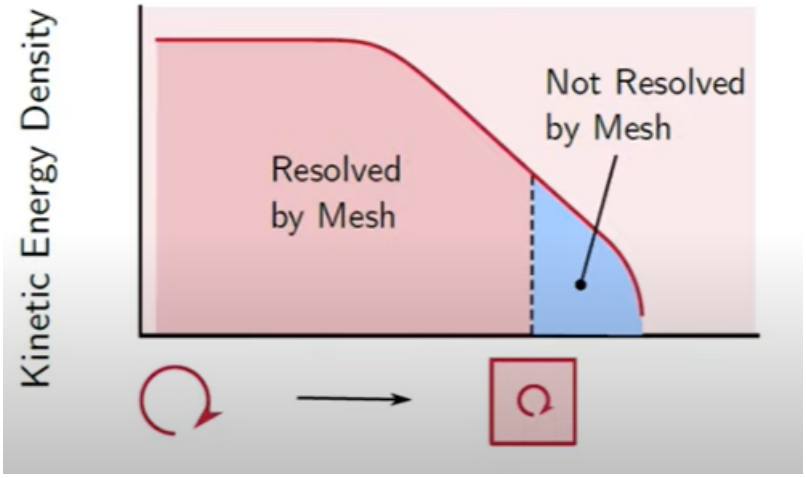

Figure 2.2: Graph showing the expected TKE resolved by the LES mesh (the area in red) and the TKE not resolved by the LES mesh (the area in blue). The x axis represents the eddy size, all the TKE not resolved by the mesh in contained by the smallest eddies.

The average eddy size, or mixing length l_m, has been mentioned last chapter, and is a great predictor for the needed grid density, usually we define $\Delta = l_m/5$ (since l_m is the average eddy size and not the grid size needed to capture 80% of the TKE), however, this is just a guess and nothing guarantees that it is indeed correct.
Unlike RANS, in LES it is possible to measure the disturbances of the flow and therefore measuring the instantaneous Reynolds stresses and there is no need to model them.
On the other hand, we do need to model the effect of the smallest eddies on the flow. without doing so, not only that a meaningful amount of TKE wont be taken into account, but the eddies wont be able to dissipate since eddies usually dissipate in the smallest scales where the molecular viscosity dominates, and these scales are not represented in our grid, for that goal the sub-grid models were created.

2.2.1 Sub-Grid Models

We can use the TKE equations to find k_{sgs} (the TKE that our grid doesn't capture):

$$\frac{\partial(\rho k_{sgs})}{\partial t} + \nabla(\rho U k_{sgs}) = \nabla[\rho D_k \nabla k] - C_\epsilon \frac{\rho k_{sgs}^{3/2}}{\Delta} + \rho G_{sgs} - \frac{2}{3}\rho k_{sgs}\nabla U$$

We can also use the average size of the eddies we don't capture (l_{sgs}), usually it is defined as:

$$l_{sgs} = C_s \left(\Delta\right)^{1/3}$$

Where C_s is defined by the model (usually around $C_s = 0.1$), this model doesn't take into account the wall effects, a fixed model might be:

$$l_{sgs} = \min\left(\kappa y, C_s \left(\Delta\right)^{1/3}\right)$$

It's important that the sub-grid model will alter the viscosity since, as mentioned before, one of the main roles the sub-grid model has is to dissipate eddies that would be too big to dissipate otherwise, for that reason τ_{sgs} is added to the momentum equations:

$$\frac{\partial\left(\rho U_i\right)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho U_i U_j\right) = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\tau_{ij} + \tau_{sgs}\right)$$

τ_{sgs} is usually found using the Boussinesq eddy viscosity assumption:

$$\tau_{sgs} = 2\rho\nu_{sgs}S^{\star}_{ij} - \frac{2}{3}\rho k_{sgs}\delta_{ij}$$

ν_{sgs} is different for every model, but depends on the grid density - since the finer the grid the less viscosity needs to be added to dissipate the smallest eddies the grid can capture, therefore:

$$\nu_{sgs} = f\left(\Delta\right)$$

One of the challenges LES faces is to simulate the flow close to the wall, that's because the velocity gradients there are high and the eddies are effected by them and are smaller there, which makes the simulation very hard and computationally expensive to model.

2.2.2 WRLES (Wall Resolved Large Eddy Simulation)

This group contains models that have very fine grid close to the wall and because of it - able to capture enough eddies very close to the wall, the only problem with that method is that it is almost as computationally expensive as DNS.

2.2.3 WMLES (Wall Modeled Large Eddy Simulation)

This group contains models that doesn't have a very fine grid close to the wall, and instead attempts to model the flow close to the wall.

2.2.3.1 Wall Stress Modeling

The "law of the wall" states that the velocity profile close to the wall will be:

$$u^{+}\left(y^{+}\right) = \frac{1}{\kappa}\log\left(Ey^{+}\right) + C^{+}$$

Where C^{+} is a parameter that changes according to the wall roughness ($C^{+} \approx 5$) for smooth walls, $\kappa \approx 0.41$ is the von karman constant and $E \approx 9.8$.
This assumption doesn't take into account the linear sub-layer or the buffer layer. there are different methods that take this into account either by by writing one continuous function to contain them both (Spalding), or by mixing the linear and logarithmic profiles, these models manage to do that by altering the viscosity:

$$\nu_w = \nu + \nu_t$$

$$\nu_t = \begin{cases} 0 & y^+ < 11.25 \\ \nu\left(\frac{y^+}{\frac{1}{\kappa}\log(Ey^+)} - 1\right) & y^+ > 11.25 \end{cases}$$

A more flexible model can be used by using the thin boundary-layer equation (TBLE), for the incompressible case:

$$\frac{\partial \tilde{u}_i}{\partial t} + \frac{\partial \tilde{u}_i \tilde{u}_j}{\partial x_j} + \frac{1}{\rho}\frac{\partial \tilde{p}}{\partial x_i} = \frac{\partial}{\partial y}\left[\left(\nu + \tilde{\nu}_t\right)\frac{\partial \tilde{u}_i}{\partial y}\right],\ i = 1...3$$

$$\tilde{u}_i\left(y_w\right) = 0,\ \tilde{u}_i\left(y^*\right) = U\left(y^*\right)$$

Where y^* is the height where we match the wall model to the LES solution.

2.2.3.2 Equilibrium Wall Models

The equilibrium wall models are models where the left-hand side of the TBLE equations is neglected, the result is a steady RANS solution for the velocity profile in the inner layer which can be thought of as an ensemble average of the many eddies the grid is incapable to capture. This simplification will lead to constant total stress (viscous and Reynolds) in the inner layer which does seem to match our existing knowledge about the inner layer of a boundary layer, to assess the accuracy of this approximation one can analyze it using modified Von Karman integral analysis (set $\frac{1}{\rho}\frac{dp}{dx}$ as $U_\infty \frac{dU_\infty}{dx}$):

$$\frac{\tau_w}{\rho} \equiv u_\tau^2 = \frac{\tau\left(y^*\right)}{\rho} + \frac{\partial}{\partial t}\int_0^{y^*}\left(U_\infty - \tilde{u}\right)dy + \frac{dU_\infty}{dx}\int_0^{y^*}\left(U_\infty - \tilde{u}\right)dy$$

$$+\frac{\partial}{\partial x}\int_0^{y^*}\left[\tilde{u}\left(U_\infty - \tilde{u}\right)\right]dy + \left(U_\infty - \tilde{u}\right)v\left(y^*\right)$$

The limit of $y^* \to \delta\left(x\right)$ in this equation will cause the first and last terms in the right-hand size to vanish, the 3 surviving terms will be functions of the displacement thickness (δ^*), the momentum thickness (θ) and the last term can be non-dimensionalized and bounded:

$$\frac{\rho U_\infty}{\tau_w}\frac{\partial U_\infty}{\partial x}\int_0^{y^*}\left(1 - \frac{\tilde{u}}{U_\infty}\right)dy \le \frac{\rho U_\infty \delta^*}{\tau_w}\frac{dU_\infty}{dx}\frac{y^*}{\delta^*} = \Pi\frac{y^*}{\delta^*}$$

Where Π is the Clauser parameter. Therefore by controlling $\frac{y^*}{\delta^*}$ (which depends on our grid) the effects of the pressure gradient can be made negligible, this implies that the LES grid should approach closer to the wall at locations with higher pressure gradients.

2.2.3.3 Non-Equilibrium Wall Models

The non-equilibrium wall models are models where some terms in the left-hand side of the TBLE equations are not neglected.

If the advective terms are included: Some of the Reynolds stress will be resolved, therefore, $\tilde{\nu}_t$ must be reduced to avoid skin friction overprediction.
Wang and Moin (2002) argued that by matching the Reynolds stress at $y = y^*$, the unresolved Reynolds stress is matched, this can be done by modifying the von Karman constant:

$$\kappa_{wm} = \frac{\nu_{SGS}(y^*)}{y^{*+}\left[1 - exp\left(-y^{*+}/A^+\right)\right]^2}$$

Where ν_{SGS} is the LES eddy viscosity. Park and Moin (2014) found that the optimal value of the von karman scaling coefficient is flow and mesh dependent and suggested:

$$\mu_{t,wm}(y) = \mu_{t,neq}(y) + \frac{\rho \bar{R}_{ij} \bar{S}^d_{ij}}{2\bar{S}^d_{ij}\bar{S}^d_{ij}}(y)$$

Where $\mu_{t,neq} = \rho(\kappa y)^2 |S| D(y)$, $|S| = \sqrt{2S_{ij}S_{ij}}$ and $\bar{R}_{ij}$ is the resolved Reynolds stress in the inner layer solution.

If the pressure gradient is included without including convective or unsteady terms: A normalized wall distance is defined $\hat{y} = y u_{\tau p}/\nu$, where:

$$u_{\tau p} = \sqrt{u_\tau^2 + u_p^2},\ u_p \left| \frac{\nu}{\rho} \frac{\partial \langle p \rangle}{\partial x} \right|^{\frac{1}{3}}$$

While this inclusion of the pressure gradient does improve the results relative to the equilibrium wall models, Hickel et al. (2013) suggested that inclusion of the gradient effects without the convective and unsteady terms is inconsistent with the momentum equations outside of the boundary layer.

It is also possible to account for the nonequilibrium effects through the presumed shape of the near wall velocity profile: Yang et al. (2015) suggested an approximation of the velocity profile that consists of 11 undetermined coefficients which can be solved from the boundary conditions, continuity, and the von Karman momentum integral equation.
This method assumes linear velocity profile in the linear sub-layer and the following form in the overlap region:

$$\frac{\langle u \rangle}{u_\tau} = B + \frac{1}{\kappa} \log\left(\frac{y}{y^*}\right) + A\frac{y}{y^*}$$

2.2.3.4 Separation

Separation points are not treated well by most of the methods presented above, the problem arise from the fact that while the separation can be far from the wall $u_\tau \to 0$ (and therefore $\mu_t \to 0$) the turbulent viscosity is very meaningful.
This has resulted in ad hoc treatment of separation where a no-slip boundary condition is applied where the separation is known or expected to be.

2.2.3.5 The Off-Wall Matching Location

The location of The off-wall matching location (y^*) is where the LES provides the outer boundary condition for the wall model, since all models rely on the constant stress assumption it is very important that y^* won't be too large, on the other hand, using the first grid cell velocity (or

temperature) is known to give shifted results, that problem is referred to as "the log-layer mismatch"
Wu and Meyers (2013) observed that the largest errors occur at the first cell, probably since the stress carrying eddies that scale with the distance from the wall must be underresolved, and found the leading error discrete approximation for a logarithmic velocity profile:

$$\frac{\delta \langle u \rangle}{\delta y} = \frac{d \langle u \rangle}{dy} \left(1 + \frac{c_n n!}{d^n} \right)$$

Where $y = d\Delta y$ so the first grid cell is $d = \frac{1}{2}$ or $d = 1$ depending on the centering of the solution variables, Kawai and Larsson (2012) suggested that the matching location won't be placed at the first cell and instead be placed at the m grid cell where m can be chosen by the numerical scheme and the boundary layer thickness, while they advocated that this off wall location be a specified fraction of the boundary layer thickness it is difficult to provide an off wall location at the m grid point with a specified y/δ fraction for complex geometries.
Bou-Zeid et al. (2004) proposed that since the mean wall stress algebraic formulation is related to the velocity squared in the matching location $\langle \tau_w \rangle \propto \left\langle u \left(y^* \right)^2 \right\rangle$ but the law of the wall satisfies $\langle \tau_w \rangle \propto \langle u \left(y^* \right) \rangle^2$ and of course $\left\langle u \left(y^* \right)^2 \right\rangle > \langle u \left(y^* \right) \rangle^2$.
Yang et al. (2017) suggests that the cause of the log layer mismatch is unphysically high correlation between the wall stress and the LES data at the first grid cell, this problem can be solved either by filtering the wall model input or using LES data further away from the wall.

2.2.3.6 Virtual Wall Boundary Conditions

Another approach is to determine the LES domain at some small height above the wall (h) and is provided a fitting boundary conditions, the wall stress is derived by integrating the TBLE equations up to h:

$$\frac{\partial \eta_0}{\partial t} = \frac{2\eta_0}{\tilde{u}|_h} \left[-\frac{1}{h} \tilde{uv}|_h - \frac{\partial \tilde{uu}|_h}{\partial x} - \frac{\partial \tilde{uw}|_h}{\partial z} - \frac{d\tilde{p}}{dx}|_h + \frac{v}{h} \left(\frac{\partial \tilde{u}}{\partial y}|_h - \eta_0 \right) \right]$$

Where $\tilde{.}$ denotes a wall-parallel filtration and $\eta_0 = \frac{\partial \tilde{u}}{\partial y}$. the wall slip velocity is determined by assuming - constant stress, logarithmic profile for the streamwise velocity, and no spanwise velocity at at $0 \leq y \leq h$.

2.3 Reynolds-Averaged Navier-Stokes (RANS)

This is the computationally cheapest model type for turbulent flow, it uses the RANS equations to find the time averaged flow field, for the incompressible case the equations are:

$$\rho \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \rho \bar{f}_i + \frac{\partial}{\partial x_j} \left[-\bar{p}\delta_{ij} + \mu \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \rho \overline{u_i' u_j'} \right]$$

To close this problem we need to model the Reynolds stresses ($\rho \overline{u_i' u_j'}$), the way this modeling is done is the main difference between various RANS models.
RANS is split into 3 different groups, LEVM, NEVM, RSM.

2.3.1 Linear Eddy Viscosity Models (LEVM)

This group contains all the common models of RANS ($k-\epsilon$,$k-\omega$,Spalrt-Allmaras, and more...), these models use the Boussinesq eddy viscosity assumption:

$$-\rho\overline{u_i' u_j'} = 2\mu_t S_{ij}^{\star} - \frac{2}{3}\rho k \delta_{ij}$$

Where k denotes turbulent kinetic energy (TKE) and:

$$k = \frac{1}{2}\left(\bar{u_i'}\right)$$

$$S_{ij}^{\star} = S_{ij} - \frac{1}{3}\frac{\partial \bar{u}_k}{\partial x_k}\delta_{ij}$$

$$S_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)$$

This modeling type stems from the clear connection of the Reynolds stresses and velocity gradient, and TKE consistency.

2.3.1.1 Algebraic models

In algebraic models, the mixing length (l_m) is used to find the turbulent viscosity (μ_t), l_m describes the average eddy size within every cell.

$$\mu_t = \rho k^{\frac{1}{2}} l_m$$

Since the eddies are expected to be smaller close to the wall, Prandtl suggested:

$$l_m = \kappa y,\ \kappa = 0.41$$

While Van Driest took into account the effect of the viscous sub-layer and suggested:

$$l_m = \kappa y\left(1 - \exp\left(-\frac{y^+}{A^+}\right)\right)$$

One of the main problems in those methods is that since they are algebraic the value of l_m needs to be set for every point in the grid, and it stays constant, which is a bad description for a chaotic flow such as turbulent flow, for that reason, advection models were created.

2.3.1.2 The $k-\epsilon$ Model

In this model we are looking for the dissipation (ϵ) instead of the mixing length, in this case:

$$\mu_t = C_\mu \frac{\rho k^2}{\epsilon}$$

The TKE equation is:

$$\frac{\partial(\rho k)}{\partial t} + \nabla(\rho U k) = \nabla\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\nabla k\right] + P_k + P_b - \rho\epsilon + S_k$$

Were the terms marked in P are production terms and terms marked in S are dissipative terms, the dissipation equation is:

$$\frac{\partial(\rho\epsilon)}{\partial t} + \nabla(\rho U \epsilon) = \nabla\left[\left(\mu + \frac{\mu_t}{\sigma_\epsilon}\right)\nabla\epsilon\right] + C_1\frac{\epsilon}{k}(P_k + C_3 P_b) + C_2\rho\frac{\epsilon^2}{k} + S_\epsilon$$

The constants C_1, C_2, C_3 are set by the model, after the dissipation equations are solved the TKE equations are solved and then μ_t can be found and the momentum equation can be solved. In this model we still need to take into account the effect of the wall, for that reason restraining function for C_1, C_2, C_3 were created - f_1, f_2, f_3, for example:

$$f_1 = 1$$

$$f_2 = 1 - 0.3 \exp\left(-Re_T^2\right)$$

$$f_3 = \exp\left(\frac{-3.4}{\left(1 + \left(Re_T/50\right)\right)^2}\right)$$

Where:

$$Re_T = \frac{\rho k^2}{\mu \epsilon}$$

This model is used primarily for large Reynolds numbers, otherwise, the $k - \omega$ model is used.

2.3.1.3 The $k - \omega$ Model

Since the $k - \epsilon$ model is inaccurate for separation and shock waves, the $k - \omega$ model was developed, this model is very similar to its counterpart but solves the equations for the specific turbulence dissipation (ω) instead of the equations for the dissipation (ϵ), in this case:

$$\omega = \frac{\epsilon}{C_\mu k}, \;\; C_\mu = 0.09$$

And the equation for ω is:

$$\frac{\partial \left(\rho\omega\right)}{\partial t} + \nabla \left(\rho U \omega\right) = \nabla \left[\left(\mu + \frac{\mu_t}{\sigma_\omega}\right) \nabla\omega\right] + \frac{\gamma}{\nu_t} P_k - \beta \rho \omega^2$$

In the $k - \omega$ the consideration of the effects of the wall are built-in, which improves its ability to predict separation.
On the other hand, the $k - \omega$ model is very dependent on the TKE of the main flow.

2.3.1.4 The $k - \omega$ SST (Shear Stress Transport) Model

This model attempt to combine the advantages of $k - \epsilon$ and $k - \omega$ by using $k - \epsilon$ of the main flow, $k - \omega$ close to the wall, and a function to combine them continuously

2.3.2 Nonlinear Eddy Viscosity Models (NEVM)

This group contains all models that don't use the Boussinesq eddy viscosity assumption but still close the model by finding μ_t, this group contains models like Cubic $k - \epsilon$, EARSM, v2-f, and more...

2.3.3 Reynolds Stress Model (RSM)

This model finds the Reynolds stresses by solving the Reynolds stress equation:

$$\frac{\partial}{\partial t}\left(\rho\overline{u_i'u_j'}\right)+\frac{\partial}{\partial x_k}\left(\rho u_k\overline{u_i'u_j'}\right)=-\frac{\partial}{\partial x_k}\left[\rho\overline{u_i'u_j'u_k'}+\overline{p'\left(\delta_{kj}u_i'+\delta_{ik}u_j'\right)}\right]+\frac{\partial}{\partial x_k}\left[\mu\frac{\partial}{\partial x_k}\left(\overline{u_i'u_j'}\right)\right]$$

$$-\rho\left(\overline{u_i'u_k'}\frac{\partial u_j}{\partial x_k}+\overline{u_j'u_k'}\frac{\partial u_i}{\partial x_k}\right)+\overline{p'\left(\frac{\partial u_i'}{\partial x_j}+\frac{\partial u_j'}{\partial x_i}\right)}-2\mu\overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_j'}{\partial x_k}}-2\rho\Omega_k\left(\overline{u_j'u_m'}\epsilon_{ikm}+\overline{u_i'u_m'}\epsilon_{jkm}\right)$$

The constants are set by the specific model.
Since this model contains 6 equations it is computationally expensive

Chapter 3

Latest Advancements In The Field Of Deep Learning Application In Turbulence Modeling

3.1 Reynolds Averaged Turbulence Modeling Using Deep Neural Networks With Embedded Invariance By Ling et al. (2016)

3.1.1 Description

MLP had been used to find the Reynolds stresses and close the RANS turbulent model before, but this paper presents a special model called tensor basis neural network (TBNN) that embeds Galilean invariance in the network, and shows improved results over other NN architectures or classical models.

- S - the mean strain tensor: $S_{ij} = \frac{1}{2}(k/\epsilon)(U_{i,j} + U_{j,i})$
- R - the mean rotation tensor: $R_{ij} = \frac{1}{2}(k/\epsilon)(U_{i,j} - U_{j,i})$
- b - the mean Reynolds stress tensor: $b_{ij} = \overline{u_i' u_j'}/2k - 1/3\delta_{ij}$

Pope (1975) proved that the mean Reynolds stress tensor can be expressed as a linear combination of 10 isotropic basis tensors:

$$b = \sum_{n=1}^{10} g^{(n)}(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)\, T^{(n)}$$

And gave a detailed derivation of these 10 tensors, and $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$, expressing them as a function of S, R and the unit tensor (I):

$$\begin{cases} T^{(1)} = S & T^{(6)} = R^2S + SR^2 - \frac{2}{3}I \cdot Tr(SR^2) \\ T^{(2)} = SR - RS & T^{(7)} = RSR^2 - R^2SR \\ T^{(3)} = S^2 - \frac{1}{3}I \cdot Tr(S^2) & T^{(8)} = SRS^2 - S^2RS \\ T^{(4)} = R^2 - \frac{1}{3}I \cdot Tr(R^2) & T^{(9)} = R^2S^2 + S^2R^2 - \frac{2}{3}I \cdot Tr(S^2R^2) \\ T^{(5)} = RS^2 - S^2R & T^{(10)} = RS^2R^2 - R^2S^2R \end{cases}$$

$$\lambda_1 = Tr\left(S^2\right),\ \lambda_2 = Tr\left(R^2\right),\ \lambda_3 = Tr\left(S^3\right),\ \lambda_4 = Tr\left(R^2S\right),\ \lambda_5 = Tr\left(R^2S^2\right)$$

Where $Tr(x)$ represents the trace of x.
Therefore, the only part left for the TBNN is to find the values of g using the eigenvalues for every point in the grid, and since the eigenvalues are independent of the coordinate system orientation - so is the solution.
In Figure 3.1 the classic MLP architecture is presented next to the TBNN architecture. The TBNN model takes the invarients $\lambda_1, ..., \lambda_5$ as inputs and outputs $g^{(n)}$, then $g^{(n)}$ is merged with $T^{(n)}$ to result with b.

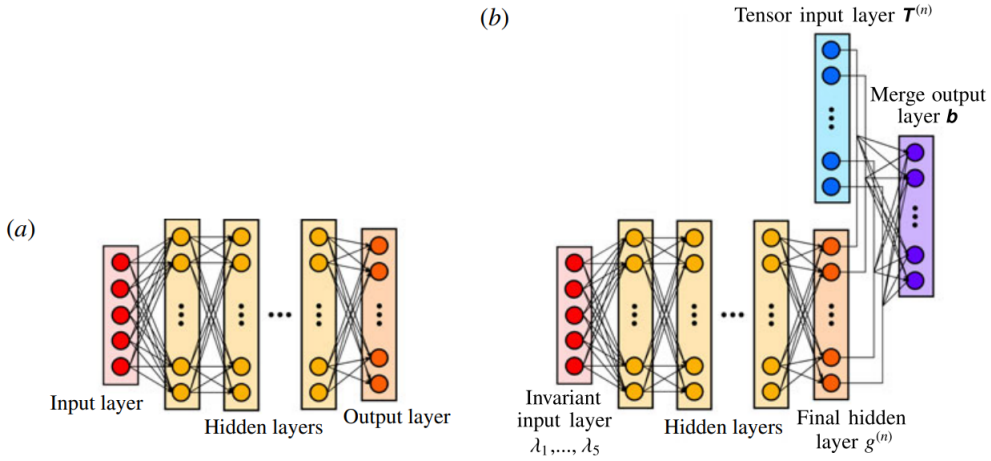

Figure 3.1: Schematic of (a) the "vanila" MLP model (b) the TBNN model

3.1.2 Results

Nine data sets for canonical turbulent flow cases were used, six for training, one for validation, and two for testing, all of these data sets were gathered by DNS simulations:

- Training flows - duct flow ($Re_b = 3500$), channel flow ($Re_\tau = 590$), perpendicular jet in cross flow around a square cylinder, flow through a converging-diverging channel.
- Validation flow - a wall-mounted cube in cross flow ($Re = 5000$)
- Testing flows - duct flow ($Re_b = 2000$), flow over a wavy wall ($Re = 6850$), while the first flow is pretty similar to a flow from the training data, the second one is completely different.

In Figure 3.2 the results of the TBNN are compared with the results of the DNS simulations, a regular MLP, and some classic CFD models like LEVM and QEVM (quadratic eddy viscosity model):

Model	Duct flow	Flow over wavy wall
LEVM	0.25	0.18
QEVM	0.20	0.11
TBNN	0.14	0.08
MLP	0.31	0.09

Figure 3.2: Root mean squared error on test cases

The results of the TBNN were the best ones in both cases, and it's the only model that managed to predict separation in the wavy wall flow (see Figure 3.3)

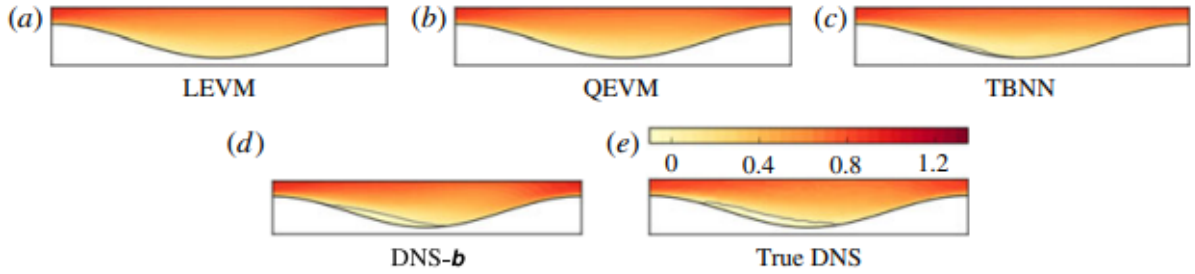

Figure 3.3: Contours of the mean streamwise velocity normalized by bulk velocity in a wavy wall test case

3.2 Machine Learning-Accelerated Fluid Dynamics By Kochkov et al. (2021)

3.2.1 Description

This paper focuses on the ability of ML and DL to accelerate DNS and LES simulation, the two ML components this paper discusses are:

- Learned interpolation - rather than using typical polynomial interpolation in order to find the convective term $u \cdot \nabla u$ we use ML model to interpolate the convective term based on local features.
- Learned Correction - this approach is closer to LES in spirit, by using the ML model a residual correction is found, this correction is added to the velocity field from the numerical solver.

The method used to solve the velocity field is described in Figure 3.4, $v\,(t)$ is passed through the CNN, the CNN filters have some constraints to improve the result (for example constraining the filters to sum to unity makes sure that the interpolation is always at least first order accurate with respect to the grid spacing), the result provides the convective flux, then the divergence operator enforces local conservation of momentum, then the explicit time step that provides a continuous dynamics in time, and finally the pressure projection that enforces incompressibility.

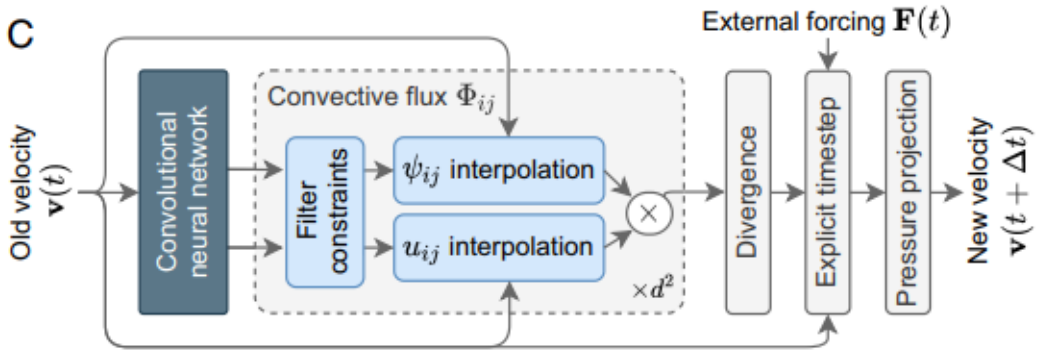

Figure 3.4: Schematic of the interpolation method

3.2.2 Results

3.2.2.1 Accelerating DNS

DNS accuracy usually degrades quickly with lower grid resolution, however, with the ML based approach, this effect was strongly mitigated. Figure 3.5 shows the huge improvements in the results for Kolmogorov flows (two dimensional and unidirectional shear flow with a specific sinusoidal mean velocity profile) at $Re = 1000$, the results were especially good for the energy spectrum (Figure 3.5 Image C), since the lack of sufficient grid resolution hit the higher frequency features the hardest.

Since the ML based approach works around as good as a 10X finer grid, and each point takes around 12X more time to calculate using the ML based approach, the ML based approach ends up being 80X faster (!!!).

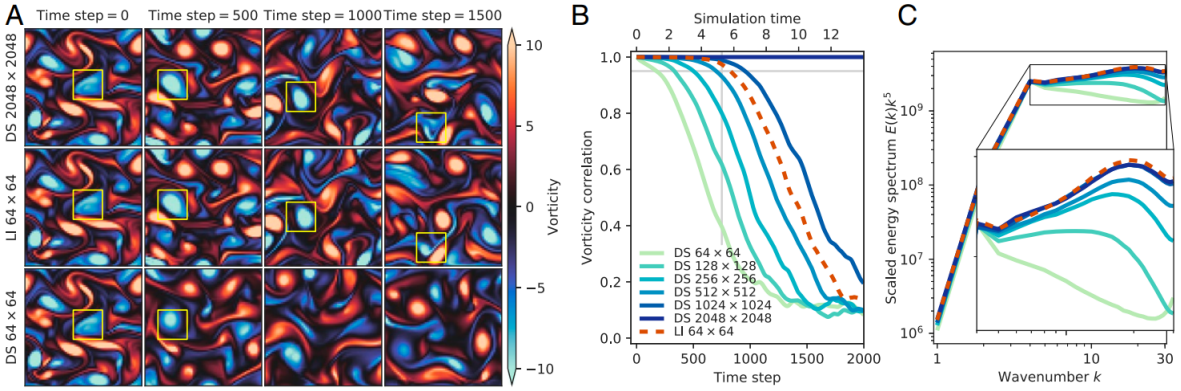

Figure 3.5: (A) Evolution of predicted vorticity fields (B) Vorticity correlation comparison (C) Scaled energy spectrum - for Kolmogorov flows at $Re = 1000$ (training set)

However, these results are for the training set. For the testing set a decaying turbulent flow and higher Re number Kolmogorov flow ($Re = 4000$) was used, since the model learns local operators it is expected to generalize well, and indeed, for decaying turbulence the ML based approach works around as good a 8X finer resolution (see Figure 3.6),and 6X for the $Re = 4000$ Kolmogorov flow (see Figure 3.7).

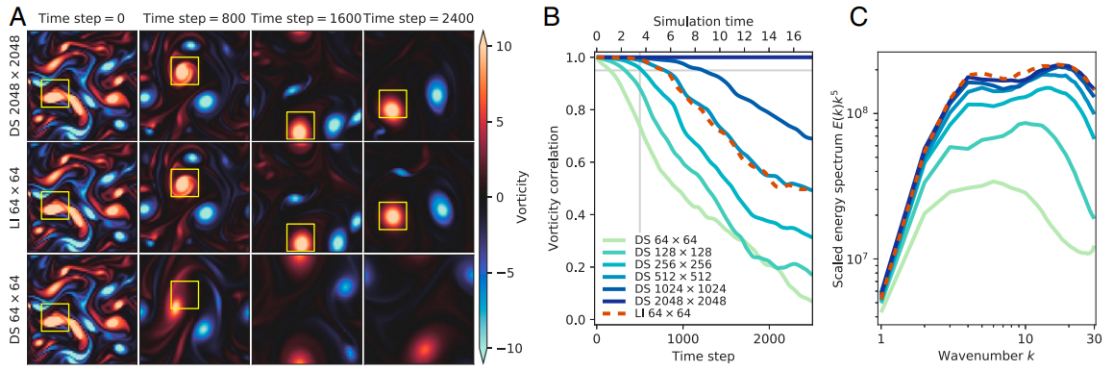

Figure 3.6: (A) Evolution of predicted vorticity fields (B) Vorticity correlation comparison (C) Scaled energy spectrum - for decaying turbulent at $Re = 1000$ (testing set)

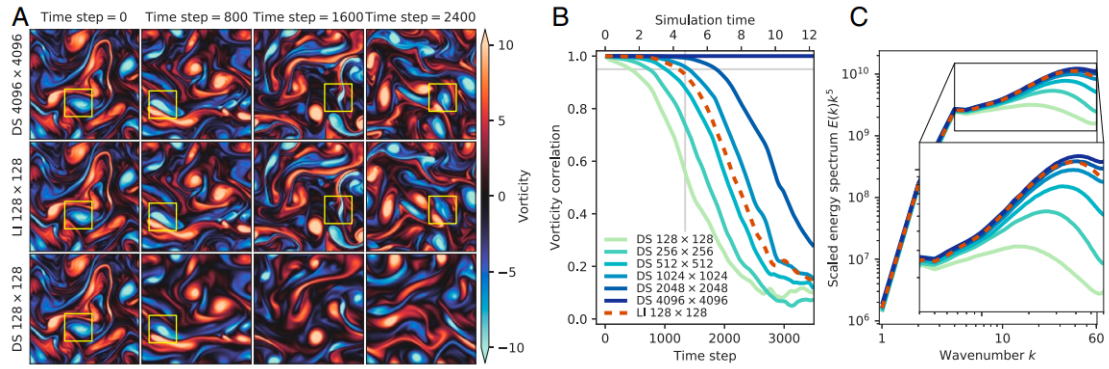

Figure 3.7: (A) Evolution of predicted vorticity fields (B) Vorticity correlation comparison (C) Scaled energy spectrum - for Kolmogorov flows at $Re = 4000$ (testing set)

3.2.2.2 Accelerating LES

For LES the results were pretty similar, and since the method is local and works with interpolation of the current solution using it for LES doesn't demand any special treatment.

3.3 RANS Turbulence Model Development Using CFD-Driven Machine Learning By Zhao et al. (2020)

3.3.1 Description

This paper focuses on the use of GEP in order to improve RANS accuracy, the paper's approach is fixing the stress tensor by adding an extra anisotropic stress a_{ij}:

$$\tau_{ij} = \frac{2}{3}\rho k \delta_{ij} - 2\mu_t S'_{ij} + a_{ij}$$

Where a_{ij} is defined as Pope (1975) suggested :

$$a_{ij} = \sum_k f^{(k)}\left(I_1, I_2, ..., I_n\right) V_{ij}^k$$

Where k and n are constant that depend on the flow dimension.
The flows chosen for the training and testing of the model were high pressure turbine and low pressure turbine, since those flows are challenging for RANS calculations due to their extremely complex flow features.
Since the paper focuses on statistically 2d flows which are representative of mid span section of turbine blades:

$$k = 3, \quad n = 2$$

$$V_{ij}^1 = s_{ij}, \; V_{ij}^2 = s_{ik}\omega_{kj} - \omega_{ik}s_{kj}$$

$$V_{ij}^3 = s_{ik}s_{kj} - \frac{1}{3}\delta_{ij}s_{mn}s_{nm}$$

$$I_1 = s_{mn}s_{nm}, \; I_2 = \omega_{mn}\omega_{nm}$$

Where s_{ij} is the non-dimensionalized strain rate tensor and ω_{ij} is the non-dimensionalized rotation rate tensor:

$$s_{ij} = t_I S'_{ij}, \; \omega_{ij} = t_I \Omega_{ij}, \; t_I = \frac{1}{\omega}$$

and ω is the specific dissipation rate.
Two methods to deploy the GEP model have been suggested in this paper:

3.3.1.1 Frozen Training

In the frozen training method, candidate models a_{ij}^{GEP} are generated by GEP and compared to a_{ij}^{HiFi} obtained from the HiFi data, the cost function is defined as:

$$J^{fro} = \frac{1}{N}\sum_{n=1}^{N}\sum_{i,j}\frac{\left|a_{ij}^{HiFi} - a_{ij}^{GEP}\right|}{\left|a_{ij}^{HiFi}\right|}$$

Where N denotes the number of training data points. With the "frozen" here meaning the preprocessed HiFi data remains unchanged during the training, However, once the model is implemented into RANS, good performance is not guaranteed. In fact RANS will predict a flow field that may deviate from the Hi-Fi flow field.

3.3.1.2 CFD Driven Training

As shown in Figure 3.8, in the CFD driven training, different models are tested in parallel for every generation of candidate models, then the cost function is based on the difference between the RANS results and the HiFi data. based on the cost function of the different models in the current generation, a new generation of models is formed through GEP evolution. This model iterates and the population of models evolves until the best model provides an acceptable cost value.
One big advantage of this method is the ability to choose the cost function in many different ways, and change according to the need of the designer.

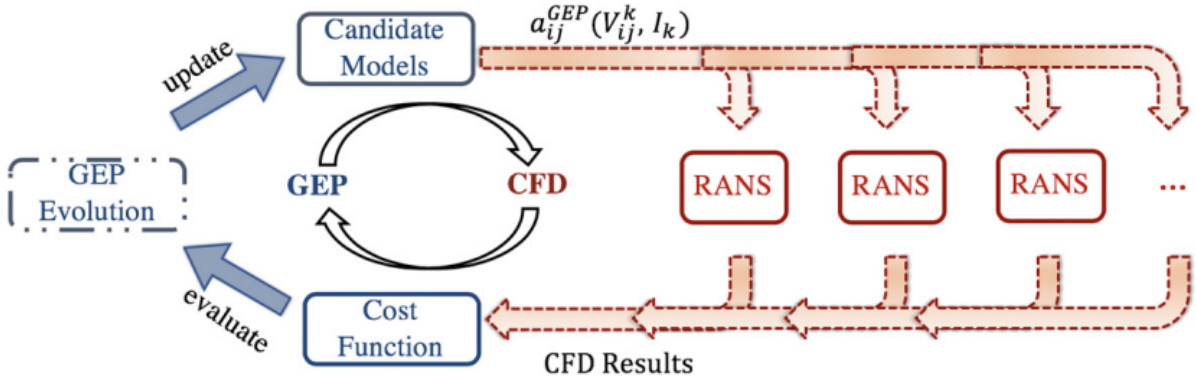

Figure 3.8: Schematic of the model training in the CFD driven training method

3.3.2 Results

According to Pichler et al. (2016), the common weakness of traditional RANS models is capturing wake mixing. Therefore the CFD driven cost function is defined as:

$$J^{CFD} = \Delta_{x_1} + \Delta_{x_2}$$

Where:

$$\Delta_x = \frac{1}{L_y} \int_0^{L_y} \left(\frac{\omega_{HiFi}(y) - \omega_{RANS}(y)}{\max_y(\omega_{HiFi})} \right)^2 dy$$

$$\omega(y) = \frac{p_t^i i - p_t(y)}{p_t^i - p^o}$$

Here, $\omega(y)$ represents the kinetic energy loss profile along the pitchwise axis y (see Figure 3.9), p_t^i, p_t^o and p^o denoting inlet total pressure, outlet total pressure, and outlet static pressure, respectively. Δ_x stands for normalized deviation between the wake loss profiles from HiFi and RANS results. And $x_1 = 1.15C_{ax}$, $x_2 = 1.25C_{ax}$ (in order to attempt to test the whole wake region).

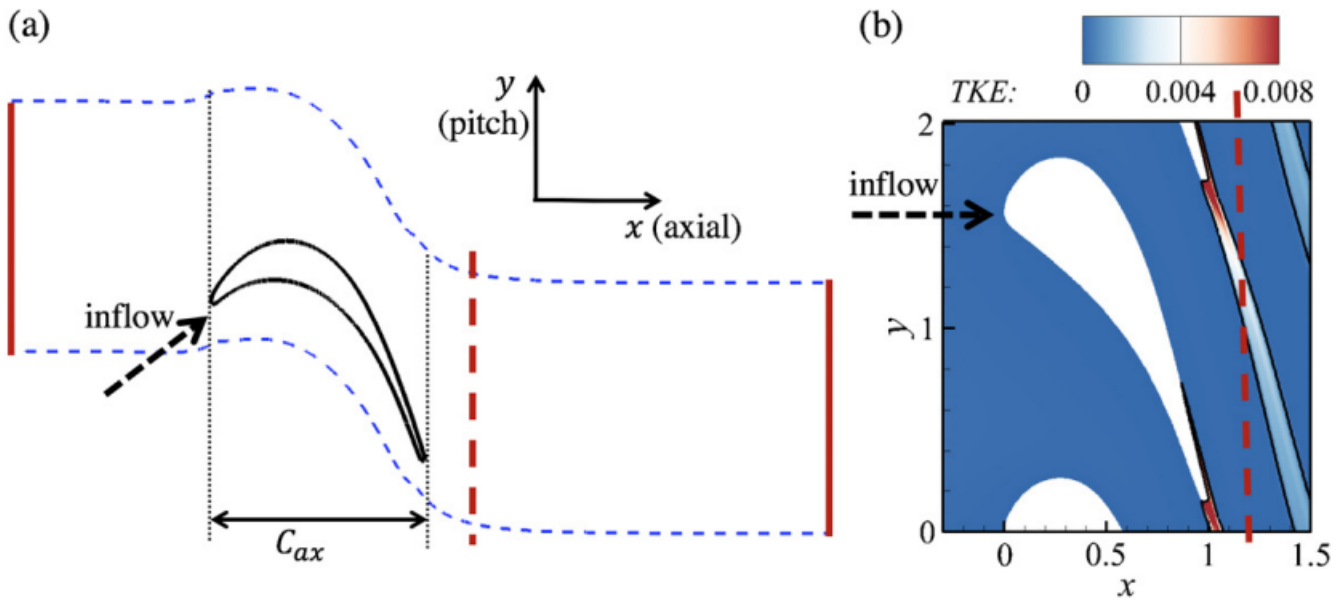

Figure 3.9: A visualization of the simulation environment

The resulting stress tensors were:

$$\tau_{ij}^{CFD} = \frac{2}{3}\rho k\delta_{ij} - 2\mu_t S'_{ij} + 2\rho k\left[\left(-2.57 + I_1\right)V_{ij}^1 + 4.0V_{ij}^2 + \left(-0.11 + 0.09I_1I_2 + I_1I_2^2\right)V_{ij}^3\right]$$

$$\tau_{ij}^{fro} = \frac{2}{3}\rho k\delta_{ij} - 2\mu_t S'_{ij} + 2\rho k[$$

$$\left(-1.334 + 0.438I_1 + 2.653I_2 + 0.0102I_1^2 - 1.021I_2^2 + 12.280I_1I_2\right)V_{ij}^1$$

$$+ \left(0.573 - 1.096I_1 + 8.985I_2 - 0.1102I_1^2 + 2.876I_2^2 + 90.633I_1I_2\right)V_{ij}^2$$

$$+ \left(12.861 - 25.094I_1 + 6.449I_2 + 1.020I_1^2 - 304.979I_1I_2 - 184.519I_2^2\right)V_{ij}^3]$$

It is noted that the CFD driven model has a simpler form compared to the frozen model, that's because the CFD driven model needs to run and converge for RANS simulation (if a model doesn't converge the loss will be set as extremely high and it couldn't be the chosen model), therefore the τ_{ij}^{CFD} chosen will have to be very stable, and some high terms will not survive the training.
By looking at the leading terms:

$$\tau_{ij}^{CFD-L} = \frac{2}{3}\rho k\delta_{ij} - 2\mu_t\left(1 + 2.57\right)S'_{ij}$$

$$\tau_{ij}^{fro-L} = \frac{2}{3}\rho k\delta_{ij} - 2\mu_t\left(1 + 1.334\right)S'_{ij}$$

We can see that the CFD driven method causes additional diffusion, appropriate to the problem of wake mixing and thus a better prediction of the wake profile spreading as seen in Figure 3.10 (where $E_\omega = \frac{\left|\omega_{max}^{HiFi} - \omega_{max}^{RANS}\right|}{\omega_{max}^{HiFi}}$).

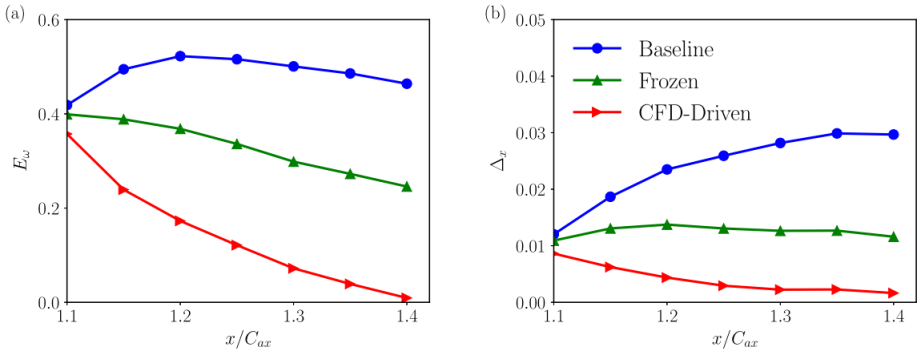

Figure 3.10: Graphs of E_ω and Δ_x as a function of $x/C_{\alpha x}$

3.4 Deep learning to replace, improve, or aid CFD analysis in built environment applications: A review By Calzolari and Liu (2021)

Some of the other papers on this subject can be divided to two groups: constant tuning methods and turbulence models enhancements.

3.4.1 Constant Tuning Methods

Constant tuning methods are methods that use deep learning in order to find the best coefficients of the turbulent models for each problem presented to the model.
Yarlanki et al. (2012) estimated the temperature in data centers using an MLP model. Compared to a RANS model the results improved by 35%, and Luo et al. (2020) did a similar work but provided a physics informed neural network and embedded the turbulence physics in the loss function, this acted as sort of regularization for the neural network.

3.4.2 Turbulence Models Enhancements

Turbulence models enhancements are methods that use deep learning in order to change the structure of the turbulence model, to act on the source of uncertainties such as the evident failure of the eddy viscosity hypothesis. Some representative works are:
Tracey et al. (2015) considered a one equation turbulence model and a Spalart Allmaras model and used an MLP to find the lengthscale for the one equation turbulence model by training it with the Spalart Allmaras model. In later works the Spalart Allmaras model was replaced with DNS and well resolved LES data.
Singh et al. (2017) replaced the Spalart Allmaras model with measured lift coefficient data from experiments and tried to minimize the difference between it and the simulation.
Wang et al. (2017) used random forest algorithm to find a representation for the Reynolds stresses.
Rudy et al. (2019) used a CNN to create a novel set of 2D N-S equations to solve the spanwise average of the flow directly (also called the SANS equations), and thus this method allows the use of inexpensive 2D simulations instead of the demanding 3D simulations.

Chapter 4

Future work

- The separation location on a body can be learned using a classic NN, and the area of separation can be learned using a CNN, by knowing upfront the places separation can occur classic RANS models such as $k-\omega$ and LES models can improved drastically.
- A quick and simple NN can be used to replace wall models by finding the change in the wall equations constants (may change according to the wall equations chosen). One can also add effect of some polynomial of arbitrary order (and of course study it's constants using the NN).
- TCN unique architecture can be adopted to improve regular TCN performance specifically for fluid mechanics - different size order kernels can be chosen. For the large kernels, large time dilation to capture the large, slow, flow properties, and for the small kernels, small time dilation to capture the small, quick, flow properties.

Bibliography

S. T. Bose and G. I. Park. Wall-modeled large-eddy simulation for complex turbulent flows. *Annual review of fluid mechanics*, 50:535–561, 2018.

E. Bou-Zeid, C. Meneveau, and M. B. Parlange. Large-eddy simulation of neutral atmospheric boundary layer flow over heterogeneous surfaces: Blending height and effective surface roughness. *Water Resources Research*, 40(2), 2004.

S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

G. Calzolari and W. Liu. Deep learning to replace, improve, or aid cfd analysis in built environment applications: A review. *Building and Environment*, 206:108315, 2021.

D. R. Chapman. Computational aerodynamics development and outlook. *AIAA journal*, 17 (12):1293–1313, 1979.

H. Choi and P. Moin. Grid-point requirements for large eddy simulation: Chapmanâs estimates revisited. *Physics of Fluids*, 24(1):011702, 2012. doi: 10.1063/1.3676783.

S. Hickel, E. Touber, J. Bodart, and J. Larsson. A parametrized non-equilibrium wall-model for large-eddy simulations. In *Eighth International Symposium on Turbulence and Shear Flow Phenomena*. Begel House Inc., 2013.

S. Kawai and J. Larsson. Wall-modeling in large eddy simulation: Length scales, grid resolution, and accuracy. *Physics of Fluids*, 24(1):015105, 2012.

J. Kim and P. Moin. Application of a fractional-step method to incompressible navier-stokes equations. *Journal of computational physics*, 59(2):308–323, 1985.

D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

J. N. Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017.

J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

S. Luo, M. Vellakal, S. Koric, V. Kindratenko, and J. Cui. Parameter identification of rans turbulence model using physics-embedded neural network. In *International Conference on High Performance Computing*, pages 137–149. Springer, 2020.

G. I. Park and P. Moin. An improved dynamic non-equilibrium wall-model for large eddy simulation. *Physics of Fluids*, 26(1):37–48, 2014.

R. Pichler, R. Sandberg, V. Michelassi, and R. Bhaskaran. Investigation of the accuracy of rans models to predict the flow through a low-pressure turbine. *Journal of Turbomachinery*, 138 (12):121009, 2016.

S. Pope. A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 72(2): 331–340, 1975.

S. Rudy, A. Alla, S. L. Brunton, and J. N. Kutz. Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, 18(2):643–660, 2019.

R. D. Sandberg and Y. Zhao. Machine-learning for turbulence and heat-flux model development: A review of challenges associated with distinct physical phenomena and progress to date. *International Journal of Heat and Fluid Flow*, 95:108983, 2022.

A. P. Singh, S. Medida, and K. Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA journal*, 55(7):2215–2227, 2017.

B. D. Tracey, K. Duraisamy, and J. J. Alonso. A machine learning strategy to assist turbulence model development. In *53rd AIAA aerospace sciences meeting*, page 1287, 2015.

J.-X. Wang, J.-L. Wu, and H. Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, 2 (3):034603, 2017.

M. Wang and P. Moin. Dynamic wall modeling for large-eddy simulation of complex turbulent flows. *Physics of Fluids*, 14(7):2043–2051, 2002.

P. Wu and J. Meyers. A constraint for the subgrid-scale stresses in the logarithmic region of high reynolds number turbulent boundary layers: A solution to the log-layer mismatch problem. *Physics of Fluids*, 25(1):015104, 2013.

X. Yang, J. Sadique, R. Mittal, and C. Meneveau. Integral wall model for large eddy simulations of wall-bounded turbulent flows. *Physics of Fluids*, 27(2):025112, 2015.

X. I. Yang and K. P. Griffin. Grid-point and time-step requirements for direct numerical simulation and large-eddy simulation. *Physics of Fluids*, 33(1):015108, 2021.

X. I. Yang, G. I. Park, and P. Moin. Log-layer mismatch and modeling of the fluctuating wall stress in wall-modeled large-eddy simulations. *Physical review fluids*, 2(10):104601, 2017.

S. Yarlanki, B. Rajendran, and H. Hamann. Estimation of turbulence closure coefficients for data centers using machine learning algorithms. In *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pages 38–42. IEEE, 2012.

Y. Zhao, H. D. Akolekar, J. Weatheritt, V. Michelassi, and R. D. Sandberg. Rans turbulence model development using cfd-driven machine learning. *Journal of Computational Physics*, 411:109413, 2020.