Applications in Practical High-End Computing

Alexis Balayre[1,a)], Deng Chuan Chang[1,b)], Chung-Yueh Cheng[1,c)], Julien Godfroy[1,d)], Lyderic Faure[1,e)], Majuran Chandrakumar[1,f)], and Marcell Gyorei[1,g)]

[1] Cranfield University, College Road, Cranfield MK43 0AL, United Kingdom
[a)] alexis.balayre.500@cranfield.ac.uk
[b)] DengChuan.Chang.672@cranfield.ac.uk
[c)] Chung-Yueh.Cheng.101@cranfield.ac.uk
[d)] Julien.Godfroy.183@cranfield.ac.uk
[e)] Lyderic.Faure.062@cranfield.ac.uk
[f)] Majuran.Chandrakumar.255@cranfield.ac.uk
[g)] Marcell.Gyorei.494@cranfield.ac.uk

Abstract: This project investigates the integration of machine learning techniques to advance turbulence modeling in the realm of computational fluid dynamics (CFD). The objective is to overcome the challenges posed by traditional Reynolds-Averaged Navier-Stokes (RANS) models through the leverage of Physics-Informed Neural Networks (PINNs) and the employment of the Sparse Identification of Nonlinear Dynamical Systems (SINDy) approach. The primary aim focuses on developing predictive models capable of accurately simulating turbulent flows under various conditions, alongside enhancing CFD simulation efficiency through the use of high-performance computing resources. The methodology involves the creation and validation of machine learning models trained using Direct Numerical Simulation (DNS) data, with an emphasis on improving simulation capabilities across diverse flow regimes. This entails meticulous testing and validation processes to fine-tune parameters before integration into customised software. Furthermore, a Streamlit application has been developed to support user interaction and facilitate the visualisation of turbulence modeling predictions. Results indicate that the PINNs model exhibits superior performance in capturing the dynamics of turbulent flows more accurately than conventional models, particularly in intricate boundary layer contexts. The integration of these models into standard CFD workflows has the potential to significantly enhance the prediction of turbulent behaviors, offering substantial reductions in computational costs while retaining high levels of accuracy.

Keywords: physics-informed neural networks, computational fluid dynamics, reynolds-averaged navier-stokes, sparse identification of nonlinear dynamical systems, turbulence modeling, turbulent channel flows

1 Introduction

1.1 Background

The significance of RANS equations in CFD for predicting turbulent fluid flow is undeniable. By breaking down the fluid flow into mean and fluctuating components, RANS models, ranging from simple to sophisticated, enable precise fluid flow simulations under various conditions. This is essential for designing machines and vehicles across sectors such as aerospace, automotive, wind engineering, and shipbuilding. The common challenge in CFD, namely the unpredictable nature of turbulence, is addressed through methods like the Boussinesq hypothesis and the $k - \epsilon$, $k - \omega$ models, enhancing the practicality of fluid simulations.

1.2 Motivation

CFD's impact extends across industries, notably in aerospace and automotive, where it drives significant advancements in aerodynamics for efficiency and safety. Examples include Boeing's drag reduction on the 747 wings, contributing to substantial fuel cost savings, and automotive research suggesting a 20% reduction in drag and environmental pollution. Inspired by these achievements, our research aims to merge machine learning with turbulence modeling to further optimise design, operational safety, and environmental sustainability.

1.3 Aim

This study seeks to exploit machine learning and deep learning techniques on DNS datasets to develop advanced turbulence models. The end goal is to create a software tool that simplifies comparing these models' predictions with DNS data, offering insights into their accuracy and efficiency. This exploration will underline the most promising approaches for future research in turbulence prediction.

1.4 Objectives

- Simulate channel flows using CFD software
- Implement diverse machine learning strategies
- Evaluate these strategies for their effectiveness
- Develop user-friendly software for turbulence flow prediction

2 Literature Review

2.1 PINNs

PINNs represent a novel approach that integrates differential equation solvers with the flexibility and power of deep learning. By embedding physical laws as part of the training process, PINNs offer the potential to solve complex PDEs across various domains, including fluid dynamics, structural engineering, and beyond.

2.1.1 Fundamental Concepts

PINNs leverage the structure of neural networks to approximate the solutions of PDEs. These models are informed by physical laws, which are encoded directly into the loss function as regularisation terms. This approach ensures that the solutions not only fit the available data but also comply with the governing physical principles.

2.1.2 Turbulence Modeling for PINNs

Since the introduction of the PINNs model design, many researchers have used it to solve the RANS equations. These include the following papers, which have made real advances in turbulence prediction models: Sandberg, Hanrahan, Kozul, Jagode, Anzt, Juckeland, and Ltaief [10], Luo, Vellakal, Koric, Kindratenko, and Cui [7], Eivazi, Tahani, Schlatter, and Vinuesa [5], Pioch, Harmening, Müller, Peitzmann, Schramm, and el Moctar [8].

2.1.3 Methodological Advances

Several methodological advances have been instrumental in enhancing the performance and applicability of PINNs. For instance, Bischof and Kraus [1] introduced a Mixture-of-Experts-Ensemble Meta-Learning approach, which significantly improves PINNs's ability to solve complex PDEs by utilising an ensemble of specialised neural networks. Additionally, Lagaris, Likas, and Fotiadis [6] pioneered the use of neural networks for solving differential equations, laying the groundwork for the development of PINNs. Moreover, choosing the right activation function is critical for the success of PINNs, as it affects the model's ability to capture complex behaviors governed by the PDEs. Traditional activation functions like ReLU may not be suitable due to their limited differentiability. Instead, functions with higher order differentiability, such as the sine function, have been explored by Raissi, Perdikaris, and Karniadakis [9], Bischof and Kraus [1].

2.2 PySINDy

The source code for the PySINDy models are available in the following GitHub repository: PySINDy_Models Repository.

PySINDy is recognised as a type of machine learning and has gained attention for its ability to identify actuated systems, PDEs and implicit differential equations. It applies the concept of sparse identification of non-linear dynamics, using sparse matrices to provide a new approach in computational modeling [3].

One of the core concepts of PySINDy is that it will generate dynamic equations for each variable; in other words, it will not combine them into a single equation for the whole system. There is a huge advantage to this, as it allows a better understanding of the behaviour of individual components within the whole dynamic system. The implementation of PySINDy has been developed based on the SINDy method which is used to discover the connections between each variable that follows the time series in the dynamical system models. This method is instrumental in simplifying the process of modeling complex systems, enabling researchers to remain focused on the important areas of system dynamics [2].

The use of the SINDy method is to solve the modeling of dynamical systems as a sparse regression problem. Using the time series data matrix X of state variables and library functions θ(X), SINDy would try to find a sparse coefficient matrix Ξ to satisfy the following conditions [4]:

$$\dot{X} \approx \Theta(X)\Xi$$

Each row corresponds to a partial equation for a state variable. After that, SINDy will solve this matrix approximation problem to reveal the interrelationships between each variable in dynamic systems.

It is well-known for PySINDy to use at analysing the Lorentz system, the classic chaotic system [4]. It is a simple nonlinear differential equation system which composed of three variables that can produce extremely complex chaotic behavior, and PySINDy can accurately identify the governing equations of it from time series data, demonstrating its powerful ability to capture the essential dynamics of chaotic systems.

Moreover, because PySINDy's input is suitable for time series format; this requirement aligns well to the method's focus on time series data processing, making it suitable for a wide range of applications from fluid dynamics to biological systems.

2.3 Computational Fluid Dynamics

CFD is a branch of fluid mechanics that employs numerical methods and algorithms to analyse and solve problems involving fluid flows. Rooted in the 1960s, evolved rapidly with advances in computing technology. Initially developed for aerospace applications, it expanded to diverse fields due to its versatility. It encompasses a wide range of applications, from aerospace and automotive engineering to environmental science and biomedical research. CFD simulations enable engineers and scientists to predict fluid behavior, such as velocity, pressure, and temperature distributions, in complex geometries and under various operating conditions. By providing insights into fluid flow phenomena, CFD plays a crucial role in the design optimisation, performance evaluation, and troubleshooting of engineering systems. This section compiles the key concepts to understand basic theory behind CFD and further research in this report.

2.3.1 Navier-Stokes equation

Navier-Stokes equations serve as the fundamental equations governing fluid motion in Computational Fluid Dynamics. They mathematically describe the conservation of mass, momentum and energy, providing a framework to simulate and predict fluid behaviour in complex systems. By solving these partial differential equations numerically, CFD models can predict flow characteristics such as velocity, pressure, and temperature distributions.

2.3.2 RANS closure problem

The RANS equations, while effective for simulating mean flow behaviour in turbulent flows, face challenges due to the RANS closure problem. This arises from the need to model the effects of unresolved turbulent fluctuations. Turbulence models, such as k-epsilon or k-omega, aim to close the system by expressing turbulent quantities in terms of mean flow variables. However, these models rely on assumptions and simplifications, leading to inaccuracies in certain flow conditions. The sensitivity to flow conditions and the lack of universal models contribute to ongoing research efforts. Improving closure strategies remains crucial for enhancing the accuracy and reliability of RANS simulations.

2.3.3 Models

The k-ω and Reynolds Stress Model (RSM) are two popular turbulence models used in RANS simulations to predict turbulent flow behaviour.

The k-ω model is a two-equation turbulence model that solves transport equations for turbulent kinetic energy (k) and specific dissipation rate (ω). It excels in accurately predicting near-wall flows due to its enhanced treatment of near-wall turbulence. The k-ω model offers robustness and efficiency, making it suitable for a wide range of engineering applications, especially those involving complex geometries and boundary layers. However, its performance can be limited in highly anisotropic flows or those with strong rotational effects.

In contrast, the RSM directly solves for the Reynolds Stress Tensor (RST) components, offering a more comprehensive representation of turbulence. It captures anisotropic turbulence effects more accurately and is suitable for flows with significant rotational or strain-rate effects. The RSM provides detailed insights into turbulence behaviour, making it valuable for research and engineering applications requiring high-fidelity simulations. However, it is computationally expensive and sensitive to grid resolution and modelling assumptions, limiting its applicability in some cases.

Choosing between the k-ω and RSM depends on the flow characteristics, computational resources, and accuracy requirements of the simulation.

3 Methodology

3.1 PINNs

A PINNs model was designed and trained to predict the RST $\boldsymbol{\tau}$ of a plate in a turbulent channel flow.

3.1.1 Data Exploration

The training dataset was generated using the DNS simulations of Oden database. These simulations are DNS of incompressible turbulent flow between two parallel planes. Periodic boundary conditions are applied in the streamwise (x) and spanwise (z) directions, and no-slip/no-penetration boundary conditions are applied at the wall. The computational domain sizes are $L_x = 8\pi\delta$ and $L_z = 3\pi\delta$, where δ is the channel half-width, so the domain size in the wall-normal (y) direction is 2δ. The flow is driven by a uniform pressure gradient, which varies in time to ensure that the mass flux through the channel remains constant.

In channel flow, the primary motion of the fluid is in the streamwise (x) direction due to the pressure gradient applied in this direction. This makes $\overline{U}$, the mean streamwise velocity, the most significant component for characterising the flow. $\overline{V}$ (wall-normal velocity) and $\overline{W}$ (spanwise velocity) are usually much smaller by comparison and often average out to zero when considering fully developed flow, especially in simulations where the flow is assumed to be statistically steady and homogeneous in the streamwise and spanwise directions. Besides, here turbulent channel flows are supposed statistically homogeneous in the streamwise (x) and spanwise (z) directions, meaning that any statistical property of the flow (when averaged over time) does not vary along these directions. This homogeneity implies that the derivatives $\frac{\mathrm{d}\overline{U}}{\mathrm{d}x}$, $\frac{\mathrm{d}\overline{V}}{\mathrm{d}x}$, $\frac{\mathrm{d}\overline{W}}{\mathrm{d}x}$, and $\frac{\mathrm{d}\overline{P}}{\mathrm{d}x}$ tend to zero.

3.1.2 Targets & Features Selection

The PINNs model focuses on capturing critical aspects of turbulent flow through the following outputs:

- **Streamwise velocity variance** ($u'u'$): Measures fluctuations along the flow direction.
- **Wall-normal velocity variance** ($v'v'$): Captures fluctuations perpendicular to the wall.
- **Spanwise velocity variance** ($w'w'$): Indicates fluctuations across the flow width.
- **Streamwise-wall-normal velocity covariance** ($u'v'$): Reflects momentum transfer.
- **Mean streamwise velocity** (U): Shows the average flow velocity profile.
- **Mean streamwise velocity gradient** ($\partial U/\partial y$): Essential for shear rate analysis.
- **Mean pressure** (P): Provides force distribution insights.
- **Turbulent kinetic energy** (k): Quantifies energy in turbulent eddies.

The chosen input features aim to comprehensively represent the flow's dynamics:

- **Kinematic viscosity** (ν): Fundamental to fluid flow characteristics.
- **Friction velocity** (u_τ): Scales velocity near the wall.
- **Reynolds number based on friction velocity** (Re_τ): Describes the flow regime.
- **Wall-normal grid point in wall units** (y^+): Utilised for boundary layer analysis.
- **Wall-normal grid point normalised by channel half-width** (y/δ): Indicates position within the flow domain.

3.1.3 Justification for Feature Redundancy

Though some selected features may appear redundant, their inclusion is purposeful, serving to:

1. **Comprehensive Flow Characterisation**: Captures both specific and general turbulence properties.
2. **Capturing Flow Scales**: Ensures modeling accuracy across different flow scales, important for both near-wall behavior and core region dynamics.
3. **Enhancing Model Robustness and Accuracy**: Improves training, reduces overfitting risk, and ensures generalisability across fluid dynamics applications.

This strategic selection ensures the PINNs model's accurate flow pattern predictions while adhering to foundational fluid dynamics principles.

3.1.4 PINNs Model Workflow

The workflow of a PINNs model for turbulence modeling using RANS equations encompasses:

1. **Inputs:** Include physical parameters and non-dimensional numbers defining the turbulent flow and its properties. These inputs enable the consideration of essential fluid dynamics aspects in turbulence modeling.
2. **Neural Network:** A deep learning architecture with multiple hidden layers and advanced activation functions to capture the complex relationships in turbulence.
3. **Outputs:** Represent critical variables of turbulent flow such as state, energy characteristics, and flow dynamics. The network aims to predict these based on the inputs.
4. **Automatic Differentiation:** Facilitates the integration of fluid dynamics equations into the learning process by efficiently computing derivatives, essential for applying Navier-Stokes equations.
5. **Governing Equations:** Incorporation of RANS equations into the loss function ensures predictions adhere to conservation laws of fluid motion, grounding the learning in physics.
6. **Loss Function:** A composite of data fidelity (minimising prediction data differences) and physics fidelity (ensuring RANS equations compliance), balancing data exploitation with physical constraints adherence.
7. **Optimizer:** Adjusts neural network parameters to minimise loss, using algorithms like Adam or SGD, essential for converging towards physically meaningful solutions.

This approach effectively integrates comprehensive neural network architectures, automatic differentiation, and physical laws, ensuring accurate and physically consistent turbulence predictions, as illustrated in diagram 1.

3.1.5 Loss Function Design

The loss function for the PINNs model encompasses two primary components: the physics-informed loss and the data loss.

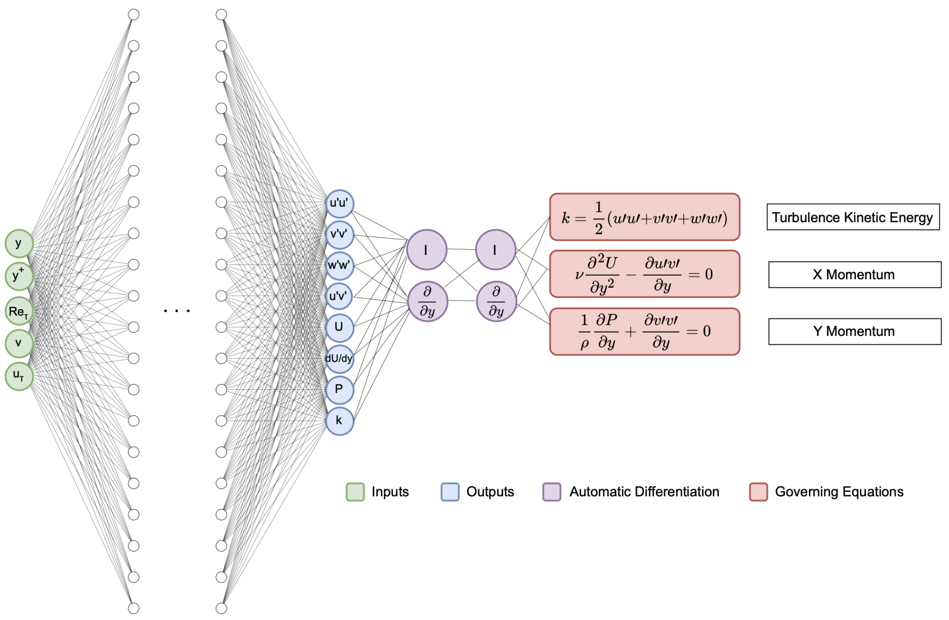

y
y+
Re_τ
ν
u_τ
. . .
u'u'
v'v'
w'w'
u'v'
U
dU/dy
P
k
I
∂/∂y
$k = \frac{1}{2}(u'u' + v'v' + w'w')$
$\nu \frac{\partial^2 U}{\partial y^2} - \frac{\partial u'v'}{\partial y} = 0$
$\frac{1}{\rho}\frac{\partial P}{\partial y} + \frac{\partial v'v'}{\partial y} = 0$
Turbulence Kinetic Energy
X Momentum
Y Momentum
Inputs
Outputs
Automatic Differentiation
Governing Equations

Fig. 1: PINNs Model Workflow Diagram

Physic Loss Function
The physics-informed loss, derived from the deviation from the Navier-Stokes equations, includes:

- **Momentum loss:**

$$\begin{aligned}\mathcal{L}_{\text{momentum}} &= \mathcal{L}_{\text{x-momentum}} + \mathcal{L}_{\text{y-momentum}} \\ \mathcal{L}_{\text{x-momentum}} &= \left\| \nu \frac{\partial^2 U}{\partial y^2} - \frac{\partial u'v'}{\partial y} \right\|_2 \\ \mathcal{L}_{\text{y-momentum}} &= \left\| \frac{1}{\rho}\frac{\partial P}{\partial y} + \frac{\partial v'v'}{\partial y} \right\|_2\end{aligned} \tag{1}$$

- **Turbulence kinetic energy (TKE) consistency loss:**

$$\mathcal{L}_{\text{TKE}} = \left\| k_{\text{pred}} - \frac{1}{2}(u'u'_{\text{pred}} + v'v'_{\text{pred}} + w'w'_{\text{pred}}) \right\|_2 \tag{2}$$

Combining these terms, the overall physics-informed loss is:

$$\mathcal{L}_{\text{physics}} = w_{\text{momentum}}\mathcal{L}_{\text{momentum}} + w_{\text{TKE}}\mathcal{L}_{\text{TKE}} \tag{3}$$

Data Loss Function
The data loss, quantifying the model's prediction accuracy, includes:

- **Velocity magnitude loss:**

$$\mathcal{L}_{\text{U}} = \|U_{\text{pred}} - U_{\text{data}}\|_2 \tag{4}$$

- **Velocity gradient loss:**

$$\mathcal{L}_{\partial U/\partial y} = \left\| \frac{\partial U}{\partial y}_{\text{pred}} - \frac{\partial U}{\partial y}_{\text{data}} \right\|_2 \tag{5}$$

- **Pressure loss:**

$$\mathcal{L}_P = \|P_{\text{pred}} - P_{\text{data}}\|_2 \tag{6}$$

- **TKE loss:**

$$\mathcal{L}_k = \|k_{\text{pred}} - k_{\text{data}}\|_2 \tag{7}$$

- **Reynolds stress loss:**

$$\mathcal{L}_{\text{stress}} = \left\| \begin{bmatrix} uu_{\text{pred}} \\ v'v'_{\text{pred}} \\ w'w'_{\text{pred}} \\ u'v'_{\text{pred}} \end{bmatrix} - \begin{bmatrix} u'u'_{\text{data}} \\ v'v'_{\text{data}} \\ w'w'_{\text{data}} \\ u'v'_{\text{data}} \end{bmatrix} \right\|_2 \tag{8}$$

The total data loss is a weighted sum of these components:

$$\mathcal{L}_{\text{data}} = w_{\text{U}}\mathcal{L}_{\text{U}} + w_{\partial U/\partial y}\mathcal{L}_{\partial U/\partial y} + w_P\mathcal{L}_P + w_k\mathcal{L}_k + w_{\text{stress}}\mathcal{L}_{\text{stress}} \tag{9}$$

Finally, the comprehensive loss function for the PINNs model is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{physics}} + \mathcal{L}_{\text{data}} \tag{10}$$

3.1.6 Model Architecture

The architecture of PINNs is a critical aspect where a balance between the depth and width of the network, type of activation functions, and learning strategies is crucial.

Layers and Nodes
The choice of the optimal depth (number of layers) and width (number of neurons per layer) of the network is fundamental. The architecture must be complex enough to model the complex patterns and relationships intrinsic to the physical laws and data sets involved, but not so complex that it becomes intractable or subject to overfitting.
A shallow but extensive network architecture is often preferred for PINNs for several reasons. Firstly, it mitigates the risk of gradients disappearing or exploding, a common challenge in deep neural network learning, particularly critical when differentiating across the network to apply the laws of physics. Secondly, wide networks are able to capture a broad spectrum of features or interactions in the input space with fewer parametric layers, which matches the continuous nature of many physical problems [1].

Activation Functions
The choice of activation function influences the network's ability to model complex, nonlinear phenomena and its differentiability—a key requirement in PINNs, where the model's output and its derivatives must adhere to specified differential equations. Activation functions with smooth, non-saturating gradients are favored to facilitate efficient learning and accurate physical representation.

Optimizer and Learning Rate Scheduler Configuration
The learning rate and optimisation strategy must be carefully calibrated to balance rapid convergence and stability of the learning process. For this model, a two-tier strategy incorporating a learning rate optimiser and a learning rate scheduler is used to improve the efficiency of the training process. This approach allows adaptive adjustments of the learning rate as the training progresses, providing optimised convergence properties.

- **Optimizer Selection**: The Adam optimizer is selected for its adaptive learning rate capabilities, which fine-tune the learning process by adjusting the learning rate for each parameter. Adam combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).
- **Learning Rate Scheduler**: To further refine the training dynamics, we integrate a learning rate scheduler that adjusts the learning rate based on the model's performance on the validation set. The *ReduceLROnPlateau* scheduler is employed, decreasing the learning rate when a metric has stopped improving, aiming to find fine-grained optima by slow convergence when plateauing. This scheduler reduces the learning rate by a factor of 0.9 after a *patience* of 100 epochs without improvement in the monitored metric, specifically the validation Mean Squared Error (`val_mse`), enhancing the ability to converge to the optimal solution with refined learning rate adjustments.

Balancing Loss Terms
PINNs typically comprise multiple loss components, including terms that enforce adherence to the governing physical laws (physical loss) and, when applicable, terms that measure the discrepancy between the network's predictions and observed data (data loss). Properly balancing these components is essential for ensuring that the network does not disproportionately prioritise one type of loss over the other, potentially at the expense of the model's overall accuracy and generalisability.

Hyperparameter Tuning
The selection of hyperparameters, including activation functions, learning rate, and the balance between physical loss and data loss weights, is a delicate process that significantly impacts the performance of PINNs. We employed a grid search strategy to fine-tune the following hyperparameters:

- **Activation Functions:** Exponential Linear Unit (ELU), Scaled Exponential Linear Unit (SELU), and Hyperbolic Tangent (TANH) were considered, given their unique properties beneficial for maintaining smooth gradients essential for PINNs.

- **Learning Rates (LR):** A range of learning rates, 1×10^{-3}, 1×10^{-4}, 5×10^{-4}, were tested to find the optimal speed at which the model converges without overlooking significant features of the solution space.
- **Neural Network Size:** Both the number of layers (2, 3, 4, 8) and the number of nodes per layer (32, 64, 128, 256) were varied to ascertain the architecture's impact on the model's performance.
- **Weight of the Physical Loss and Data Loss:** The trade-off between physical law compliance (with weights 1, 10, 100) and fitting to the data (weights 1, 0.1, 0.01) was calibrated to ensure the model not only learns the underlying physics but also adheres closely to the empirical data.

3.1.7 Model Implementation

The source code for the PINNs model is available in the following GitHub repository: PINNs Repository.

Framework (PyTorch Lightning)
The PINNs model was developed using the PyTorch Lightning Framework. This is an AI framework that builds on PyTorch to provide a more organised code structure and reduce the amount of standard code required. It clearly separates model learning logic and automation code, making code easier to develop, debug and reuse. PyTorch Lightning supports advanced features such as multi-GPU, TPU, distributed training and mixed precision, making the training process more efficient and scalable.

Core Components
Here are the different elements that make up the PINNs model:

- **NNModel:** This component represents the backbone of the Physics-Informed Neural Network. It is a customisable neural network architecture that can be tailored according to the specific requirements of the turbulence modeling task at hand. Parameters such as the dimensions of the input and output layers, the size and number of hidden layers, and the types of activation functions can be adjusted. This flexibility ensures that the network is capable of capturing the complex dynamics of turbulent flows, facilitating accurate predictions of the various turbulence characteristics.
- **TurbulenceDataset:** Specialised class that handles the loading, pre-processing and formatting of turbulence data. It structures the data in a way that is compatible with machine learning models, encompassing features such as spatial coordinates, Reynolds number and other relevant inputs, as well as target outputs such as velocity components and kinetic energy. This dataset provides the basic data needed to train and evaluate the PINN model, ensuring that the network has access to high-quality information that reflects the physical system it aims to model.
- **TurbulenceDataModule:** This component acts as a data management hub within the PyTorch Lightning framework, orchestrating the flow of data through different phases of the model lifecycle, including training, validation, and testing. The TurbulenceDataModule leverages the TurbulenceDataset to efficiently batch and shuffle data. It ensures that data is fed into the model in a structured and consistent manner, optimising the learning process and aiding in the model's convergence towards accurate predictions.
- **TurbulenceModelPINN:** Core component that integrates the neural network model (NNModel) with physics-informed learning. It goes beyond conventional neural network applications by incorporating the physics of turbulence directly into the learning process, using a physics-informed loss function. This function penalises the model not only for deviations from the training data, but also for violations of the physical laws governing it, such as the Navier-Stokes equations. It thus guides the PINN model to develop solutions that are both accurate and physically plausible. The TurbulencePINN model therefore represents the culmination of the physics-informed neural network approach, integrating both data-based knowledge and the rigour of physical laws into the model's predictions.

3.1.8 Training Pipeline

Dataset Segmentation

- **Training Set (68%):** For basic training, allowing model learning and adaptation.
- **Validation Set (12%):** Acts as a performance checkpoint for model tuning.
- **Test Set (20%):** Provides an unbiased final evaluation of the model.

Training Cycles
Training involves several cycles with cumulative learning:

- **Cycle 1:** Initial rapid learning with a Learning Rate (LR) of $1e^{-3}$ and equal focus on empirical data fit and physical law adherence.
- **Cycle 2:** Refinement with a reduced LR of $1e^{-4}$, maintaining the balance between empirical data learning and physical constraints.
- **Cycle 3 & 4:** Further tuning with LR of $1e^{-4}$, with increasing emphasis on physical law adherence (Cycle 3: 10/1, Cycle 4: 100/1 ratio).

Ensuring Reproducibility
Key practices for reproducibility:

- **Fixed Random Seeds:** For consistent stochastic processes.
- **Version Control:** For code, data, and detailing the entire pipeline.
- **Explicit Documentation:** Of hyperparameters, training configurations, and environment setups.
- **Environment Management:** Specifying OS, programming versions, and dependencies.
- **Deterministic Algorithms:** Minimising variability.
- **Comprehensive Logging:** For training, validation metrics, and model checkpoints.
- **Detailed Reporting:** Methodology sections include thorough descriptions of research processes.

This approach ensures continuous model evolution, focusing on a balance between empirical accuracy and physical fidelity, resulting in a robust model for turbulence prediction. It also emphasises reproducibility, facilitating validation and verification of the training process and results.

3.2 PySINDy

The Boussinesq hypothesis simplifies the analysis of turbulence in fluid mechanics. However, this approximation leads to equations that include terms that are hard to measure directly, such as the Reynolds stress tensor τ.

As we know, PySINDy can discover underlying differential equations from complex data. By applying it to DNS data, the objective would be to derive a new system which models the relationship between the velocity U and the Reynolds stress tensor τ without relying on the Boussinesq hypothesis. Besides, PySINDy could identify dynamic relationships implicit in the DNS data that are not obvious at first glance. Once we have the PySINDy's equations, we will transform them in order to run simulations based on the new turbulence model. Therefore, we will compare this new model with existing models to validate whether or not the results obtained are similar.

Selection of the DNS variables
To implement a turbulence model with PySINDy that is able to be as accurate as traditional models such as DNS, it is necessary to select the variables that are important in the turbulent flow. From the DNS data, it was decided to select the following variables:

- y^+, the grid point in wall-normal direction;
- Re_τ, the Reynolds number;
- U, the mean profile of streamwise velocity;
- u'u', v'v', w'w' and u'v', the variances and covariances of velocity fluctuations;
- P, the mean profile of pressure;
- $\frac{dU}{dy}$, the derivative of U in wall-normal direction;

Data processing
To obtain a robust PySINDy model, some processing of the DNS data associated with the selected variables may be necessary. Firstly, it is important to use the linear interpolation method to our data. After all, looking at the dataset, the data are not measured at regular intervals with respect to y^+ and the variations between the data points appear to be effectively linear. Consequently, linear interpolation will allow us to fill in the missing observations at certain intervals and may also allow us to increase on the number of data points that can be utilised for training and testing the model. From a general point of view, interpolation will help to increase the reliability of our data and will allow us to capture the behaviour of turbulence more accurately. Moreover, in order to enrich the PySINDy model and to improve its accuracy, it was decided to use numerical differentiation using the gradient method in order to compute the first and the second derivatives of the selected variables. To compute the derivatives, we used the gradient method from the Python library Numpy.

Data partitioning
We investigated two different approaches to splitting our DNS data into training and test datasets: For the first approach, we used the train_test_split method available in the Scikit-Learn Python library to split our data. However, we found out that the splitting performed by this method is random, which may alter the sequential structure inherent in time series. For the second approach, we used a division based on Reynolds number, allocating data for a specific Reynolds number to the training dataset and data corresponding to any other Reynolds number to the test dataset. This method keeps the temporal integrity of the time series data, an essential condition for the coherent application of PySINDy and for capturing physical dynamics. It was finally decided to apply the second approach for training and testing the PySINDy model as it looked the most appropriate to be used in the development of a PySINDy model.

Selection of hyperparameters (Feature_Library and parameters in optimizer)
In order to optimise the PySINDy model and make it more robust, we implemented our own version of the GridSearch algorithm to select the most effective hyperparameters in order to obtain the most accurate model possible. Therefore, the application of this method will let us test a predefined set of values for a range of hyperparameters. In PySINDy, the most relevant hyperparameters

are the Feature_Library and the parameters of the optimizer. With our PySINDy model, we will just use the GridSearch algorithm in order to select the optimal Feature_Library from the following set:

- PolynomialLibrary of degree 1;
- PolynomialLibrary of degree 2;
- FourierLibrary;

Due to an iterative process, this approach guarantees both the robustness and the fine-tuning of the method to our DNS data. For the optimizer, the default one will be employed: Sequential Threshold Least Squares (STLSQ). This optimizer has two hyperparameters: the threshold and the alpha coefficient. The GridSearch algorithm will not be used to select the value of these two hyperparameters because they are both extremely sensitive to the data used and it is therefore better at this stage to adjust them manually in order to obtain the best combination of parameters based on the training data provided to the model.

Training, prediction and evaluation of the PySINDy model

To train the PySINDy model, we used the training dataset obtained after data splitting and the hyperparameters obtained with the GridSearch algorithm. In the following, you will find the set of features variables used for the PySINDy model:

- U, u'u', v'v', w'w', u'v', P, $\frac{dU}{dy}$, $\frac{du'u'}{dy}$, $\frac{dv'v'}{dy}$, $\frac{dw'w'}{dy}$, $\frac{du'v'}{dy}$ and $\frac{dP}{dy}$

Here's the set of target variables used for the PySINDy model:

- $\frac{dU}{dy}$, $\frac{du'u'}{dy}$, $\frac{dv'v'}{dy}$, $\frac{dw'w'}{dy}$, $\frac{du'v'}{dy}$, $\frac{dP}{dy}$, $\frac{d^2U}{dy^2}$, $\frac{d^2u'u'}{dy^2}$, $\frac{d^2v'v'}{dy^2}$, $\frac{d^2w'w'}{dy^2}$, $\frac{d^2u'v'}{dy^2}$ and $\frac{d^2P}{dy^2}$

Training the PySINDy model will generate a set of equations. These PySINDy equations will be used to simulate the PySINDy model by using the integration method (solve_ivp) available in the scipy.integrate package on the PySINDy equations. With this simulation, we are able to predict the values for the velocity U and the variances and covariances of the velocity fluctuations (u'u', v'v', w'w', u'v'). For the evaluation of the model's performance and robustness, the focus remains on metrics such as the coefficient of determination (R^2) and mean square error (MSE). Besides, in order to improve our analysis of the simulation's validity, it was decided to also use the Mean Absolute Error (MAE) metric. Our decision is therefore based on MAE's capability to deliver a clear measure of the average absolute error between our predictions and the actual values

4 Results

4.1 PINNs Model

It is important to note that the following results can be reproduced using seed '123' and the batch size of 32 training samples per step.

4.1.1 Hyper-parameters Tuning

During the hyper-parameter tuning process, we explored the influence of model size, learning rate, and activation functions on the performance of our PINN. Table 2 showcases the meticulous selection process for the ideal model size alongside the final choice of the ELU as the activation function, revealing a minimum total MSE on the test dataset when employing a neural network configuration of 4×128. Similarly, in our examination of the optimal learning rate, detailed in Table 1, the learning rate of 5×10^{-4} emerged as the most effective, further substantiating the findings from our model size exploration. The impact of the activation function on model performance is demonstrated in Table 3, where ELU outperformed other tested activation functions in terms of total MSE on the test dataset. Furthermore, the balancing of physical and data loss weights was critically evaluated, as summarised in Table 4, pinpointing an equal weighting strategy as most conducive to minimising the total MSE.

Tab. 1: Selection of the best initial learning rate on a 4-layer model with 128 nodes/layer using the total MSE on the test dataset.

Batch Size	NN Size	Learning Rate	Epoch	Activation Func.	Total MSE
32	4×128	1×10^{-3}	1000	ELU	0.001421
32	$\mathbf{4 \times 128}$	$\mathbf{5 \times 10^{-4}}$	**1000**	**ELU**	**0.000554**
32	4×128	1×10^{-4}	1000	ELU	0.000719

Tab. 2: Selection of the best model size with the final activation function (ELU) using the total MSE on the test dataset.

Batch Size	NN Size	Learning Rate	Epoch	Activation Func.	Total MSE
32	2×32	5×10^{-4}	1000	ELU	0.173699
32	3×32	5×10^{-4}	1000	ELU	0.039388
32	4×32	5×10^{-4}	1000	ELU	0.010825
32	8×32	5×10^{-4}	1000	ELU	0.002820
32	2×64	5×10^{-4}	1000	ELU	0.107276
32	3×64	5×10^{-4}	1000	ELU	0.016589
32	4×64	5×10^{-4}	1000	ELU	0.004392
32	8×64	5×10^{-4}	1000	ELU	0.001045
32	2×128	5×10^{-4}	1000	ELU	0.484129
32	3×128	5×10^{-4}	1000	ELU	0.110587
32	$\mathbf{4 \times 128}$	$\mathbf{5 \times 10^{-4}}$	**1000**	**ELU**	**0.000554**
32	8×128	5×10^{-4}	1000	ELU	0.000778
32	2×256	5×10^{-4}	1000	ELU	0.053500
32	3×256	5×10^{-4}	1000	ELU	0.001884
32	4×256	5×10^{-4}	1000	ELU	0.000616
32	8×256	5×10^{-4}	1000	ELU	0.015669

Tab. 3: Selection of the best activation function on a 4-layer model with 128 nodes/layer using the total MSE on the test dataset.

Batch Size	NN Size	Learning Rate	Epoch	Activation Func.	Total MSE
32	$\mathbf{4 \times 128}$	$\mathbf{5 \times 10^{-4}}$	**1000**	**ELU**	**0.000554**
32	4×128	5×10^{-4}	1000	SELU	0.001051
32	4×128	5×10^{-4}	1000	TANH	0.204692

Tab. 4: Initial weights selection for physic and data losses in a 4-layer, 128 nodes/layer model based on total MSE on the test dataset.

Batch Size	NN Size	Learning Rate	Epoch	Activation Func.	w_{phys}	w_{data}	Total MSE
32	$\mathbf{4 \times 128}$	$\mathbf{5 \times 10^{-4}}$	**1000**	**ELU**	**1**	**1**	**0.000551**
32	4×128	5×10^{-4}	1000	ELU	1	1×10^{-1}	0.009988
32	4×128	5×10^{-4}	1000	ELU	1	1×10^{-2}	0.002866
32	4×128	5×10^{-4}	1000	ELU	1×10^{1}	1	0.007187
32	4×128	5×10^{-4}	1000	ELU	1×10^{1}	1×10^{-1}	0.026153
32	4×128	5×10^{-4}	1000	ELU	1×10^{1}	1×10^{-2}	0.015185
32	4×128	5×10^{-4}	1000	ELU	1×10^{2}	1	0.002753
32	4×128	5×10^{-4}	1000	ELU	1×10^{2}	1×10^{-1}	0.044795
32	4×128	5×10^{-4}	1000	ELU	1×10^{2}	1×10^{-2}	0.602007

4.1.2 Training Step

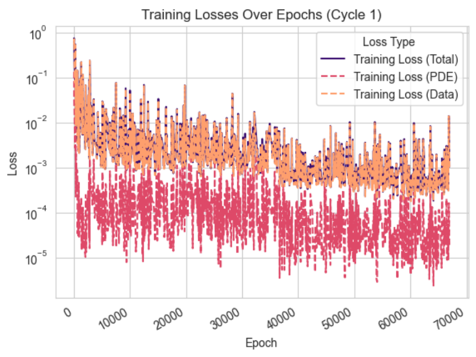
Figure 2 illustrates the training losses, including the total training loss, the loss from the PDE, and the loss from the data over multiple epochs for four successive cycles of the PINN model.

For Cycle 1 (Figure 2a), as the training progresses, the total loss and PDE loss decrease gradually, but the data loss remains higher, suggesting that the model is initially struggling to fit the data accurately. The PDE loss is lower than the data loss, implying that the model is prioritising the satisfaction of the governing equations over data fitting in this cycle.

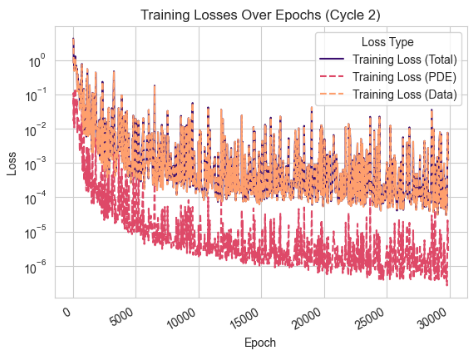
In Cycle 2 (Figure 2b), the data loss, remains high initially but decreases as the training progresses. The model is focusing more on fitting the data while maintaining the ability to satisfy the governing equations learned in the previous cycle. The PDE loss and data loss converge to similar levels towards the end of the cycle, indicating a balance between the two objectives.

Cycle 3 (Figure 2c) shows a more consistent decrease in all losses throughout the training process. The total loss, PDE loss, and data loss all start at lower levels compared to the previous cycles, indicating that the model has improved its overall performance. The model is still prioritising the satisfaction of the governing equations over data fitting, but the gap between the two losses is smaller than in previous cycles.

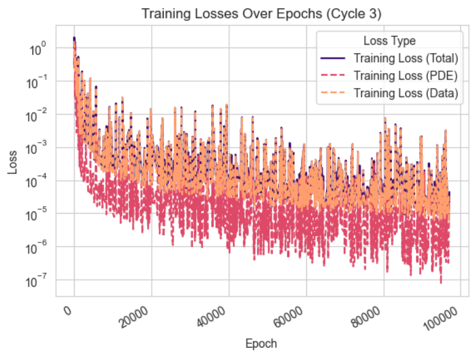
In the final cycle (Figure 2d), all losses continue to decrease and converge to relatively low values. The total loss, PDE loss, and data loss are closely aligned, suggesting that the model has achieved a good balance between satisfying the governing equations and fitting the data. The convergence of the losses indicates that the model has effectively learned the underlying physics and can accurately represent the data while adhering to the physics-based constraints.



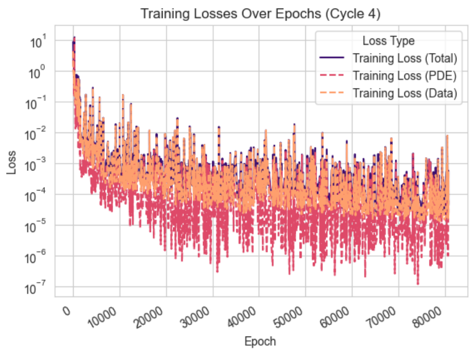
(a) Cycle 1



(b) Cycle 2



(c) Cycle 3



(d) Cycle 4

Fig. 2: Training Losses Over Epochs for Four Cycles

Figure 3 illustrates the training and validation losses over multiple epochs for four successive cycles of the PINN model.

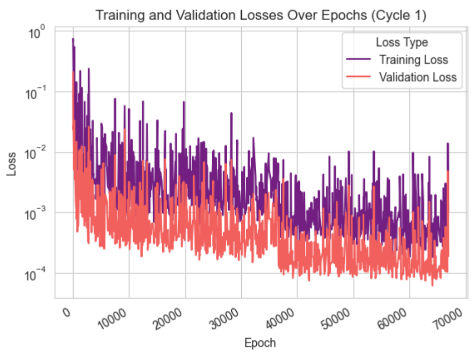
The first subfigure 3a shows the training and validation losses over epochs for cycle 1. Initially, both losses are high, indicating that the model has not learned the underlying patterns and relationships in the data. As learning progresses, the learning and validation losses decrease. The difference between the two losses indicates a certain level of over-fitting.

In Cycle 2 (Figure 3b), training and validation losses start at a lower level than in Cycle 1, implying that the model has retained some knowledge from the previous cycle. Both losses show a downward trend, but the validation loss remains higher than the learning loss throughout the cycle.

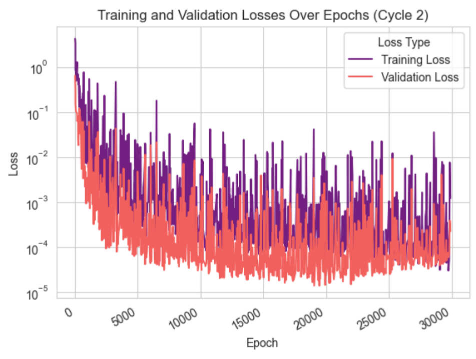
The Cycle 3 (Figure 3c) shows a more promising trend. While training and validation losses start at relatively high values, they decrease significantly as training progresses. Notably, the validation loss closely follows the training loss, with a smaller gap compared to previous cycles. This suggests that the model achieves better generalisation performance and is less prone to overfitting. This result is due to the increased weight of PDE loss, as shown in the figure 2c.

In the last cycle, cycle 4 (Figure 3d), the training and validation losses continue to decrease, reaching lower values than in the previous cycles. The gap between the two losses narrows further, indicating that the model has improved its ability to generalise to the validation set. Indeed, the weight of the PDE loss was again increased. The validation loss closely follows the learning loss, suggesting that the model has achieved a good balance between adaptation to the learning data and generalisation to the unseen data.
The total training time took 10.3 hours. Here is how long it took to obtain each of the best snapshots of each cycle:

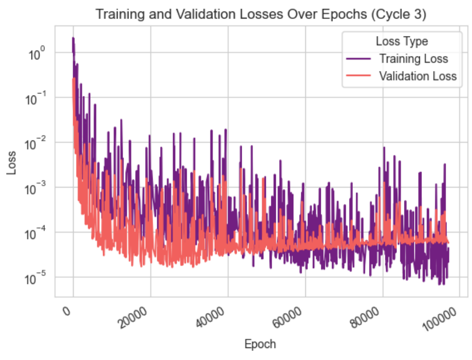
- **Cycle 1:** 3.5 hours (58886 epoch)
- **Cycle 2:** 2.7 hours (16290 epoch)
- **Cycle 3:** 1.0 hour (20332 epoch)
- **Cycle 4:** 3.1 hours (54851 epoch)



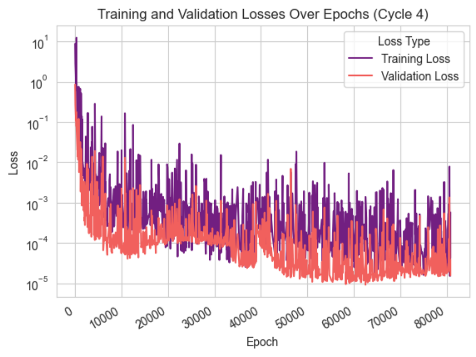
(a) Cycle 1



(b) Cycle 2



(c) Cycle 3



(d) Cycle 4

Fig. 3: Training & Validation Loss Over Epochs for Four Cycles

4.1.3 Test Step

Best Trained Model Selection
After training, the best snapshot from each cycle was used to perform inference over the test dataset. As shown in Figure 4, the best model is the one from Cycle 3, which is why we decided to use it for the following analysis.

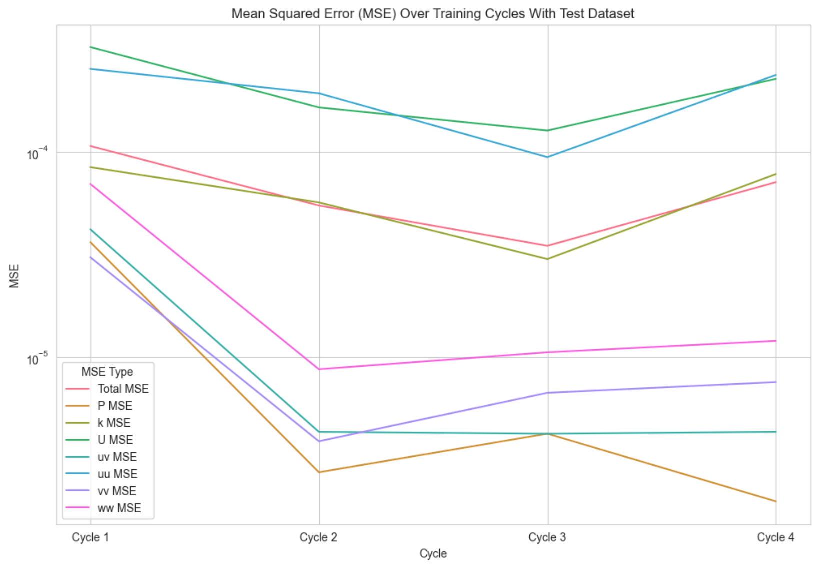


Fig. 4: MSE Over Training Cycles With Test Dataset

Best Trained Model Inference Plot
As shown in the figure 5, the trained PINN Model (Cycle 3) perfectly captures the dynamics of turbulent flow in the boundary layer.

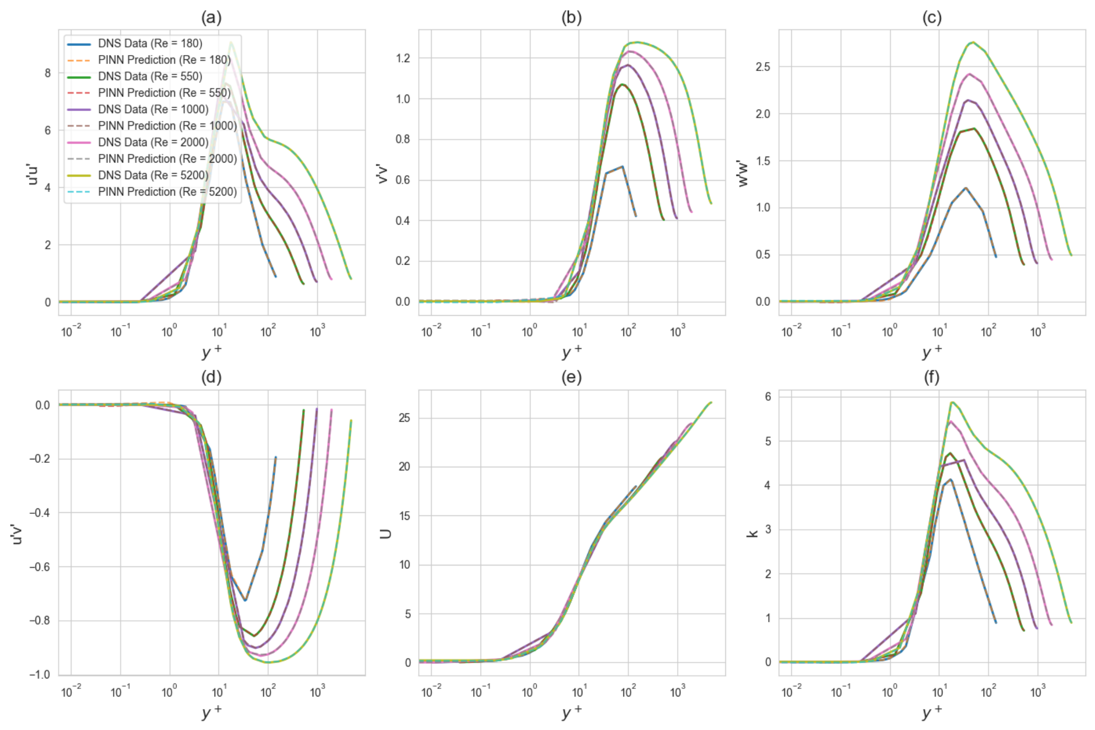


Fig. 5: Comparison of PINNs predictions and DNS ($Re \in (180, 550, 1000, 2000, 5200)$)

Best Trained Model Relative ℓ_2-norm of Errors Analysis
To evaluate the accuracy of the PINNs model predictions, we consider the relative ℓ_2-norm of errors E_i on all the computational points and for the ith variable as [5]:

$$E_i = \frac{\left\| U_i - \widetilde{U_i} \right\|_2}{\|U_i\|_2} \times 100 \tag{11}$$

where $\|.\|_2$ denotes ℓ_2-norm, U the vectors of reference data, and $\widetilde{U}$ indicate the vectors PINNs predictions. Table 5 details the relative ℓ_2-norm of errors for various quantities over different Reynolds numbers. Observations from the table indicate several key points:

- The relative errors generally decrease with the increase in Reynolds numbers, suggesting that PINNs potentially have improved prediction accuracy for higher flow rates. For example, the error in velocity prediction (E_u) noticeably decreases from 0.26% at Re=180 to 0.03% at both Re=2000 and Re=1000.
- Different quantities exhibit varying degrees of error, with turbulent properties (e.g., $E_{u'u'}$, $E_{v'v'}$) presenting higher errors compared to scalar quantities like velocity (E_u) and temperature (E_θ), highlighting the challenges in accurately predicting turbulent flows.
- A consistent trend in the reduction of error percentages across most quantities as Reynolds numbers increase, showcases the adaptability and reliability of the PINN approach in a range of fluid dynamic conditions.

Tab. 5: Relative ℓ_2-norm of errors, in the predictions using PINNs over the Test Dataset

Re	E_u	E_θ	E_k	$E_{u'u'}$	$E_{v'v'}$	$E_{w'w'}$	$E_{u'v'}$
180	0.26%	0.79%	0.57%	0.57%	1.19%	0.73%	1.12%
550	0.10%	0.34%	0.39%	0.52%	0.39%	0.51%	0.46%
1000	0.03%	0.33%	0.13%	0.18%	0.35%	0.20%	0.32%
2000	0.03%	0.21%	0.11%	0.18%	0.32%	0.17%	0.33%
5200	0.04%	0.17%	0.10%	0.16%	0.21%	0.15%	0.20%

Best Trained Model Mean Square Error Analysis
The MSE data provided in Table 6 underscore the model's performance in quantitative terms across different predictions:

- The Total MSE of 3.48×10^{-5} represents the overall prediction accuracy of the PINN model, indicating a generally strong performance across all tested metrics.

- Analysis of MSE for specific quantities reveals high accuracy in pressure prediction (MSE P = 4.23×10^{-6}) and challenges in accurately predicting velocity (MSE U = 1.27×10^{-4}), highlighting areas where PINNs perform well and where improvements might be beneficial.
- The low MSE values for velocity gradient (MSE dUdy = 1.81×10^{-6}) and turbulent kinetic energy (MSE k = 3.00×10^{-5}) further demonstrate the capability of PINNs in capturing crucial aspects of fluid motion and turbulence.
- Among the turbulent quantities, the higher MSE for uu (MSE uu = 9.42×10^{-5}) suggests predicting turbulent stresses, particularly in the streamwise direction, poses a significant challenge, indicating a potential area for focused improvement in modeling turbulent flows.

Tab. 6: MSE over the Test Dataset

Test Metric	Value	
Total MSE	3.48	$\times 10^{-5}$
MSE P	4.23	$\times 10^{-6}$
MSE U	1.27	$\times 10^{-4}$
MSE dUdy	1.81	$\times 10^{-6}$
MSE k	3.00	$\times 10^{-5}$
MSE uu	9.42	$\times 10^{-5}$
MSE vv	6.69	$\times 10^{-6}$
MSE ww	1.05	$\times 10^{-5}$
MSE uv	4.23	$\times 10^{-6}$

4.1.4 Overall Conclusion

The collective analysis of the relative ℓ_2-norm of errors and MSE values from Tables 5 and 6 reflects the efficacy of PINNs in accurately modeling fluid dynamics. The results showcase the promise of PINNs, especially at higher flow rates, while also highlighting the need for enhanced modeling strategies to tackle the intricacies of turbulent flow predictions more effectively. Through targeted enhancements, there is potential for PINNs to achieve even greater accuracy, making them a powerful tool for simulating and understanding complex fluid dynamics systems.

4.2 PySINDy

4.2.1 Interpolation Method

The results presented below from our PySINDy model were generated by analysing DNS data obtained from Oden, using the variables described in the methodology section for a set of Reynolds numbers (5200, 2000, 1000, 550, 180). Figure 6 shows several graphs describing velocity profiles, as well as the variances or covariances of velocity fluctuations, as a function of y^+.

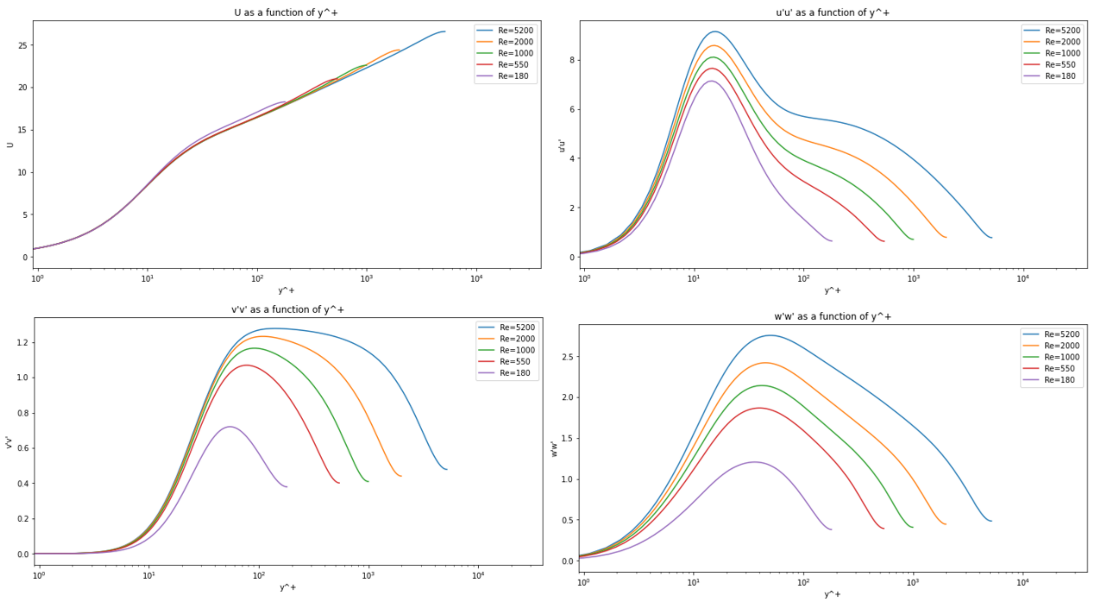


Fig. 6: Representation of velocity, covariances and variances of velocity fluctuations as a function of y^+ for different Reynolds numbers for DNS data (before interpolation)

Building on the methods described in the methodology section 3.2, we used the interpolation method to enrich our data set, generating a complete set of 10,000 data points. This enriched data set enabled a more detailed and continuous representation of turbulent flow characteristics. The following graphs, which we present here, allow a visual comparison of velocity and velocity fluctuations as a function of y^+ before and after the interpolation process.

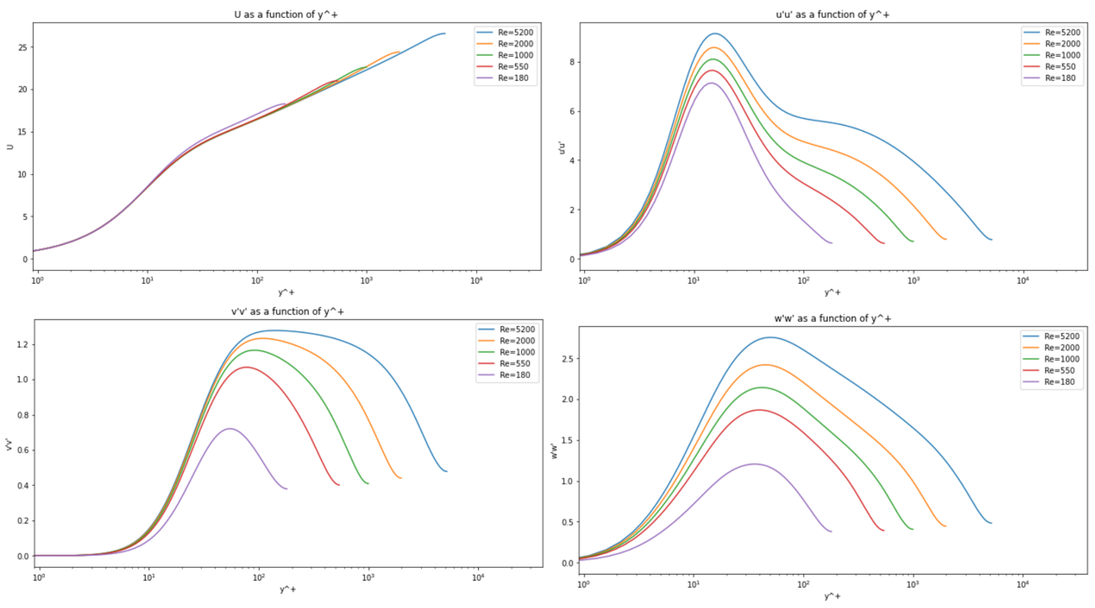


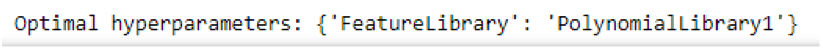
Fig. 7: Representation of velocity, covariances and variances of velocity fluctuations as a function of y^+ for different Reynolds numbers for DNS data (after interpolation)

Figures 6 and 7 show that the representation of velocity and velocity fluctuations as a function of y^+ is similar before and after interpolation. Consequently, these graphs not only validate the interpolation method, but also provide a better understanding of the fluid dynamics involved at different turbulence scales, as indicated by the range of Reynolds numbers.

4.2.2 Output Equations

In this results section, we adopted the strategy of using the data corresponding to the highest Reynolds number available, 5200, for training, while utilising the data associated with a Reynolds number of 2000 for testing. The decision is based on the fact that higher Reynolds numbers can capture more complex turbulence behaviour.

The application of the GridSearch algorithm as mentioned in the methodology (section 3.2) enabled us to identify the optimal feature_library as the PolynomialLibrary of degree 1, as illutrated in figure 8.

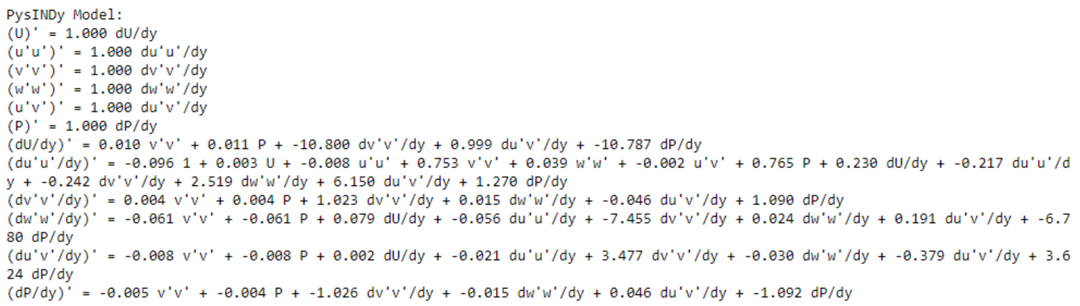


```
Optimal hyperparameters: {'FeatureLibrary': 'PolynomialLibrary1'}
```

Fig. 8: Optimal hyperparameter obtained through the grid search algorithm

After an extensive manual testing phase for a range of threshold and alpha coefficient values, we have converged on the parameters alpha = 0.001 and threshold = 0.0001.

This study has implemented the PySINDy Model to develop an advanced turbulence model of turbulence model. PySINDy Model generated dynamic equations for each variable to represent the turbulent flow. These equations are fundamental as they encapsulate the interactions between the velocity components and the pressure gradients throughout the dynamic flow field. The equations are shown below.



```
PysINDy Model:
(U)' = 1.000 dU/dy
(u'u')' = 1.000 du'u'/dy
(v'v')' = 1.000 dv'v'/dy
(w'w')' = 1.000 dw'w'/dy
(u'v')' = 1.000 du'v'/dy
(P)' = 1.000 dP/dy
(dU/dy)' = 0.010 v'v' + 0.011 P + -10.800 dv'v'/dy + 0.999 du'v'/dy + -10.787 dP/dy
(du'u'/dy)' = -0.096 1 + 0.003 U + -0.008 u'u' + 0.753 v'v' + 0.039 w'w' + -0.002 u'v' + 0.765 P + 0.230 dU/dy + -0.217 du'u'/d
y + -0.242 dv'v'/dy + 2.519 dw'w'/dy + 6.150 du'v'/dy + 1.270 dP/dy
(dv'v'/dy)' = 0.004 v'v' + 0.004 P + 1.023 dv'v'/dy + 0.015 dw'w'/dy + -0.046 du'v'/dy + 1.090 dP/dy
(dw'w'/dy)' = -0.061 v'v' + -0.061 P + 0.079 dU/dy + -0.056 du'u'/dy + -7.455 dv'v'/dy + 0.024 dw'w'/dy + 0.191 du'v'/dy + -6.7
80 dP/dy
(du'v'/dy)' = -0.008 v'v' + -0.008 P + 0.002 dU/dy + -0.021 du'u'/dy + 3.477 dv'v'/dy + -0.030 dw'w'/dy + -0.379 du'v'/dy + 3.6
24 dP/dy
(dP/dy)' = -0.005 v'v' + -0.004 P + -1.026 dv'v'/dy + -0.015 dw'w'/dy + 0.046 du'v'/dy + -1.092 dP/dy
```

Fig. 9: Output equations generated by PySINDy

Notable terms in the equations, such as $\frac{dU}{dy}$ and $\frac{dP}{dy}$ represent the changes in velocity and pressure along the direction of fluid flow, which follow the physical principles of fluid dynamics. Such terms enrich our understanding of fluid behavior under different conditions.

4.2.3 Comparison with DNS Data

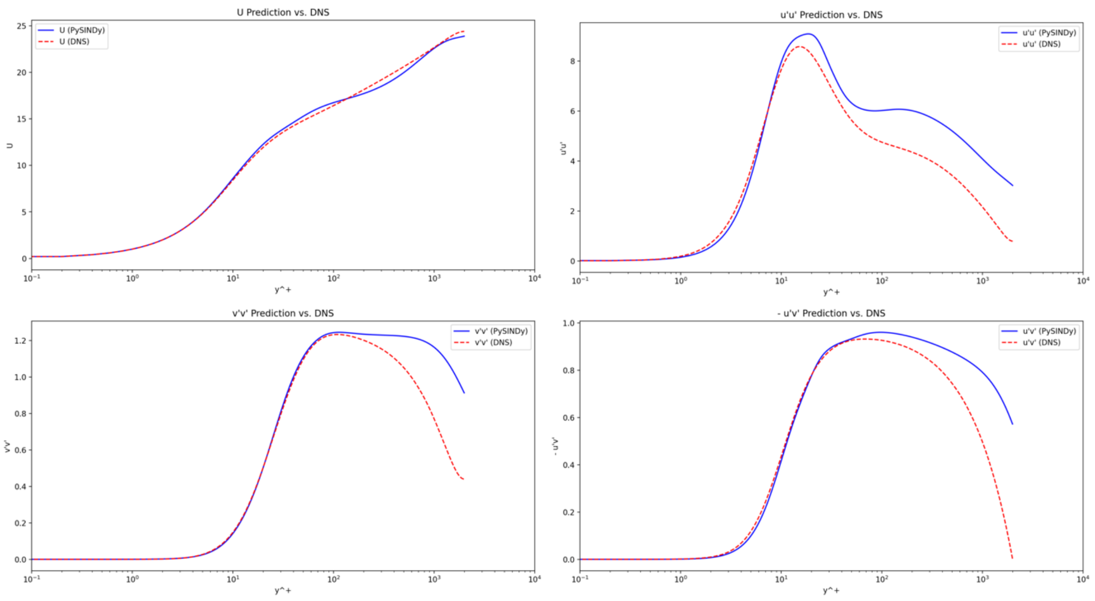


Fig. 10: Comparison of PySINDy predictions and DNS Data (Re = 2000)

Upon comparison with DNS data (Figure 10), our PySINDy model demonstrates a close correspondence, particularly for terms like U, u'u', v'v', w'w' and u'v' as functions of y^+, the normalised distance from the wall. This strong alignment highlights PySINDy's capability to capture essential characteristics of fluid behavior, especially in the critical areas near boundary layers where turbulence is most affected by the presence of the wall.

4.2.4 Improving PySINDy Model

After training our PySINDy model and generating this set of equations, we evaluated model performance and obtained an R^2 of 0.9068654365 and an MSE of 0.0000045599. At first glance, the scores obtained for the coefficient of determination and the MSE are excellent, indicating that the model is perfectly suited to the dataset used. However, in the case of PySINDy, although the R^2 value is already very high and the MSE is also very low, neither of these two values, on its own, allows us to conclude with certainty that the PySINDy model is effective in the specific field of fluid mechanics. From the figure above, we can clearly see that in the near-wall region (i.e. low y^+), PySINDy can accurately model the physics prevailing in this zone, but in the far-wall region (i.e. high y^+), the PySINDy model has some limitations with a clear deviation from the DNS data with regard to the variances and covariances of the velocity fluctuations, indicating that the model can still be improved.

To improve the PySINDy model's capability to predict turbulence, it will be necessary to refine the model in the future in order to take into account energy dynamics on larger scales. Indeed, as turbulence consists of a series of vortices and vortexes of varying sizes, we assume that it can predict these larger structures more accurately for realistic simulations by incorporating higher-order derivatives and nonlinear terms that reflect the energy cascades in turbulence.

Besides, we realise the necessity of a larger data set covering different flow conditions in order to enrich the model. By performing more high-fidelity experiments and direct numerical simulations, the PySINDy model will learn from a wider range of turbulence phenomena. At the end, it is important that the models offer better generalisation across a range of conditions, and deliver highly accurate predictions, in order to effectively meet the needs of industries that rely on accurate fluid dynamics modeling.
At the end, the PySINDy model which achieves such accuracy in predictions compared with DNS data highly suggests that PySINDy can be a useful tool in fluid dynamics research, providing an effective alternative to more traditional methods, without compromising the accuracy needed in order to capture complex turbulent behavior.

5 Discussions

We performed an in-depth and detailed comparison between the various models implemented and simulated, and the DNS data that will serve as a reference. In the context of turbulence modelling, the accuracy of different models in predicting turbulent flows is critical. Our comparative analysis is focused on evaluating the performance and accuracy of four different approaches: the

PINNs model, the PySINDy model, the k-ω turbulence model and the RSM.

Each model offers its unique perspectives and methodologies for capturing the complex turbulent flow patterns. By performing a comparison of these different models against DNS data, our objective is to evaluate their predictive accuracy and to identify their main strengths as well as their limitations. We have decided to compare them with DNS data, generally viewed as the standard for fluid simulations, as they offer an accurate reference for assessing the fidelity of these models. The graphs below illustrate the results of the predictions of each model compared with the DNS data.

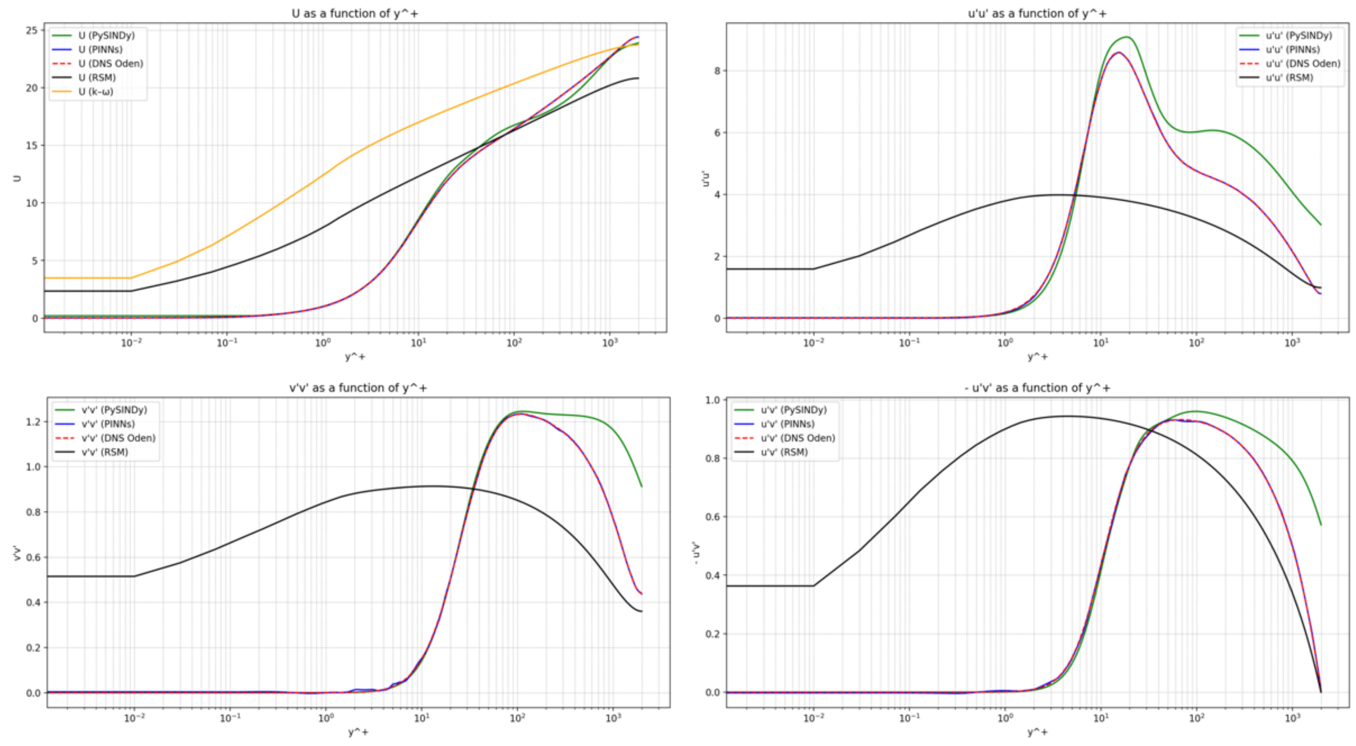


Fig. 11: Comparison between PINNs, PySINDy, k-ω, and RSM Model Simulations against DNS Data (Re = 2000)

1. **k-ω Model:** From the literature review, it is known that the k-ω model is known for its high accuracy in velocity prediction. As shown in the figure 11, the simulation of the model confirms this observation, with a prediction fairly close to the DNS data near the walls, where turbulence dynamics are strongest. Moreover, the model converges rapidly towards the maximum velocity profile, which demonstrates its effectiveness. However, the model has significant limitations in terms of the accuracy of its predictions of the covariances and variances of velocity fluctuations.

2. **RSM:** From the literature review, it is known that RSM is able to directly compute the covariances and variances of velocity fluctuations. As we can see in figure 11, for the different velocity covariances, RSM succeeds in reproducing the trend in DNS data, despite some deviations. Besides, RSM is known to be accurate in describing transitions from turbulent to laminar flow, making it particularly suitable for high-fidelity simulations requiring in-depth analysis. However, the model has considerable limits in terms of computational intensity, requiring more computational resources, making it difficult to use in large-scale applications.

3. **PySINDy Model and PINNs Model:** As shown in figure 11 , these two approaches have delivered very promising results which can rival traditional models such as DNS. Indeed, these two models not only provide an accurate analysis of fluid dynamics, but also greatly reduce computational costs in comparison with other traditional methods such as RSM. These models can provide an effective alternative for a wide range of turbulence simulation applications.

In summary, although traditional models such as k-ω and RSM are still quite accurate in specific scenarios, with the emergence of machine learning methods such as PySINDy or PINNs in turbulence modelling, we can expect to see a major evolution in this domain. Indeed, these innovative models are not only highly accurate, they also deliver other advantages in terms of relatively short execution times. As the CFD field is continuing evolving, these tools should play a key role in improving our understanding in the behavior of complex fluids.

6 Conclusions

6.1 Overall

This research underscores the transformative potential of machine learning in tackling the complexities of turbulence modeling through the use of PINNs and PySINDy. These ML techniques have demonstrated significant efficacy in addressing the closure problems of RANS equations, marking a substantial advancement in the field. The study involved the development and implementation of various machine learning models using Python, enhancing the practical application and usability of advanced ML models in real-world scenarios. Notably, the efficiency and accuracy of the PySINDy and PINNs models in managing complex simulations have surpassed traditional methods, setting a new standard and potentially revolutionizing how researchers and engineers approach turbulence modelling.

6.2 Future Works

- **Enhancing the versatility of PySINDy:**
  To broaden the application of PySINDy across diverse flow conditions, advanced regularisation techniques can be explored and implemented. These techniques will improve the robustness of the model, ensuring optimal performance under a variety of challenging conditions. Regularisation will also prevent overfitting, thereby improving the predictive accuracy of the model across extensive datasets.

- **Expanding Development for PINNs:**
  The development checklist for PINNs will be expanded to include more complex turbulence scenarios. This expansion will involve refining the algorithms and integrating additional physics-informed constraints to accurately present the complex dynamics of flow patterns. The goal is to broaden the scopes of PINNs the practical utility for simulating complex turbulent flows.

- **Integration within the ANSYS Environment:**
  To assess the practical viability and effectiveness of PINNs and PySINDy, efforts to integrate the ML models into ANSYS software environment are required. This integration will allow comprehensive simulations and direct comparisons with traditional models. By conducting side-by-side evaluations on this industry-standard platform, will allow us to determine which models yield the most reliable and accurate results in real-world engineering scenarios.

References

[1] R. Bischof and M. A. Kraus. Mixture-of-experts-ensemble meta-learning for physics-informed neural networks. *Forum Bauinformatik*, 2022.

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113. URL https://www.pnas.org/content/113/15/3932.

[3] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2020. doi: 10.1073/pnas.1911366116. URL https://www.pnas.org/content/116/45/22445.

[4] B. M. de Silva, K. Champion, M. Quade, J. Loiseau, J. N. Kutz, and S. L. Brunton. Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data. *Journal of Open Source Software*, 5(49):2104, 2020. doi: 10.21105/joss.02104. URL https://doi.org/10.21105/joss.02104.

[5] H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa. Physics-informed neural networks for solving reynolds-averaged navier–stokes equations. *Physics of Fluids*, 34(7):075117, 2022. doi: 10.1063/5.0095270. URL https://doi.org/10.1063/5.0095270.

[6] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[7] S. Luo, M. Vellakal, S. Koric, V. Kindratenko, and J. Cui. Parameter identification of rans turbulence model using physics-embedded neural network. pages 137–149, 2020.

[8] F. Pioch, J. H. Harmening, A. M. Müller, F.-J. Peitzmann, D. Schramm, and O. el Moctar. Turbulence modeling for physics-informed neural networks: Comparison of different rans models for the backward-facing step flow. *Fluids*, 8(2):43, 2023. doi: 10.3390/fluids8020043. URL https://doi.org/10.3390/fluids8020043.

[9] 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*, 378: 686–707, 2019.

[10] R. D. Sandberg, S. Hanrahan, M. Kozul, H. Jagode, H. Anzt, G. Juckeland, and H. Ltaief. Studying turbulent flows with physics-informed neural networks and sparse data. *International Journal of Heat and Fluid Flow*, 104:109232, 2023. doi: 10.1016/j.ijheatfluidflow.2023.109232. URL https://doi.org/10.1016/j.ijheatfluidflow.2023.109232.