Estimation of Turbulent Flow Using Physics-Informed Neural Networks (PINNs): Modeling of Velocity Fields Around Periodic Hills with PirateNET

Prepared by	:	**GÖKAY AKÇAY**
Student No	:	**090200147**
Submission Date	:	**October 10th, 2024**
Course	:	**FIZ 4902**
Supervisor	:	**PROF. DR. EMRE ONUR KAHYA**

Table of Contents

1. Introduction

Recent advancements in artificial intelligence and neural networks have facilitated the development of Physics-Informed Neural Networks (PINNs), an innovative framework that integrates physical laws into data-driven learning methodologies. PINNs combine the representational power of neural networks with governing physical principles, such as the Navier-Stokes equations in fluid dynamics, embedding these laws directly into the learning process. This approach enables the modeling of complex systems more effectively by guiding the network's training with physical constraints, reducing dependency on extensive datasets. The hybrid nature of PINNs addresses critical challenges in traditional computational methods, such as limited data availability and high computational costs, offering efficient solutions for problems like fluid dynamics simulations **[1-4]**, where computational demands are traditionally significant. By incorporating physical laws into their architecture, PINNs are capable of producing predictions that adhere to established physics, making them particularly adept at capturing intricate phenomena like turbulence with a high degree of accuracy.

This study investigates the application of PINNs in solving the Navier-Stokes equations for fluid dynamics. A pre-existing dataset generated through simulations will be utilized, including data from four turbulence models: k-ε, k-ε-ϕt-f, k-ω, and k-ω SST. The dataset **[5]** spans 29 distinct cases featuring diverse flow scenarios such as periodic hills, square ducts, parametric bumps, converging-diverging channels, and curved backward-facing steps. The analysis will particularly emphasize the "periodic hills" configuration, widely recognized for its significance in turbulence modeling.

The implementation of neural networks in this study will employ PirateNet, a platform for Physics-Informed Deep Learning with Residual Adaptive Networks. Comparative analyses will also be conducted using traditional methods, including Multi-Layer Perceptrons (MLPs), to benchmark the performance of PINNs against established systems. Key aspects of PirateNet’s framework will be evaluated, such as its block structure, activation functions, and loss components (e.g., L2 loss). This evaluation will aim to determine whether PirateNet offers superior performance in turbulence modeling and, if so, to identify the factors contributing to its efficacy. By systematically comparing the results, this research will contribute to understanding the effectiveness of physics-informed architectures in addressing complex fluid dynamics problems.

2.What are Physics Informed Neural Networks (PINN's) ?

Physics-Informed Neural Networks (PINNs) are an advanced machine learning approach designed to solve problems governed by Partial Differential Equations (PDEs). Unlike traditional numerical methods, PINNs utilize a neural network to approximate PDE solutions by minimizing a loss function that integrates physical constraints. This loss function incorporates terms that enforce initial and boundary conditions along the boundaries of the space-time domain and residuals of the PDE at selected collocation points within the domain. By embedding these physical principles directly into the learning process, PINNs ensure that the model's predictions adhere to known physics, enabling accurate and physically plausible solutions. Once trained, PINNs can provide efficient and flexible approximations for complex PDEs across the integration domain, offering significant advantages over conventional solvers.

The integration of physical knowledge into machine learning models is referred to by various terms, including "physics-informed," "physics-based," "physics-guided," and "theory-guided," which are often used interchangeably. These approaches aim to combine the representational power of deep learning with physical principles, such as governing equations, to guide the learning process. By doing so, they reduce reliance on extensive datasets while improving the accuracy and reliability of predictions.

PINNs were first introduced in 2017 through two independent studies **[6,7]** and later unified in 2019 by Raissi et al. **[8]**, who demonstrated their effectiveness in solving nonlinear PDEs such as the Schrödinger, Burgers, and Allen-Cahn equations. PINNs address both forward problems, where the goal is to approximate solutions to known mathematical models, and inverse problems, where unknown parameters are inferred from observed data. This dual capability makes PINNs a powerful and versatile tool in various scientific and engineering fields. Recent applications have demonstrated their potential in fluid mechanics, bio-engineering, material science, molecular dynamics, electromagnetics, geosciences, and thermal system design. By incorporating physical laws into their architecture, PINNs offer a highly efficient and robust framework for solving complex, real-world problems while maintaining strict adherence to underlying physical constraints.

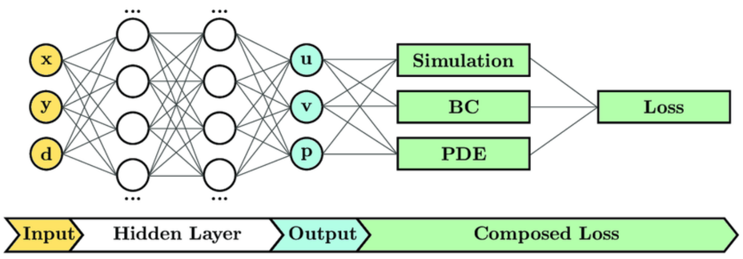


Figure 1: Neural network architecture of the PINN.

What are PirateNets (Physics-Informed Residual Adaptive Networks) ?

PirateNets are a novel architecture for physics-informed neural networks (PINNs), designed to enhance the performance of deep learning models in solving partial differential equations (PDEs). These networks optimize the minimization of residuals by focusing on the differentiability of the model, specifically targeting second-order linear elliptic and parabolic differential equations, where training errors converge more efficiently. Unlike traditional approaches, such as Multi-Layer Perceptrons (MLPs) and ResNET, which tend to increase the relative L2 error, PirateNets have demonstrated superior performance in capturing complex physics. **[8]**

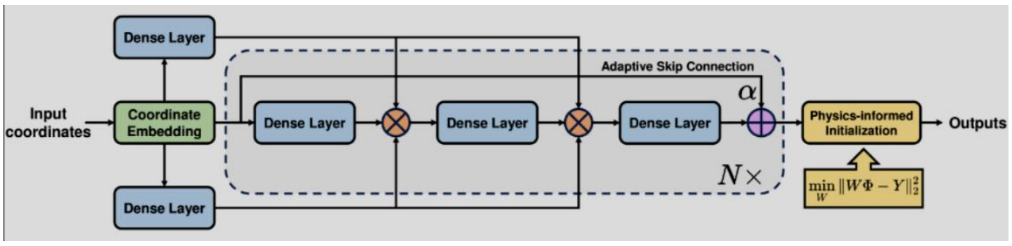


Figure 2: Neural Network Architecture of PirateNET's

As the training progresses, the model's depth gradually increases due to the activation of nonlinearities, which enhances its capacity for approximation. This adaptive learning process allows PirateNets to more effectively solve the PDEs that govern physical phenomena, yielding more accurate and efficient predictions.

One of the key innovations in PirateNets is the introduction of the **Density Block**—a specialized layer that significantly improves the network's ability to minimize PDE-related losses. The equations defining the Density Blocks are as follows:

$$f^{(l)} = \sigma\left(W_1^{(l)} x^{(l)} + b_1^{(l)}\right)$$

$$z_1^{(l)} = f^{(l)} \odot U + \left(1 - f^{(l)}\right) \odot V$$

$$g^{(l)} = \sigma\left(W_2^{(l)} z_1^{(l)} + b_2^{(l)}\right)$$

$$z_2^{(l)} = g^{(l)} \odot U + \left(1 - g^{(l)}\right) \odot V$$

$$h^{(l)} = \sigma\left(W_3^{(l)} z_2^{(l)} + b_3^{(l)}\right)$$

$$x^{(l+1)} = \alpha^{(l)} h^{(l)} + \left(1 - \alpha^{(l)}\right) x^{(l)}$$

In these equations, σ represents the activation function, W denotes weight matrices, bbb stands for biases, and U and V are learnable parameters that help refine the network's capacity to learn from the data. The adaptive skip connection, characterized by the parameter $\alpha^{(l)}$ plays a critical role in adjusting the depth of the model as it progresses through training. This allows the network to gradually transition from a linear combination of coordinate embeddings to a more complex, non-linear function that better approximates the solution to the governing PDEs.

Through the application of these Density Blocks, PirateNets are able to minimize the residuals associated with the differential equations more efficiently than traditional architectures, improving both the speed and accuracy of the solution process.

3. Navier Stokes Equations, RANS Equations and Periodic Hills

The Navier-Stokes equations govern the motion of viscous fluid substances, describing how the velocity field of a fluid evolves over time under the influence of various forces, including pressure gradients, viscous forces, and external forces. While these equations are fundamental in fluid dynamics, directly solving them for turbulent flows is computationally expensive and often impractical. To address this, turbulence models are employed, and one such model is the Reynolds-Averaged Navier-Stokes (RANS) equations. These equations simplify the Navier-Stokes equations by averaging out the effects of turbulence, allowing for more manageable computations.
RANS models decompose velocity into mean and fluctuating components, representing time-averaged flow while smoothing chaotic fluctuations [9]. In particular, data-driven closures for RANS models have revitalized turbulence modeling **[13].** This approach efficiently captures large-scale features of turbulence, as demonstrated in the Periodic Hills case, which involves periodic boundary conditions inducing flow separation, recirculation, and turbulence. Insights from such cases apply to engineering fields, including aerodynamics and hydrodynamics.

Reynolds-averaged Navier-Stokes (RANS) models offer a more efficient approach for predicting mean flow quantities and remain the industry standard, with this trend expected to continue in the future **[11]**. Their accuracy, however, depends heavily on the closure model chosen. The increasing availability of computational resources and advancements in scalable machine learning frameworks have significantly impacted computational fluid mechanics **[12].**
The Periodic Hills case is a canonical problem in fluid dynamics, involving the Navier-Stokes equations with periodic boundary conditions. This setup induces complex flow behaviors such as flow separation, recirculation zones, and turbulence. The insights gained from studying this case are applicable to a variety of real-world scenarios, including airflow over buildings and structures, river and coastal engineering, automotive and aeronautical applications, wind turbine design, and hydrodynamics in engineering. Understanding the

flow patterns in these contexts is critical for optimizing designs and improving safety and performance in engineering applications.

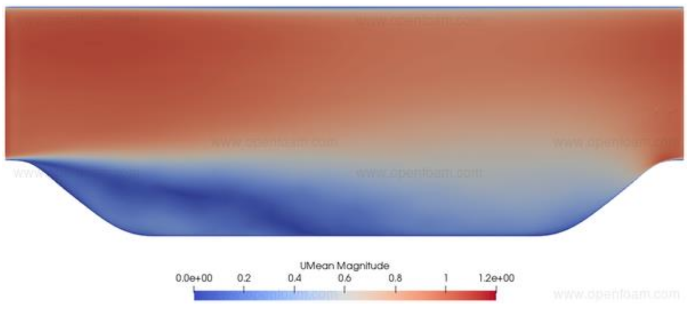


Figure 3: A sample image of a Periodic Hills, colors represent mean velocity

3. Scope of this Study

This study combines physical models with neural networks like PirateNET to assess their performance in predicting fluid dynamics under different conditions. The models aim to accurately predict key fluid characteristics, such as mean velocity, by minimizing loss functions that measure deviations from the exact solution. The impact of various turbulence models on these predictions will be examined. The study will also explore how physical parameters, such as flow conditions and boundary settings, affect accuracy. Additionally, it will investigate the influence of hyperparameters like activation functions, hidden layers, learning rates, and weight initializations. By adjusting these parameters, the study will evaluate how architectural choices, such as layer depth and regularization techniques, affect the model's ability to capture fluid dynamics. This comparison will help us better understand how turbulence models interact with neural network configurations.

The study focuses on analyzing fluid behavior under periodic boundary conditions using a dataset obtained from OpenFOAM simulations. This dataset is curated from the article "A curated dataset for data-driven turbulence modelling" by McConkey, Yee, and Lien.**[5]** The dataset contains various turbulence levels and flow conditions under periodic boundary conditions, which are crucial for this study's analysis. The turbulence is modeled using the k-epsilon model, a widely used model in computational fluid dynamics (CFD) to simulate two-dimensional turbulent flows. **[10]** Periodic boundary conditions, where the fluid properties at one boundary are repeated at the opposite boundary, are employed to simulate flow in repetitive or cyclic environments. The dataset consists of five distinct scenarios—PHLL_case_0p5, PHLL_case_0p8, PHLL_case_1p0, PHLL_case_1p2, and PHLL_case_1p5—representing various turbulence levels and flow conditions under these periodic conditions.

To relate these scenarios to physical phenomena, the study will utilize the Navier-Stokes equations, which govern the motion of incompressible fluids. These equations describe the velocity components and pressure of the fluid, with an emphasis on their derivatives. Specifically, the terms

$$f = u \cdot u_x + v \cdot u_y + p_x - \nu \cdot (u_{xx} + u_{yy}) + u \cdot u_x + u \cdot v_y$$

and

$$g = u \cdot v_x + v \cdot v_y + p_y - \nu \cdot (v_{xx} + v_{yy}) + u \cdot v_x + v \cdot v_y$$

where u u and v v are the velocity components, p p is the pressure, and ν ν is the viscosity. The term

$$\text{continuity} = u_x + v_y$$

represents the continuity equation, ensuring mass conservation.
The study will calculate the loss for these terms as part of the neural network training:

$$f_{loss} = \text{mean}(f^2) \quad ; \quad g_{loss} = \text{mean}(g^2)$$

These loss functions quantify the deviation of the model from the exact solution, which will be minimized in the process of integrating these physical models with alternative neural networks, such as PirateNET. The aim is to explore how different models perform in detecting fluid dynamics under varying conditions. By comparing the predictive capabilities of these models, the study seeks to evaluate their accuracy in determining the mean velocity magnitude of the fluid and understand how different turbulence models affect this prediction.

4. Setting Up the Model

In this section, we will explore the visualization of the dataset used, highlighting key findings and trends observed during the training process. The analysis will focus on how the accuracy scores evolve in relation to the model parameters and the changes in performance across iterations. Additionally, we will examine the variations in the model's loss function and how different parameter settings influence the convergence behavior. A detailed look into the iterative changes will provide insights into how the model adapts and improves over time, as well as the interplay between the physical constraints and data-driven components. The results will be compared to assess the effectiveness of different optimization strategies and parameter configurations, offering a comprehensive view of the model's behavior and performance during training.

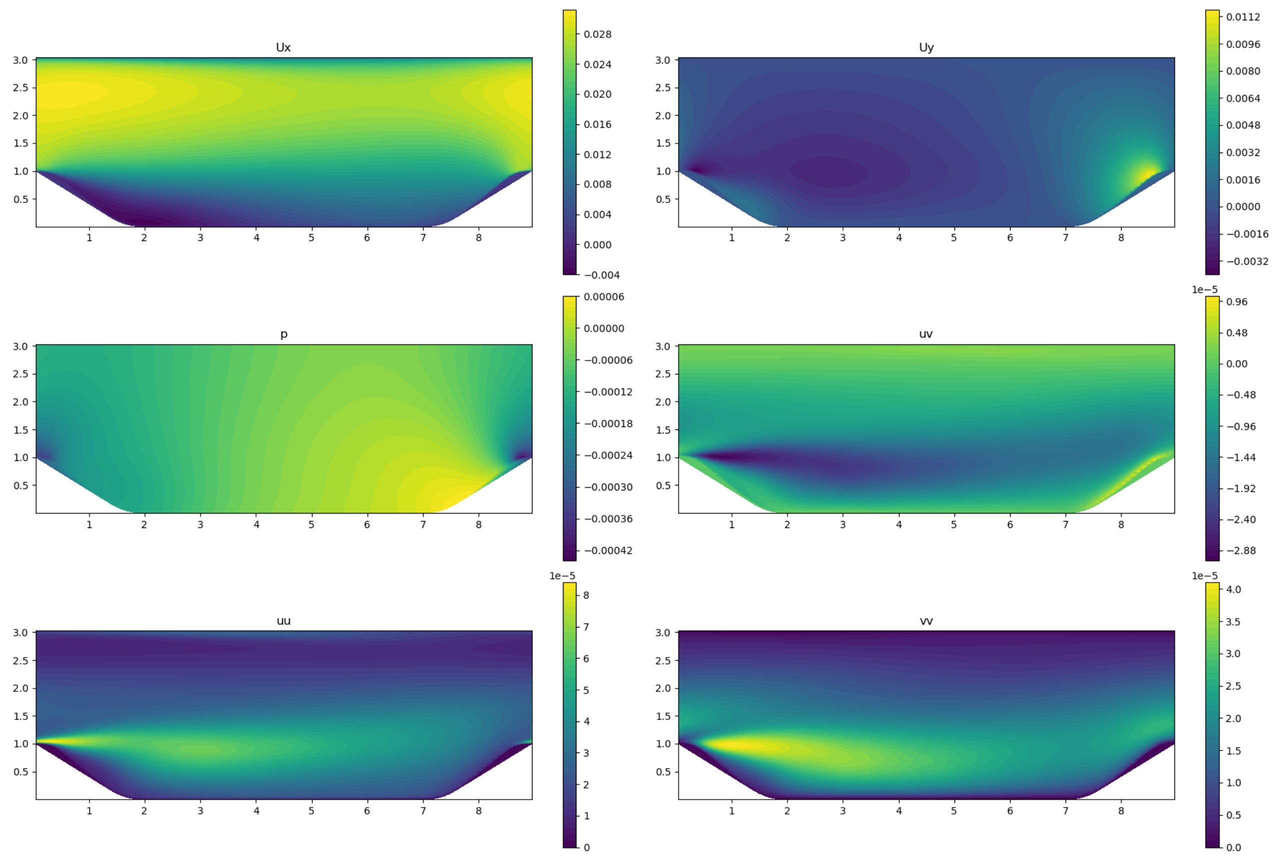
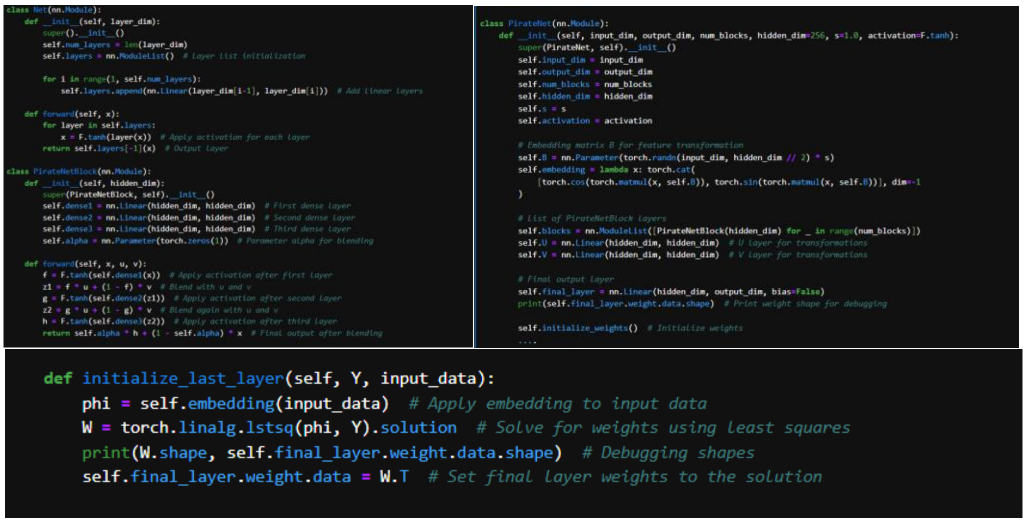


Figure 4: Visualization of Velocity and Pressure Fields in Fluid Flow on a Periodic Hill from Real Data

In the plot above, we have six subplots showing the contour plots for different fluid flow variables: Ux, Uy, p, uv, uu, and vv. These variables are crucial for understanding the dynamics of fluid flow, particularly when applying the Navier-Stokes equations. The first subplot displays Ux, the velocity component in the x-direction, which shows the speed and direction of the flow along the x-axis. The second subplot shows Uy, the velocity component in the y-direction, providing insights into the fluid's movement along the vertical axis. The third subplot presents p, the pressure field, which influences fluid behavior under various conditions and is determined by the interaction between velocity and external forces. The fourth subplot illustrates uv, the product of velocities in the x and y directions, a useful quantity for analyzing turbulence and calculating shear stress. The fifth and sixth subplots show uu and vv, the squared velocities in the x and y directions, respectively, which help in analyzing the kinetic energy distribution within the flow.

The experimental data used for these plots is based on a 14,751 by 14,751 grid derived from a Hill test case. As mentioned, the variables Ux, Uy, p, uu, and vv are predicted by solving the Navier-Stokes equations, which govern the fluid dynamics, taking into account both pressure and velocity components. For this specific setup, 7,000 collocation points have been selected across the grid to ensure the model can accurately evaluate and predict flow characteristics at these critical locations.



```python
class Net(nn.Module):
    def __init__(self, layer_dim):
        super().__init__()
        self.num_layers = len(layer_dim)
        self.layers = nn.ModuleList()  # Layer list initialization

        for i in range(1, self.num_layers):
            self.layers.append(nn.Linear(layer_dim[i-1], layer_dim[i]))  # Add linear layers

    def forward(self, x):
        for layer in self.layers:
            x = F.tanh(layer(x))  # Apply activation for each layer
        return self.layers[-1](x)  # Output layer

class PirateNetBlock(nn.Module):
    def __init__(self, hidden_dim):
        super(PirateNetBlock, self).__init__()
        self.dense1 = nn.Linear(hidden_dim, hidden_dim)  # First dense layer
        self.dense2 = nn.Linear(hidden_dim, hidden_dim)  # Second dense layer
        self.dense3 = nn.Linear(hidden_dim, hidden_dim)  # Third dense layer
        self.alpha = nn.Parameter(torch.zeros(1))  # Parameter alpha for blending

    def forward(self, x, u, v):
        f = F.tanh(self.dense1(x))  # Apply activation after first layer
        z1 = f * u + (1 - f) * v  # Blend with u and v
        g = F.tanh(self.dense2(z1))  # Apply activation after second layer
        z2 = g * u + (1 - g) * v  # Blend again with u and v
        h = F.tanh(self.dense3(z2))  # Apply activation after third layer
        return self.alpha * h + (1 - self.alpha) * x  # Final output after blending
```

```python
class PirateNet(nn.Module):
    def __init__(self, input_dim, output_dim, num_blocks, hidden_dim=256, s=1.0, activation=F.tanh):
        super(PirateNet, self).__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.num_blocks = num_blocks
        self.hidden_dim = hidden_dim
        self.s = s
        self.activation = activation

        # Embedding matrix B for feature transformation
        self.B = nn.Parameter(torch.randn(input_dim, hidden_dim // 2) * s)
        self.embedding = lambda x: torch.cat(
            [torch.cos(torch.matmul(x, self.B)), torch.sin(torch.matmul(x, self.B))], dim=-1
        )

        # List of PirateNetBlock layers
        self.blocks = nn.ModuleList([PirateNetBlock(hidden_dim) for _ in range(num_blocks)])
        self.U = nn.Linear(hidden_dim, hidden_dim)  # U layer for transformations
        self.V = nn.Linear(hidden_dim, hidden_dim)  # V layer for transformations

        # Final output layer
        self.final_layer = nn.Linear(hidden_dim, output_dim, bias=False)
        print(self.final_layer.weight.data.shape)  # Print weight shape for debugging

        self.initialize_weights()  # Initialize weights
        ....
```

```python
    def initialize_last_layer(self, Y, input_data):
        phi = self.embedding(input_data)  # Apply embedding to input data
        W = torch.linalg.lstsq(phi, Y).solution  # Solve for weights using least squares
        print(W.shape, self.final_layer.weight.data.shape)  # Debugging shapes
        self.final_layer.weight.data = W.T  # Set final layer weights to the solution
```

The code block above defines a neural network architecture implemented using PyTorch, consisting of three main components.

The Net class is a basic fully connected neural network that accepts a list of layer dimensions as input and applies the tanh activation function to each hidden layer.

The PirateNetBlock class is a custom building block used within the PirateNet architecture. It consists of three dense layers with activations, blending the input data with intermediate transformations via a parameter (alpha), allowing dynamic mixing between inputs and intermediate values.

Finally, the PirateNet class serves as the main model, which incorporates multiple PirateNetBlock layers. It features a unique embedding transformation for the input data, along with additional transformations (U and V layers) and a final linear output layer. Additionally, the initialize_last_layer function uses a least squares approach to compute and set the weights of the final layer, further refining the model's ability to make accurate predictions.

5.Model Results:

5.1 PirateNET + Adam Optimizer & 8000 iteration, lr = 0.1

In the model training process, the Adam optimizer was used in conjunction with PirateNET to integrate pressure and velocity values into the Navier-Stokes equations. With a learning rate of 0.01 over 8000 iterations, the optimizer facilitated the monitoring of loss functions, including the PDE loss (representing the contribution from the physical model) and the data loss (capturing discrepancies between the model's predictions and actual data).

The Adam optimizer was chosen because it is particularly well-suited for large and complex deep neural networks like PirateNET, which integrates both physical laws and data-driven components. As a first-order gradient-based optimization algorithm, Adam efficiently handles stochastic loss functions **[15]**, which are common in deep learning tasks

Model Loss by Iterations

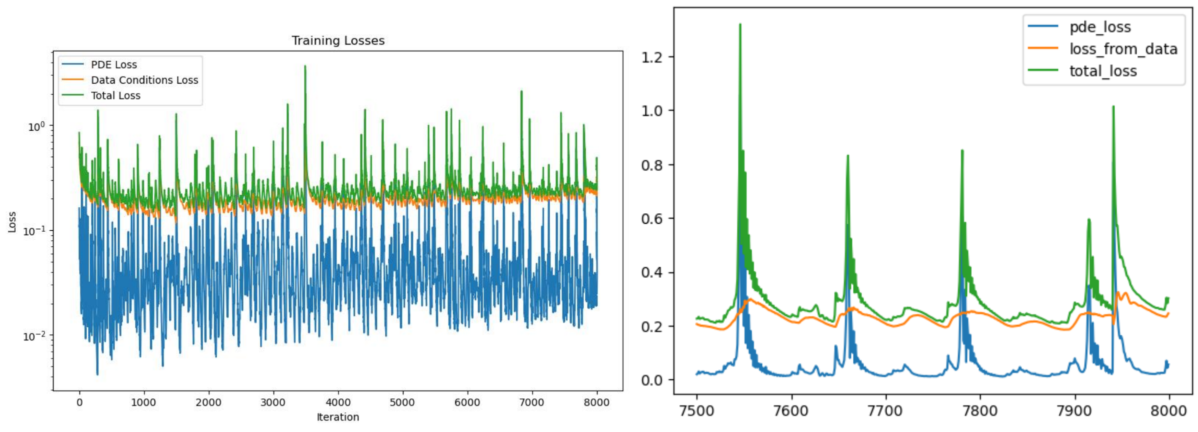


Figure 5: PirateNET's PDE,Data & Total Losses with Adam Optimizing Approach

The graphs of PDE loss, data loss, and total loss illustrate how the model progressed over time. Initially, the PDE loss fluctuated significantly as the model adapted to the physical laws, while the data-driven loss showed relatively smaller variations. However, after a certain point, the model's loss began to stagnate, indicating a lack of meaningful progress in the optimization process. This plateau was attributed to the Adam optimizer's adaptive learning rate mechanism, which, while effective in many cases, can lead to premature convergence in complex models like those based on the Navier-Stokes equations. As the training progressed, the smaller gradients in the PDE loss led Adam to reduce the learning rate excessively, causing the optimizer to make smaller and less effective updates. This resulted in the loss function reaching a plateau, and the model seemed to be stuck in a local minimum, suggesting that a higher learning rate or a different optimization approach like L-BFGS might be necessary for better convergence.

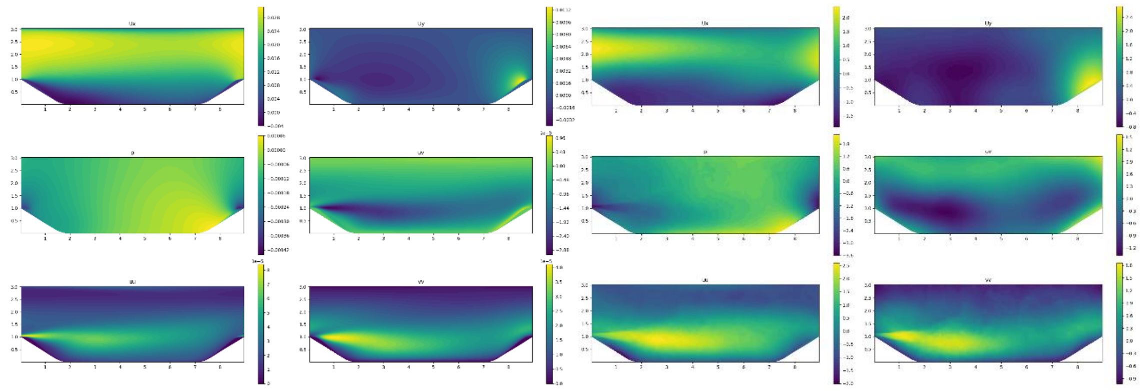


Figure 6: Actual Data (Left) vs Prediction Data with after 8000 iterations (Right)

Model L2 Accuracy Scores:

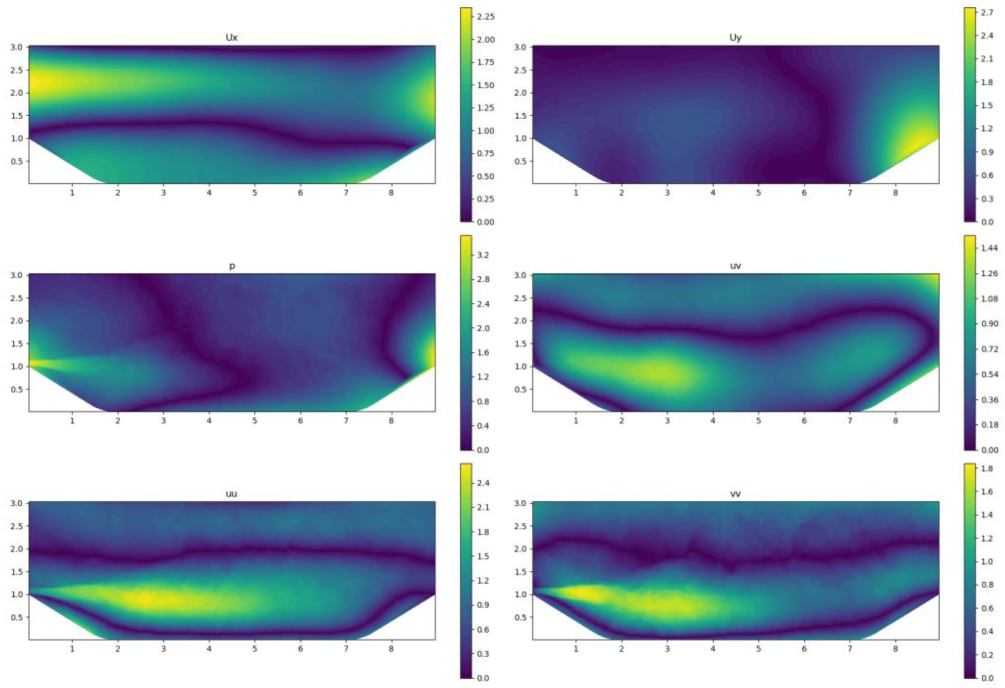


Figure 7: Relative L2 Errors

The accuracy scores were calculated by comparing the model outputs with the true outputs from the initial graph using the Relative L2 error. In the regions marked in yellow, the error is higher, whereas the blue regions indicate lower error. The Relative L2 error can be expressed mathematically as:

$$\text{Relative L2 Error} = \frac{\sqrt{\sum_i f_{\text{true}}(x_i)^2}}{\sqrt{\sum_i \left(f_{\text{model}}(x_i) - f_{\text{true}}(x_i)\right)^2}}$$

Where $f_{\text{model}}(x_i)$ is the predicted value from the model, $f_{\text{true}}(x_i)$ is the true value, and the summation is over all data points. This metric gives a measure of how well the model's predictions align with the true values, with a smaller relative L2 error indicating better accuracy.

5.2 PirateNET + L-BFGS Optimizer & 12000 iteration, lr = 0.05

When the optimizer was switched to L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno), which is a second-order optimization method, the model showed significant improvements. L-BFGS is different from Adam in that it approximates the Hessian matrix (second-order derivatives), allowing for more accurate parameter updates. This second-order approach is particularly effective in problems with complex loss surfaces, such as fluid dynamics, where Adam's first-order gradients may struggle**. [14]** L-BFGS also uses a line search to determine the optimal step size, which helps the optimizer avoid local minima or plateaus where Adam might get stuck. As a result, L-BFGS achieved more stable convergence, with a notable reduction in both the PDE and data-driven loss functions. The ability to take into account second-order information allowed L-BFGS to optimize the model more effectively, balancing the contributions from both the physical model and the data-driven components, leading to faster and more reliable convergence The sudden significant increases in the Total Loss values with the Adam optimizer were removed when optimizer switched L-BFGS. In this case, L-BFGS was able to significantly reduce the fluctuations in the loss function, demonstrating its ability to handle the complexities of the fluid dynamics model more efficiently than Adam. This improvement can be attributed to the memory-efficient nature of L-BFGS **[16]** and its application in large parameter optimization problems as implemented in TensorFlow Probability **[17].**

Due to the specified reasons, re-estimation was performed using L-BFGS instead of the Adam Optimizer in the study below, and accurate results were obtained. Here, the outputs of the study, created with 12000 iterations and an alpha rate of 0.05, will be examined.

Model Loss By Iterations

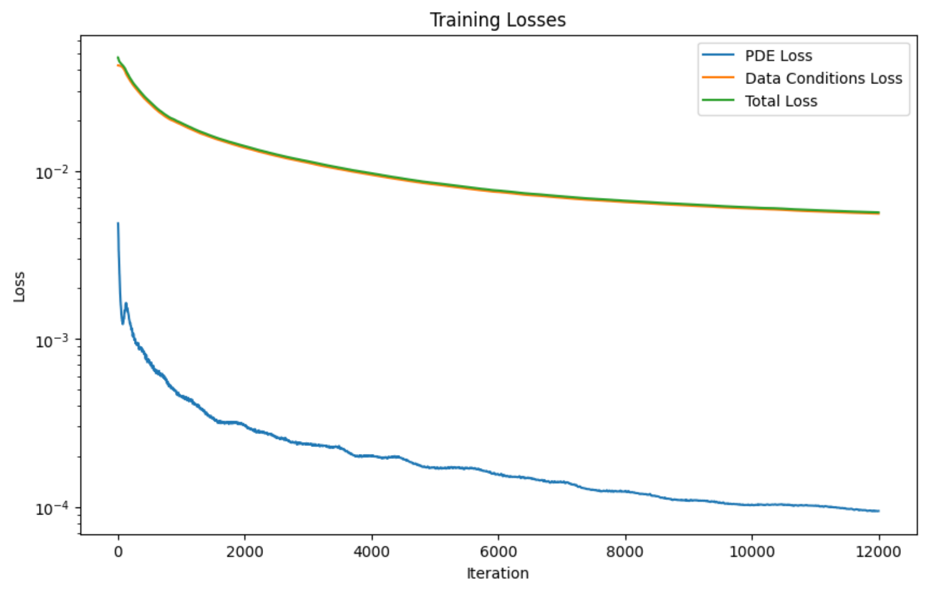


Figure 8: Training Loses using LBFGS by iterations

In the PirateNET framework, the LBFGS optimizer's ability to determine the gradient with higher precision during each iteration effectively mitigates the peaks observed in the Adam optimizer. In the graph above, the total loss achieved with the Adam optimizer at $[\mathbf{10^{-2}}]$ was reduced more quickly and without deviation to $[\mathbf{10^{-4}}]$ using LBFGS. This improvement enables more accurate pressure and velocity estimations in the fluid's local regions. By addressing the limitations of Adam, the LBFGS optimizer facilitates more reliable predictions of the fluid's behavior, ensuring that the flow characteristics are determined with greater accuracy. This advancement highlights the superiority of the LBFGS method in achieving stability and precision in fluid dynamics simulations.

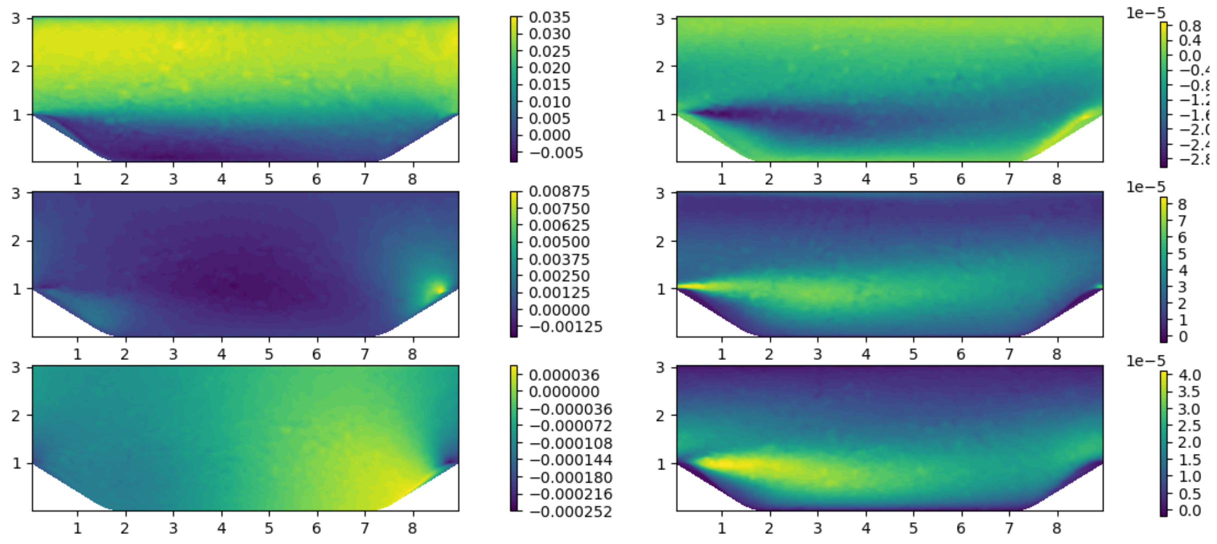


Figure 9: Pressure and Velocity predictions using L-BFGS

Model L2 Accuracy Scores

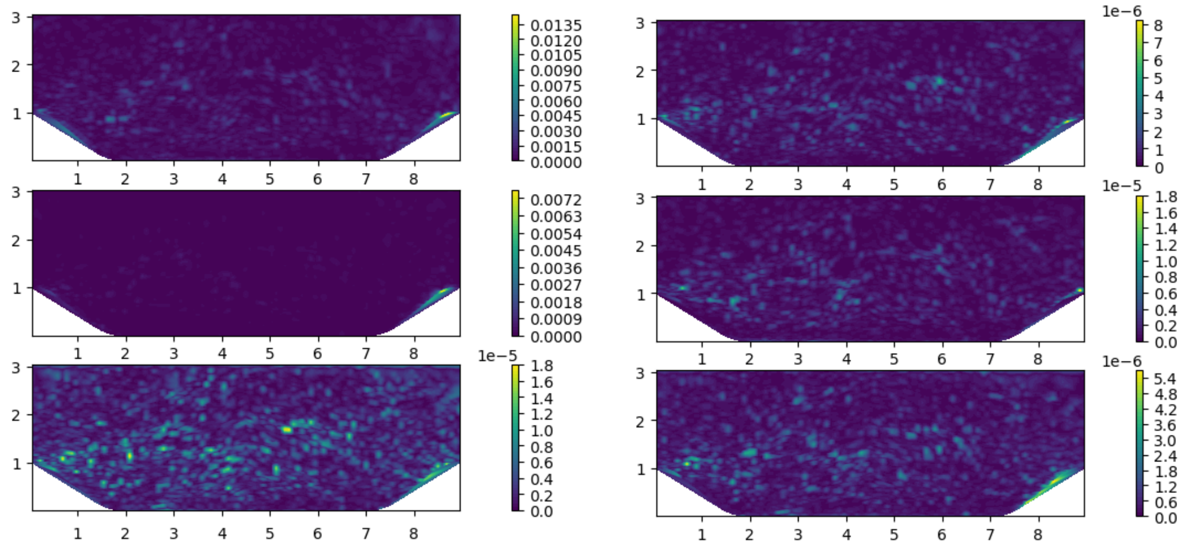


Figure 10: L2 Losses using LBFGS

4. Conclusion

In conclusion, this study demonstrates that PirateNETs are not only successful in fields such as image processing, signal processing, and financial forecasting but also show remarkable effectiveness across various branches of physics, including thermodynamics, quantum mechanics, material science, and even astrophysics. These findings highlight the versatility and power of PirateNETs in modeling complex physical systems. Specifically, in fluid modeling, the study revealed the model's impressive capability to handle intricate fluid dynamics, making it a valuable tool for real-world applications. The results also emphasize the importance of optimizer choice in training such models.

Optimizers like LBFGS outperformed others, particularly in updating the partial derivative losses in the model, with results nearly 100 times more efficient in terms of the loss function. In contrast, optimizers like Adam struggled with the complexity of the model, often leading to divergence despite the same number of iterations. This insight into the optimizer's role further strengthens the potential of PirateNETs in applications that require precise and reliable outcomes.

5. References

[1] Slotnick, J., Khodadoust, A., Alonso, J., & Darmofal, D. (2014). CFD vision 2030 study: A path to revolutionary computational aerosciences. Tech. Rep. March.

[2] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 367(6481):1026–1030, 2020.

[3] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physicsconstrained deep learning without simulation data. Computer Methods in Applied Mechanics and Engineering, 361:112732, 2020.

[4] Abhilash Mathews, Manaure Francisquez, JerryWHughes, David R Hatch, Ben Zhu, and Barrett N Rogers. Uncovering turbulent plasma dynamics via deep learning from partial observations. Physical Review E, 104(2):025205, 2021.

[5] McConkey, R., Yee, E., & Lien, F. S. (2021). A curated dataset for data-driven turbulence modelling. *Scientific Data, 8*, 255. https://doi.org/10.1038/s41597-021-01034-2

[6] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics, 335*, 736–746. https://doi.org/10.1016/j.jcp.2017.01.060

[7] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics, 348*, 683–693. https://doi.org/10.1016/j.jcp.2017.07.050

[8] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). 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. https://doi.org/10.1016/j.jcp.2018.10.045

[9] Alfonsi, Giancarlo. (2009). Reynolds-Averaged Navier-Stokes Equations for Turbulence Modeling. Applied Mechanics Reviews - APPL MECH REV. 62. 10.1115/1.3124648.

[10] Hanjalić, K., and Launder, B. E., 1972, "A Reynolds-Stress Model of Turbulence and its Application to Thin Shear Flows," *J. Fluid Mech.*, 52, pp. 609–638.

[11] Slotnick, J. P., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., & Mavriplis, D. J. (2014). *CFD vision 2030 study: a path to revolutionary computational aerosciences* (No. NF1676L-18332).

[12] Brunton, S. L., Noack, B. R., & Koumoutsakos, P. (2020). Machine learning for fluid mechanics. Annual review of fluid mechanics, 52(1), 477-508.

[13] Duraisamy, K. (2021). Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence. *Physical Review Fluids*, *6*(5), 050504.

[14] Taylor, J., Wang, W., Bala, B., & Bednarz, T. (2022). Optimizing the optimizer for data-driven deep neural networks and physics-informed neural networks (arXiv:2205.07430v1). https://arxiv.org/abs/2205.07430

[15] Kingma, D. P. and J. L. Ba (2015). Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. ArxIV

[16] Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming, 45*(1), 503–528. https://doi.org/10.1007/BF01589116

[17] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, Ł., Kudlur, M., ... Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*. https://doi.org/10.48550/arXiv.1603.04467