CS-433: Project 2: Solving Navier Stokes equations using PINNs

Adam Mesbahi 387382
Ibrahim Beniffou 370940
Matya Aydin 388895
Supervised by Professor Marco Picasso

***Abstract*—This paper explores Physics-Informed Neural Networks (PINNs) as a novel alternative for solving PDEs. By formulating a custom loss function that integrates data fidelity, PDE satisfaction, and boundary conditions, we optimize a neural network to approximate solutions to both steady and unsteady Navier-Stokes equations.**

I. Introduction

Partial differential equations are tedious and computationally expensive to solve. We propose an alternative to common industrial methods such as finite elements and finite volumes by using neural networks. An approximate loss function can indeed force the resulting function to satisfy the PDE. This paper focuses on the Navier-Stokes equations:

$$\begin{gathered}\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}+\frac{1}{\rho}\nabla p+\nu\nabla^2 u=0 \text{ on } \Omega\\ \nabla\cdot\mathbf{u}=0 \text{ on } \Omega\\ \mathbf{u}=\mathbf{u}_\Gamma \text{ on } \partial\Omega\end{gathered}$$

Where $\mathbf{u}$ is the velocity vector: $\mathbf{u}=(v_x,v_y)^\top$, p is the pressure field, ρ the density and ν the kinematic viscosity of the fluid. Ω denotes the interior of the domain.

We introduce our general framework then test it on both a steady example that admits an analytical solution then an unsteady example.

II. Model

A. Architecture

Our model is a fully connected multi-layer perceptron (MLP) that takes as input the (x,y) coordinates and the time t and gives the pressure and velocity $\mathbf{h_W}(x,y,t)=(p(x,y,t),v_x(x,y,t),v_y(x,y,t))$ associated. As we need to take the derivative of the output, we use $\sigma(l)=\tanh(l)$ as an activation function because it is continuously differentiable and its higher order derivatives are numerically stable. Moreover, $ReLU$ has a zero second derivative and the sigmoid function leads to vanishing gradient and its second derivative decays quickly.

B. Loss function

Our loss function is a weighted sum of 4 terms:

$$\mathcal{L}(\mathbf{W}|\mathbf{X}_i)=\mathcal{L}_1+\alpha\mathcal{L}_2+\beta\mathcal{L}_3+\gamma\mathcal{L}_4$$

Where $\mathbf{X}_i$ is a vector that concatenates the spatial and temporal coordinates : $\mathbf{X}_i=(x_i,y_i,t_i)^\top$

1) The first term ensures that we correctly approximate the N samples and is a standard MSE:
$$\mathcal{L}_1(\mathbf{W})=\frac{1}{N}\sum_{i=1}^{N}(h_{\mathbf{W}}(\mathbf{X}_i)-\mathbf{Y}_i)^2$$
Where $\mathbf{Y}_i$ is a vector that concatenates $\mathbf{u}$ and p: $\mathbf{Y}_i=(v_x(x_i,y_i,t_i),v_y(x_i,y_i,t_i),p(x_i,y_i,t_i))^\top$.
$h_{\mathbf{W}}(\cdot)$ denotes our neural network of parameters $\mathbf{W}$. The samples are being generated by finite volumes using Ansys Fluent.

2) The second term ensures that $h_{\mathbf{W}}(\mathbf{X},t)$ satisfies the 3 equations in Ω:
$$\mathcal{L}_2(\mathbf{W})=\frac{1}{3\tilde{N}}\sum_{i=1}^{\tilde{N}}\sum_{j=1}^{3}\mathcal{N}_j(h_{\mathbf{W}}(\mathbf{X}_i))^2$$
$\mathcal{N}_j$ is a functional operator that plugs $(h_{\mathbf{W}}(\mathbf{X}_i))$ into the j^{th} equation. Theoretical reason motivates that each of these terms should have the same importance in the sum.

We note that in general, $N\neq\tilde{N}$ and $N<\tilde{N}$. As we do not need any value of the solution for this term, we can evaluate $(h_{\mathbf{W}}(\mathbf{X}_i))$ on as much coordinates inside Ω as needed.

3) The third term ensures that the boundary conditions are met on $\partial\Omega$
$$\mathcal{L}_3(\mathbf{W})=\frac{1}{N_b}\sum_{\mathbf{X}_i\in\partial\Omega}^{N_b}(h_{\mathbf{W}}(\mathbf{X}_i)-\mathbf{Y}_i^{\text{boundary}})^2$$
4) The last term imposes the solution at a given time t_0:
$$\mathcal{L}_4(\mathbf{W})=\frac{1}{N_i}\sum_{i|t=t_0}^{N_i}(h_{\mathbf{W}}(\mathbf{X}_i)-\mathbf{Y}_i^{t_0})^2$$

Similarly to $\tilde{N}$, we can arbitrarily select N_i and N_b. As these conditions are known. α, β and γ are hyperparameters to tune to give the correct importance to each term during training. The right number of terms in each sum: N, $\tilde{N}$, N_b and N_i must also be determined.

III. Steady case

A. Problem definition

We first try our approach with a time independent ($\frac{\partial\mathbf{u}}{\partial t}=0$) and adimensional case of Navier-Stokes named Kovazsnay flow that admits a closed form analytical solution Xiaowei Jin [2020]:

$$v_x(x,y)=1-e^{\lambda x}\cos(2\pi y)$$

$$v_y(x,y) = \frac{\lambda}{2\pi} e^{\lambda x} \sin(2\pi y)$$

$$p(x,y) = \frac{1}{2}(1 - e^{2\lambda x})$$

With $\lambda = \frac{Re}{2} - \sqrt{(\frac{Re}{2})^2 + 4\pi^2}$, $Re = 40$.
Ω is a rectangle: $\Omega = [-0.5, 1] \times [-0.5, 1.5]$.

In this case, the problem can be formulated as:

$$\min_{\mathbf{W}} \mathcal{L}_2 + \beta \mathcal{L}_3$$

B. Preprocessing

The $\tilde{N}$ points in the domain have been selected using `torch.meshgrid()`. Points on each side of the domain were generated using `torch.linspace()`.
In section III-F, a loader has also been initialized to feed batches of the grid into the neural network. We only batch the points inside Ω. The batch size is N_b such that the same number of points is given at each iteration for both losses.

C. Training

We used `torch.optim.Adam()` with a learning rate $\eta = 10^{-3}$ and trained on 600 epochs. No batching was required to achieve convergence. Empirical results showed that both losses saturate and additional training does not result in any improvement.

D. Hyperparameters tuning

We need to find the parameters β, $\tilde{N}$ and N_b that leads to the smallest error. To reduce the computation time, we search them on a smaller architecture with 4 hidden layers of 50 neurons. We first try to find the best β at fixed $\tilde{N}$ and N_b as depicted on figure 1. This allows us to conclude that $\beta = 10$.

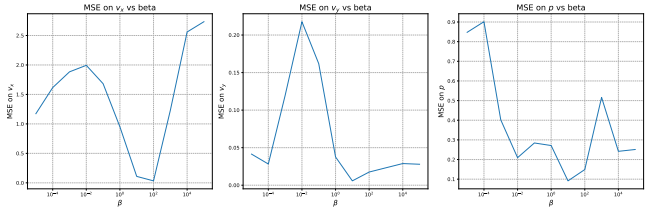


Figure 1: MSE for different values of β in the steady case

We then performed a 2 dimensional grid search with the value that we found for β as depicted on figure 2. This allows us to keep $\tilde{N} = 60^2$ and $N_b = 4 \cdot 50$.

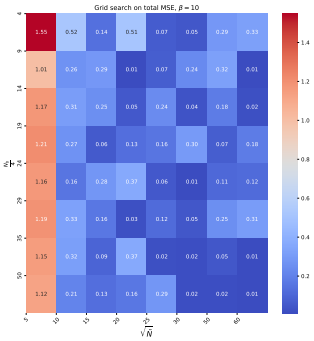


Figure 2: Grid search for $\tilde{N}$ and N_b

With these hyperparameters, we studied the influence of the size of the neural network on the error, which led to marginal improvement for the additional computation cost. This means that the function space that our model can approximate already contains a good approximate of the exact solution.

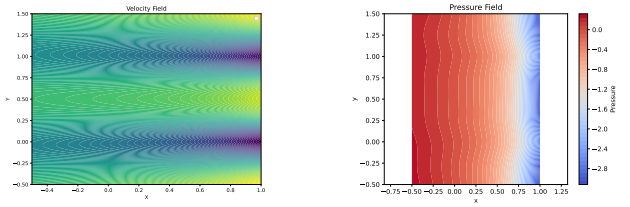
E. Results

We keep an architecture of 10 layers and 128 neurons per layer for our model and evaluate it at more coordinates of the domain than it was trained on. A visual comparison is provided on figures 4 and 3.

To quantify the error at $\mathbf{X}_i$, we use the relative L_2 error:

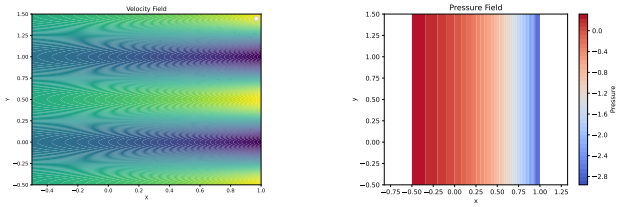
$$e_{ij} = \frac{\|\hat{\mathbf{Y}}_{ij} - \mathbf{Y}_{ij}\|_2}{\|\mathbf{Y}_{ij}\|_2}$$

Where j denotes each entry. Numerical results are provided in table I.



(a) Predicted x- velocity (b) Predicted pressure field

Figure 3: Predicted solution



(a) True x-velocity (b) True pressure field

Figure 4: True solution

We can also interpret the graph of both losses during training with $\beta = 10$ depicted on 5:

Figure 5: Loss during training

As we can see, there is a phase of the training where the equation loss is increasing and the boundary loss is decreasing. This is due to the fact that $\beta = 10$ and thus the minimization of the boundary loss is prioritized compared with $\mathcal{L}_2$. Once the boundary loss is low enough, the contribution of the boundary loss gradient vector in the total loss gradient vector is not much important because the boundary loss is close to zero and thus, the equation loss can be minimized despite having a bigger weight on $\mathcal{L}_3$.

F. Dynamical weights and new architecture

As searching for optimal weights for each loss is time consuming, computationally expensive, and application dependent, we introduce a way to dynamically adapt them during training. We explored different ways of computing these weights (we still have $\alpha = 1$):

- using the norms of the gradient vectors : $\beta = \frac{||\nabla \mathcal{L}_3||_2}{||\nabla \mathcal{L}_2||_2}$
- using the mean of the absolute values of the components of the gradient vector : $\beta = \frac{\overline{|\nabla \mathcal{L}_3|}}{\overline{|\nabla \mathcal{L}_2|}}$ Xiaowei Jin [2020]
- using the max and the mean of the absolute values of the components of the gradient vector : $\beta = \frac{\max |\nabla \mathcal{L}_3|}{\overline{|\nabla \mathcal{L}_2|}}$ Xiaowei Jin [2020]

Moreover, we tested another novel architecture presented in S. Wang and Perdikaris [2024]. It is inspired by computer vision and NLP tasks. It is basically a MLP to which we added two transformers. It has the following properties :

- it accounts for multiplicative interactions between the input dimensions
- it enhances the hidden states by adding residual connections

The model is described as following :

We compare the performance of each dynamical adjustment on a basic MLP and the novel architecture after having trained during 150 iterations. We chose to stop after 150 iterations because in all cases, the loss functions (based on the boundary conditions and on the equation) were not significantly evolving anymore. The metric is the relative L_2 error compared with the analytical solution and is evaluated on 10^4 uniformly distributed on the domain while we trained it on 2.500 points uniformly distributed in the domain. The results are presented in table I:

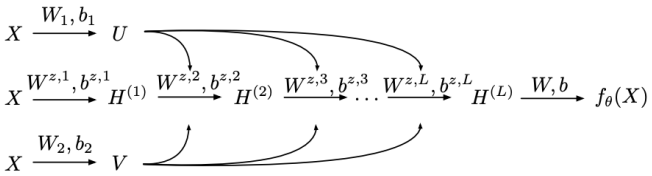


Figure 6: Novel architecture from S. Wang and Perdikaris [2024]

In the table, A1 refers to the MLP architecture, and A2 refers to the novel architecture. The MLP is composed of 10 hidden layers of 128 neurons each and tanh activation functions as well as the novel architecture, but it is worth notifying that the novel architecture has more parameters than the MLP because there are the two additional transformers layers.

	u		v		p	
$\beta =$	$A1$	$A2$	$A1$	$A2$	$A1$	$A2$
10	0.15	0.33	0.33	0.94	0.11	0.47
$\frac{\|\|\nabla \mathcal{L}_3\|\|_2}{\|\|\nabla \mathcal{L}_2\|\|_2}$	0.78	0.15	1.00	0.33	0.11	0.88
$\frac{\overline{\|\nabla \mathcal{L}_3\|}}{\overline{\|\nabla \mathcal{L}_2\|}}$	0.94	0.84	1.00	1.40	0.95	0.90
$\frac{\max \|\nabla \mathcal{L}_3\|}{\overline{\|\nabla \mathcal{L}_2\|}}$	0.92	0.19	1.00	0.50	0.34	0.28

Table I: L_2 relative error compared with the analytical solution

As we can notice, the MLP with the constant weight $\beta = 10$ is the best performer on this problem. The counter part is that we had to find the value 10 by conducting a grid search, which is computationally costly. In the context where we didn't have time or resources to conduct a grid search, the adjustment based on the norms would be the best for the MLP model and the adjustment based on the max and the mean of the absolute values of the gradient vector would be the best for the novel architecture. For more complex problems where it would be more difficult to guess the optimal value of β, we know that the dynamical adjustments presented here can be great alternatives.

IV. Unsteady case

A. Problem definition

Domain: $\Omega = [0, 1] \times [0, 0.5] \setminus \mathcal{B}((0.3; 0.25), 0.025)$

$$p(1, y, t) = 0$$

$$v_x(x, 0, t) = v_y(x, 0, t) = 0$$

$$v_x(x, 0.5, t) = v_y(x, 0.5, t) = 0$$

$$v_x = v_y = 0 \text{ on } \mathcal{B}$$

$$\|\mathbf{u}(0, y, t)\|_2 = 1$$

B. Preprocessing

We used a similar loader as described in III-B. The simulation results for each timestep were given in a different csv file. A corresponding and constant time column was manually added for each timestep.
As we observed larger fluctuations than in III, standardization also led to better results. In addition to the boundaries, we added points that we considered as critical in $\mathcal{L}_3$ to enforce minimization on them.

C. Training

We trained two models to compare the influence of $\mathcal{L}_1$:

1) Model 1 is trained on 100 epochs and solves:

$$\min_{\mathbf{W}} \mathcal{L}_1 + 100\mathcal{L}_2 + 100\mathcal{L}_3 + 10\mathcal{L}_4$$

2) Model 2 is trained on 1500 epochs and solves:

$$\min_{\mathbf{W}} \mathcal{L}_2 + 10\mathcal{L}_3$$

For both models, we have $\tilde{N} = N = 1066150$ and $N_b = 1049000$. The generated mesh has 21323 nodes on the same position for 50 timesteps. We performed the train to these 2 different models since after some researches, it turned out that the MSE was not really expected to add any major improvements. Then we decided to invest more time on the Model 2 training by multiplying the number of epoch performed by 1.5.

D. Results

For Model 1, and due to the low number of epochs, the model works very well outside the cylinder for v and p, but it attempts to extrapolate the behavior near the cylinder. As a result, it fails to capture the critical flow field around the cylinder. The model also has more difficulty accurately predicting v_x. Similarly to Model 1, Model 2 works very well for v_y and p and is significantly more effective at predicting the flow near the cylinder, with impressive results for v_y, as depicted in the appendix VII-B. While the predictions for the horizontal velocity v_x are notably improved, they still do not fully meet expectations. Visualization of our results associated to this model is available in section VII-B. They correspond to snapshots at time 3.01s but observations developed above can be generalized for any time. The two loss functions stabilized and stopped varying significantly after approximately 500 epochs out of the total 1500 epochs. It is possible that allowing the model to train for more epochs could lead to further improvements and potentially better results.

V. Ethical risks

While PINNs offer promising advantages in solving partial differential equations, their application raises several ethical considerations. Unlike conventional numerical methods, PINNs do not provide guarantees of optimality or convergence. They do not provide any bound on the error either. Furthermore, the inherent lack of interpretability in neural networks presents a challenge to explainability, particularly in high-stakes applications.

Relying on black-box methodologies to solve critical equations such as those governing the design and operation of aircraft, rockets, or other safety-critical systems introduces potential risks. Errors or inaccuracies in solutions, especially in more critical part of the domain where we would define a finer mesh with a FEM method which is not the case with a naive `meshgrid()`, may lead to catastrophic consequences, undermining safety. This trade-off between reduced computational cost and the potential lack of reliability necessitates careful ethical scrutiny and robust validation mechanisms before deploying PINNs in such sensitive domains.

If a PINN is not trained properly or overfits its training data instead of learning an underlying function, it might generalize poorly to other situations, which could lead to harmful consequences.

If we replicate the expensive steps of hyperparameters tuning presented in III-D to larger scale problem, one could also critic the ressource needed from an environmental point of view.

VI. Conclusion

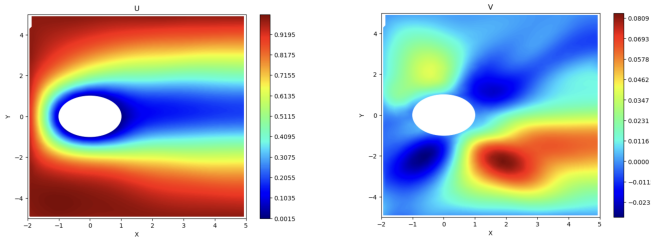
In conclusion, this method does not aim to replace traditional CFD approaches but introduces a significant innovation: the ability to enforce the respect of specific partial differential equations within a neural network at particular points. This feature allows for the seamless integration of physical constraints into the model, improving its accuracy and reliability. The results obtained on our primary problem are particularly promising, demonstrating the method's capability to provide convincing solutions. Moreover, this approach is not limited to the Navier-Stokes equations—it can be generalized to other types of equations, making it a versatile tool for tackling a wide range of complex physical problems.

VII. Appendix

A. Application on different domains

We tried to apply our methods on a more complex problem which consists of a flow passing through a cylinder. The boundary conditions are described as following :

$$\begin{cases} u = 1, v = 0 & x = 0, y \in [-5, 5] \\ u = 1, v = 0 & x \in [-2, 5], y = -5 \\ u = 1, v = 0 & x \in [-2, 5], y = 5 \\ \dfrac{\partial u}{\partial x} = 0 & x = -5, y \in [-5, 5] \\ u = 0, v = 0 & x, y \in \partial\mathcal{B}(\vec{0}, 1) \end{cases}$$



(a) Velocity in direction x. (b) Velocity in direction y.

Figure 7: Ball problem.

We conducted 150 iterations with $\beta = 10$. We firstly notice that the results seem to be qualitatively consistent with the reality. However, after having compared it with a classical CFD simulation on fluent, we know that the flow is wrong. This can be explained by several factors :

- Firstly, there is what we call a boundary limit that forms around the ball. A boundary limit is a thin layer where the speed is varying from 0 (no slip condition) to a certain constant value. However, to accurately simulate a boundary layer, we need to have a lot of points near the ball surface, which was not the case here because we used only 3600 points uniformly distributed on the domain.
- Secondly, the problem was costly to solve and we didn't have enough computation resources to refine our solution.

Note that the lack of analytical solution does not allow us to assess an accurate error as in table I.

B. Unsteady plots

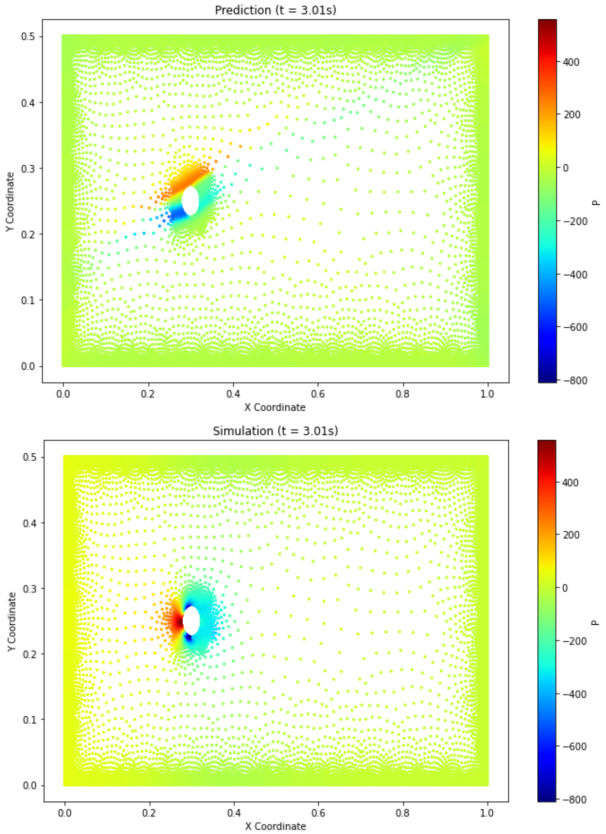


Figure 8: Comparison for the pressure

References

Guergana Petrova Andrea Bonito, Ronald DeVore and Jonathan W. Siegel. Convergence and error control of consistent pinns for elliptic pdes. *arXiv preprint arXiv:2406.09217*, 2024. URL https://arxiv.org/abs/2406.09217.

Y. Teng S. Wang and P. Perdikaris. Understanding and mitigating gradient pathologies in pinns. 2024. URL https://odi.inf.ethz.ch/teaching/AI4Science/Group7.pdf.

Hui Li George Em Karniadakis Xiaowei Jin, Shengze Cai. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *arXiv preprint arXiv:2003.06496*, 2020. URL https://arxiv.org/abs/2003.06496.

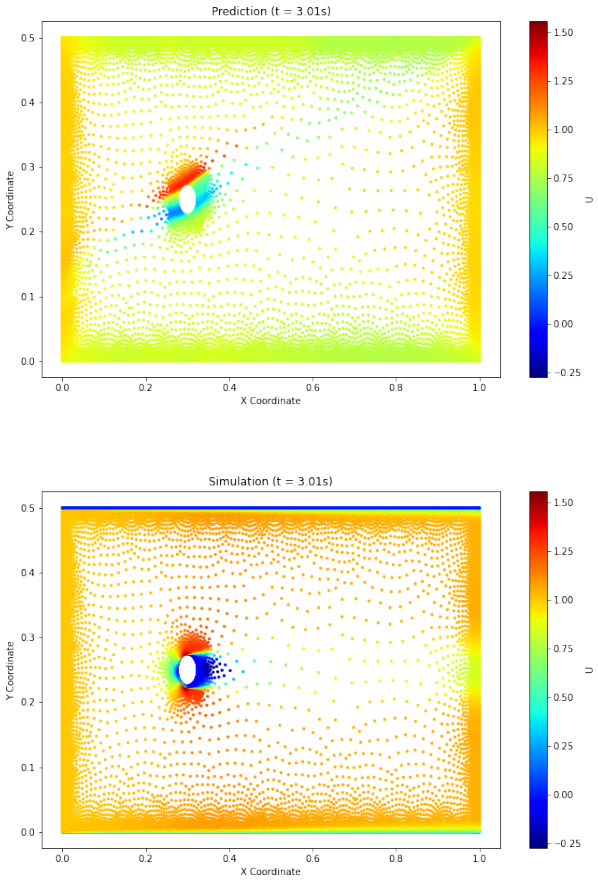


Figure 9: Comparison u

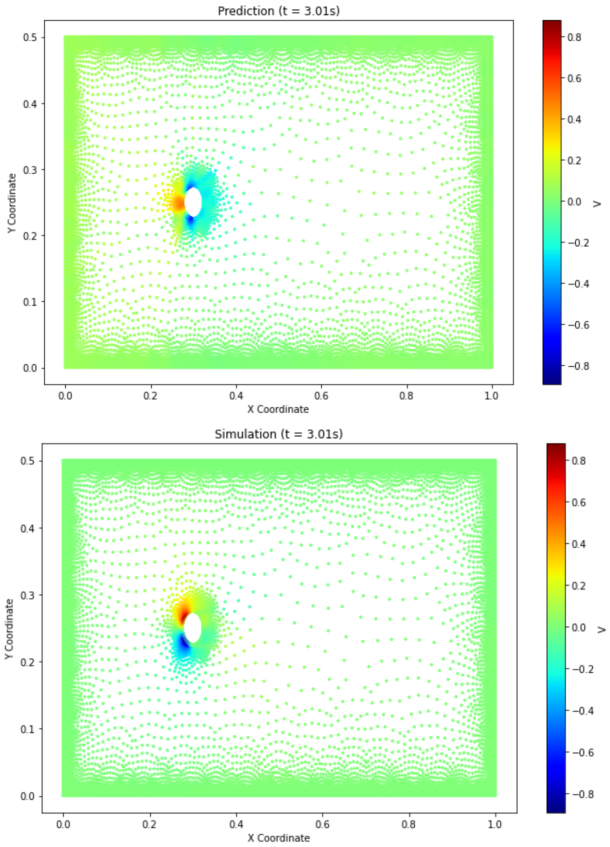


Figure 10: Comparison for v