Proceedings of the 4th International Conference on Fluid Flow and Thermal Science (ICFFTS'23)
Lisbon, Portugal- December 07 - 09, 2023
Paper No. 151
DOI: 10.11159/icffts23.151

Feasibility study of using Machine Learning to accelerate CFD solvers

Sreerag E P[1], Dr. S. Balaji[2]
[1]IIT Madras
Chennai, Tamil Nadu, India
me18b032@smail.iitm.ac.in; sbalaji@iitm.ac.in
[2] IIT Madras
Chennai, Tamil Nadu, India

Abstract - This study investigates the feasibility of using a machine learning model to build a fast computational fluid dynamics (CFD) solver. The goal of this research work is to explore whether a machine learning approach can reduce the computational cost and increase the speed of CFD simulations, which can be computationally expensive and time-consuming. The study will analyze the performance of the machine learning-based CFD solver by comparing it with the traditional CFD solvers on a set of benchmark problems. The findings of this study have the potential to contribute to the development of faster and more efficient CFD solvers, which in turn could have significant implications for a range of engineering applications.

***Keywords*:** Operator Learning, CFD, DeepONet, Accelerated CFD solver.

1. Introduction

The aerodynamics of airfoil is of great practical importance to several military and civilian applications. These applications include sailplanes, propellers, ultralight man-carrying/man-powered aircraft, high-altitude vehicles, wind turbines, unmanned aerial vehicles (UAVs), microAir vehicles (MAVs), etc. Obtaining flow field around an airfoil is an important aspect in order to obtain aerodynamic coefficients due to pressure and skin friction, to study flow separation, transition, wake formation, and vortex interaction. These coefficients and flow features are crucial for designing aerodynamic components such as an aircraft wing, wind turbine blade, and helicopter rotor blade. Conventionally, the flow field over an airfoil is obtained by solving Navier-Stokes (NS) equations on a computational mesh with appropriate boundary conditions. With the rise of high-performance computing and development of efficient numerical methods, the computational time required for a CFD simulation has been reduced to a large extent. Therefore, at present, CFD simulations are often used as computational experiments to perform aerodynamic design and analysis. However, it is still relatively time-consuming in, for example, airfoil optimization and fluid-structure interaction, where there is a requirement of large iterations of flow solutions. In such situations, employing a N-S solver is computationally expensive, and it delays the overall design cycle. Hence, there is a need for developing an efficient and accurate method, which can obtain the flow solutions much faster than the conventional CFD approach. The proposed CFD solver would significantly reduce the time taken for flow simulations over different airfoil geometries and thus it will significantly bring down the cost and time taken for airfoil optimization and fluid-structure interaction.

2. Literature Review

For CFD simulations of Incompressible flows, the treatment of incompressibility introduces a coupling between velocity and pressure, which requires any algorithm to ensure that the velocity field is divergence-free at every time step of the simulation. This is done by iteratively solving for pressure Poisson equation and momentum equation for pressure and velocity respectively till convergence for every time step. This unavoidable step of the pressure Poisson equation can be computationally expensive (it can take up to 90% of the computational effort) since it often relies on complex iterative techniques. There are several iterative methods in literature that are developed to handle the pressure velocity coupling in Incompressible flows. The first (two-dimensional) computational method for modeling incompressible fluids governed by the Navier-Stokes equations is the Marker and Cell (MAC) algorithm developed by Harlow & Welchin (1965). In this paper, they introduce the staggered grid arrangement to address the numerical problems caused by the usual treatment of calculating velocity vectors and other scalar quantities at nodal positions. Harlow and Welch developed an iterative solution to the divergence equal to zero equation when it is evaluated with the time-advanced velocities. Later in 1972, a Semi-implicit method for pressure-linked equations (SIMPLE) was developed by Patankar and Spalding. In 1980 this method was further improved by Patankar to give the SIMPLER algorithm with a faster rate of convergence. Van Doormaal and Raithby developed the SIMPLE Consistent (SIMPLEC) method in 1984. Other pressure algorithms in the literature include the Fractional-step method by Kim and Moin (1985) and Pressure Implicit with Splitting of Operators (PISO) by Issa (1986).

The conventional approach is limited in terms of its speed and efficiency due to its reliance on iterative procedures to solve the Pressure Poisson equation. With the increase in computational power, there has been an increasing amount of research in recent years that focuses on using machine learning techniques to solve problems in science and engineering. There are several ML-based PDE solvers developed in the literature. For the current study, the proposed plan is to use an operator learning-based approach called as DeepONet to learn the solution operator for the 2D Poisson Equation. There are several works of literature that have studied how well DeepONets can approximate linear and non-linear operators. The operators can be of the explicit form (like Laplace transform) or of the implicit form (like the solution operator for PDEs). Since DeepONet directly learns the solution operator for PDEs, it is not required to train it separately for each instance of the PDE (i.e., for different input conditions), unlike other methods like PINNs. It is found that DeepONets provide better generalization compared to other network architectures like FNNs or ResNets. DeepONets are mesh independent and can produce the output function at any required resolution. For problems where input and output functions are in the form of images, CNNs show slightly better performance; however, DeepONets are more flexible than CNNs for unstructured data. Since DeepONets do not have an explicit form of governing equations, they can be used for Multiscale problems where the underlying physics changes with scale and is generally difficult to model.

Studies show that DeepONets perform very well when the testing and training datasets are from the same function spaces, i.e., interpolation. However, they do not perform as well when the testing data is from a different function space, i.e., extrapolation. There are also several works of literature that study physics informed DeepONets, i.e., DeepONets with regularization term that enforces the underlying governing equations. The advantage of physics-informed DeepONets is that they can work well with sparse data, and it is observed that there is a substantial increase in predictive accuracy and better generalization for extrapolation tasks. Other notable techniques for operator learning include Fourier Neural operators, Graph Neural Operators, Low-rank Neural Operators, Multipole Graph Neural Operators, and Reduced Basis Methods; there are several studies that illustrate the effectiveness of each of these techniques.

3. Methodology

The momentum equation provides a transport equation for each velocity component. However, there is evidently no transport equation for pressure. Even though pressure appears in all three momentum equations, there is no obvious way to couple pressure with velocity. This is not an issue for compressible flow as the continuity equation provides the transport equation for density and the energy equation for temperature. Pressure can then be calculated using the density and temperature values with the help of the state equation. However, this is not possible for incompressible flows where the density is constant, and hence, the pressure is not linked to density. This results in a coupling between velocity and pressure which introduces a new constraint on the flow field. It is now required to apply the correct pressure field in the momentum equation so as to get a velocity field that satisfies the continuity equation. This problem is addressed by taking the divergence of the momentum equation and simplifying the resultant expression using the continuity equation, which results in a Poisson equation for pressure. Often an iterative strategy is used to address this pressure-velocity linkage, wherein the solver iteratively solves the momentum and the pressure Poisson equation till the flow field satisfies the continuity equation. The treatment of incompressibility introduces this coupling between velocity and pressure, which requires any algorithm to ensure that the velocity field is divergence-free at every time step of the simulation. The unavoidable step of the pressure Poisson equation can be computationally expensive (it can take up to 90% of the computational effort) since it often relies on complex iterative techniques. Solving the Poisson equation for large problems involving millions of degrees of freedom is only tractable by employing iterative methods. Currently, there are several algorithms to solve for Poisson equation iteratively. However, even the fastest algorithm can take a prohibitively large amount of time to iterate for very large problems. Thus, a technique that speeds up the convergence of these iterative solvers can significantly improve the speed and performance of the current CFD solvers for incompressible flows.

DeepONet is a neural network architecture based on the Universal approximation theorem for operators. DeepONets are trained to learn the entire solution operator for a PDE and have shown promising results in solving a wide variety of equations like anti-derivative operator, diffusion-reaction, advection, diffusion-advection, stochastic ODEs & PDEs, etc. The proposed workflow is to train DeepONet on data generated from a Poisson solver on simple cartesian geometry. DeepONet will be used to approximate the solution operator G: f(x,y) → u(x,y) of the 2D Poisson equation (where u(x,y) represents solution function for corresponding forcing function f(x,y). The trained network can then be used to solve the Pressure-Poisson equation along with the existing iterative solvers, potentially reducing the number of iterations to convergence. The study would then focus on using the proposed solver to solve the Navier-Stokes equation for a Lid-driven cavity problem for varying Reynolds numbers. For the Pressure Poisson solver, the input function to the DeepONet will be the forcing function or the RHS of the pressure-Poisson equation. The solution obtained from DeepONet will be very close to the actual solution, and hence this can be used as an excellent initial guess for the existing iterative solvers. Since solving the equation only involves forward propagation, DeepONets are faster than other solution techniques. Since the network learns the entire solution operator, obtaining a solution of higher resolution comes with no further requirement of re-training the network.

4. Numerical Studies

4.1. Deep Operator Network

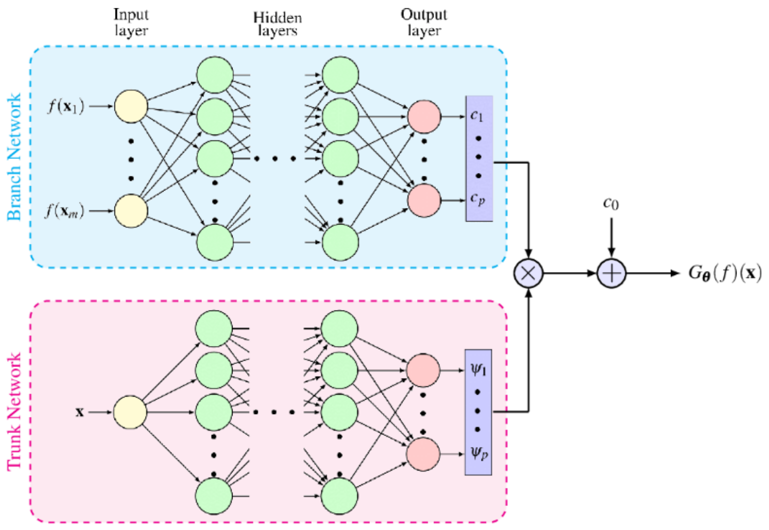


Fig. 1: DeepONet Architecture

The Deep Operator Network is based on the universal approximation theorem for operators and it basically consists of two sub-networks namely the branch net and the trunk net. The branch net is used for encoding the input function at a fixed number of points (called sensors), whereas the trunk net is used for encoding locations for the output function. The final output is evaluated by a dot product of the outputs of branch and truck networks. The basic idea of DeepONet is to approximate an operator i.e. a mapping between two infinite dimensional function spaces. Learning an operator involves learning the relationship between the input function, output function and the independent variables. This relationship of the output function on the input function and independent variables is implicitly represented by the network architecture. Consider an operator G

$$G: f(x,y) \rightarrow u(x,y) \tag{1}$$

Let f1, f2, f3,... fm be the values of input function f(x,y) sampled at sensor locations. The branch net takes in the function sampled at a fixed number of points within the domain called as sensors. It consists of series of layers with transformations and non-linear activation. The branch net encodes the information regarding the dependence of solution function on the input function. The output from the branch net can be represented as b(f1, f2, f3,... fm) where b is the branch. And Trunk net takes in the independent variables x,y represented by t(x, y). Finally the operator G(f(x, y), x, y) is approximated by DeepONet as shown in the equation:

$$G_\theta(f\big((x,y),x,y\big) = \sum_{k=1}^{p} b_k(f_1, f_2, f_3 \dots f_m).\, t_k(x,y) \tag{2}$$

DeepONet can be viewed as a trunk network with weights of the last layer parameterized by another neural network (i.e., Branch net). It is basically a neural network that is conditioned on the input function. Generally, it is observed that incorporating some prior knowledge into the neural network usually results in a better generalization. An operator G(f)(x,y) can be viewed as a function in (x,y) that is conditioned on the input function f, and this information is implicitly encoded within the architecture of DeepONets.

4.2. DeepONet as Poisson Solver

The 2-D Pressure Poisson equation is given by:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = f(x,y) \tag{3}$$

$$\frac{\partial p}{\partial x}_{x=0} = \frac{\partial p}{\partial x}_{x=1} = \frac{\partial p}{\partial x}_{y=0} = 0 \tag{4}$$

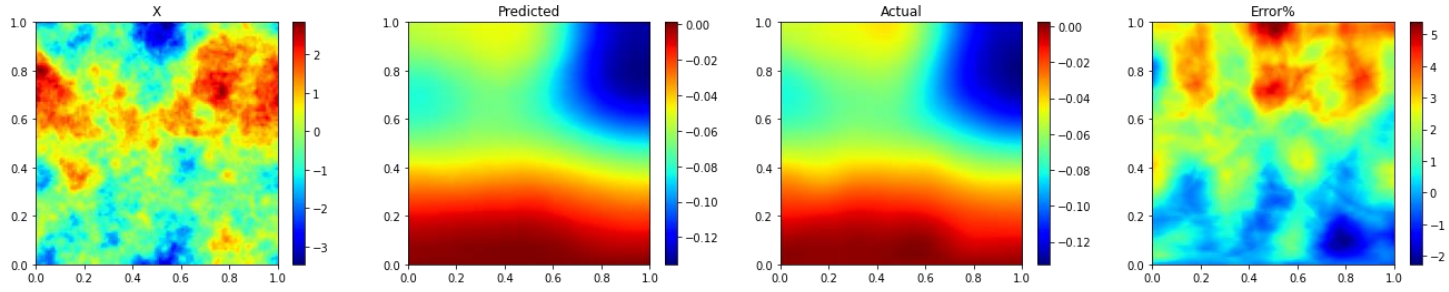
$$p(x,1) = 0 \tag{5}$$

$$\forall x \in [0,1], \forall y \in [0,1]$$

Mathematically the proposed problem is posed as learning a solution operator G that maps from the forcing function to the corresponding latent solution of the underlying PDE.

$$G: f(x,y) \rightarrow p(x,y) \tag{6}$$

It is observed that the Network predictions are within ±5% on the test set GRF Functions. The results obtained from the numerical studies show that DeepONets have sufficient representation capacity to learn solution operator for a Pressure Poisson equation



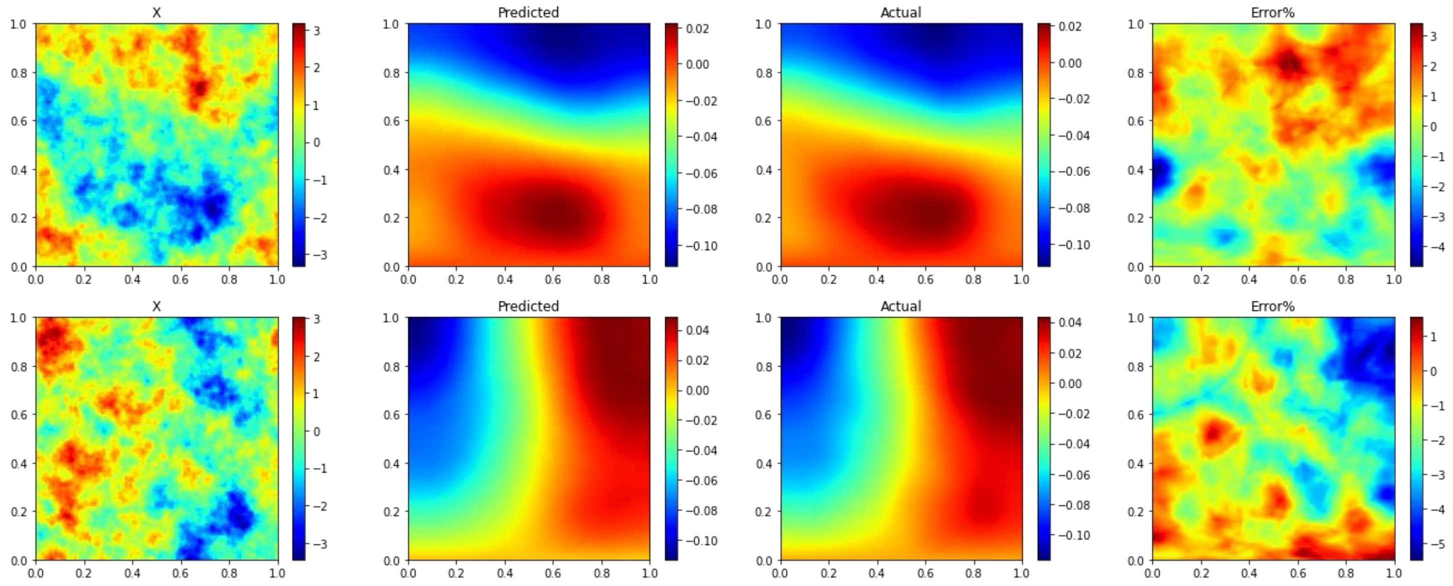


Fig. 2: Results of DeepONet as Pressure Poisson solver.

4.3. DeepONet with CFD Solver

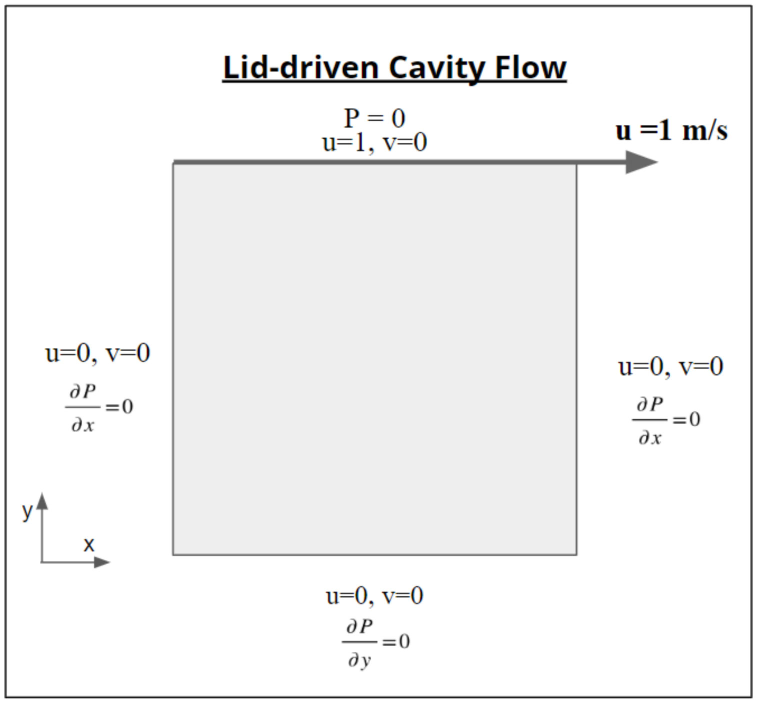
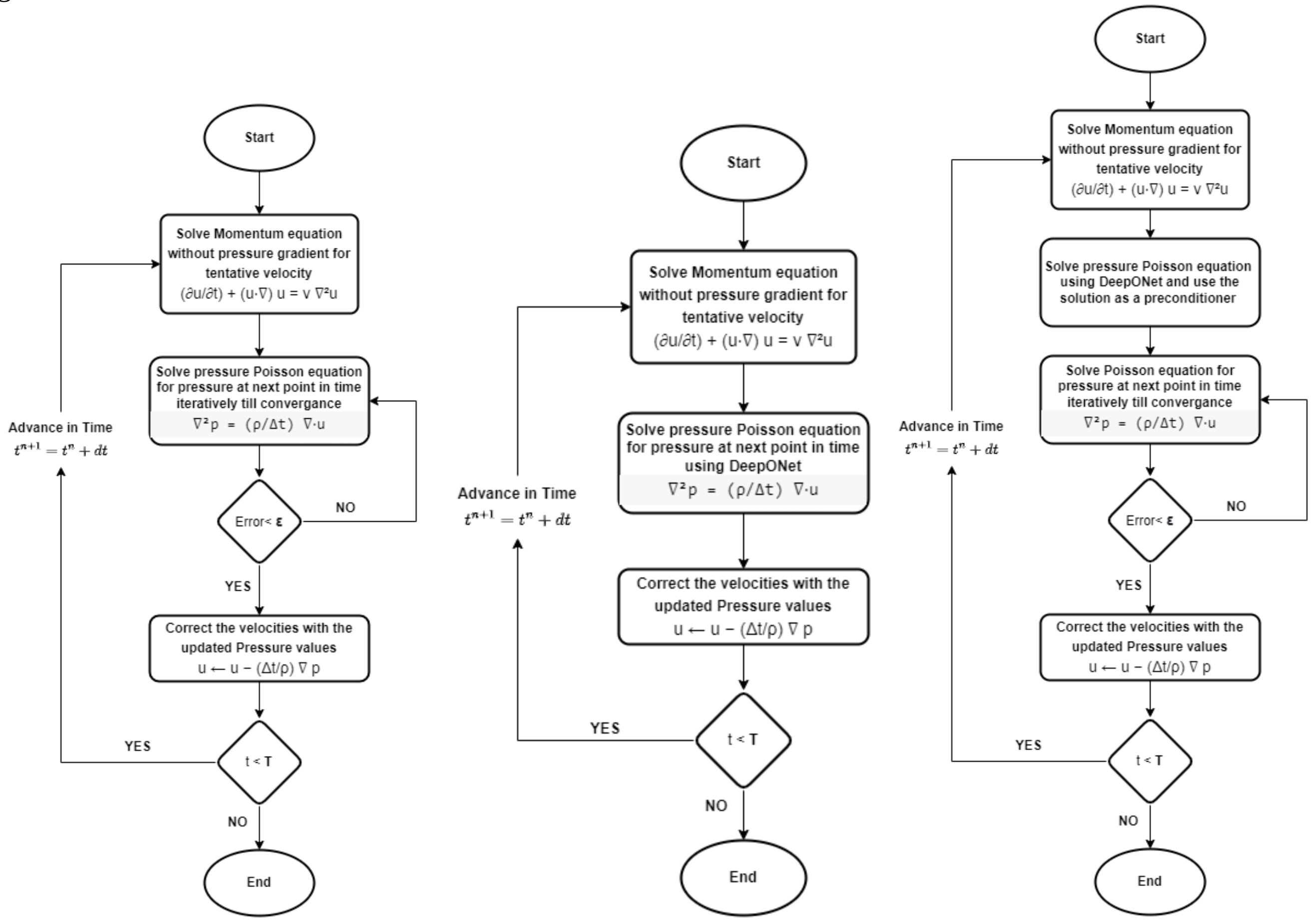

Lid-driven Cavity Flow
P = 0
u=1, v=0
u =1 m/s
u=0, v=0
$\frac{\partial P}{\partial x}=0$
u=0, v=0
$\frac{\partial P}{\partial x}=0$
y
x
u=0, v=0
$\frac{\partial P}{\partial y}=0$

Fig. 3: Lid-driven cavity Problem Setup

Lid-driven cavity flow is a classic problem in fluid mechanics that has been extensively studied over the years and is often used as a benchmark problem for viscous in-compressible fluid flow. It refers to the flow of an incompressible viscous fluid contained in a square cavity with a lid moving at a constant velocity along one of its walls. This problem has important applications in many fields of science and engineering, including heat transfer, lubrication, and materials processing. The lid-driven cavity problem is characterized by a Reynolds number, which is a dimensionless quantity that represents the ratio of inertial to viscous forces. For the lid-driven cavity, the Reynolds number is defined as Re = UL/ν where U is the velocity of the lid, L is the length of the cavity, and ν is the kinematic viscosity of the fluid. The Reynolds number determines the nature of the flow, with low Reynolds numbers leading to laminar flow and high Reynolds numbers leading to turbulent flow.



Start
Solve Momentum equation without pressure gradient for tentative velocity
(∂u/∂t) + (u·∇) u = ν ∇²u
Solve pressure Poisson equation for pressure at next point in time iteratively till convergance
∇²p = (ρ/Δt) ∇·u
Error< ε
NO
YES
Correct the velocities with the updated Pressure values
u ← u − (Δt/ρ) ∇ p
t < T
YES
NO
End
Advance in Time
$t^{n+1} = t^n + dt$
Solve pressure Poisson equation for pressure at next point in time using DeepONet
Solve pressure Poisson equation using DeepONet and use the solution as a preconditioner
Solve Poisson equation for pressure at next point in time iteratively till convergance

Fig. 4: a) Incompressible CFD Solver (Pressure Projection method), b) ML-Based Approach, c) Hybrid Approach

The projection method is an effective means of numerically solving time-dependent incompressible fluid-flow problems. This method was originally introduced by Alexandre Chorin in 1967 and independently by Roger Temam as an efficient means of solving the incompressible Navier-Stokes equations. The key advantage of the projection method is that the computations of the velocity and the pressure fields are decoupled. The algorithm consists of three key steps: First it solves the momentum equation without the pressure gradient term to get a tentative velocity field. Then, it proceeds iteratively solve Poisson equation for pressure at the next point in time using the tentative velocity field. Finally, the velocity fields are corrected using the updated Pressure values.

In this study we tested two different approaches for incorporating DeepONet model with the incompressible CFD solver: a) ML-based approach, b) Hybrid approach.

In ML-based approach the Pressure Poisson equation that arises in each time iteration is solved by prediction from a trained DeepONet. The ML-based accelerated solver gets its speed from removing the time-consuming iterative step of solving the pressure Poisson equation and replacing it with a single forward propagation through a trained DeepONet. The forward pass through the network is very fast and it solves the Poisson equation in one step. However, in the studies conducted it was noted that the accelerated solver diverges after few time iterations (approx. 150 iterations). The solver starts showing some unphysical pressure fluctuations within the domain which eventually causes the solver to diverge.

The ML-based approach failed because the DeepONet predictions on certain pressure fields were wrong and slowly slowly these fluctuations cause the solver to diverge within few iterations. The Hybrid method proposed here is an attempt attempt to address this problem. Here the pressure Poisson is solved by an iterative method however we use DeepONet to to precondition the pressure field, potentially reducing the number of iterative steps required to solve the Poisson equation. equation. Since the predictions from DeepONet would be more or less similar to the actual solution, the number of iterative steps needed to reach convergence would be reduced.

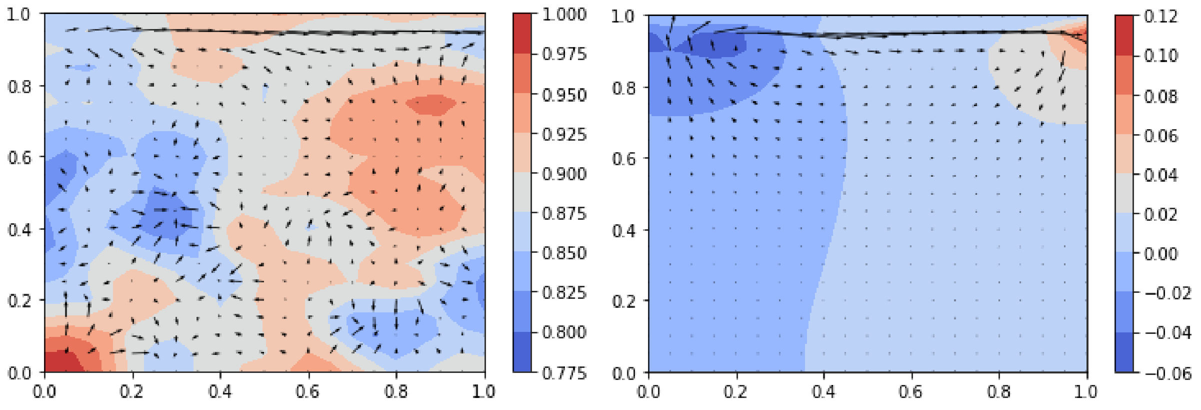
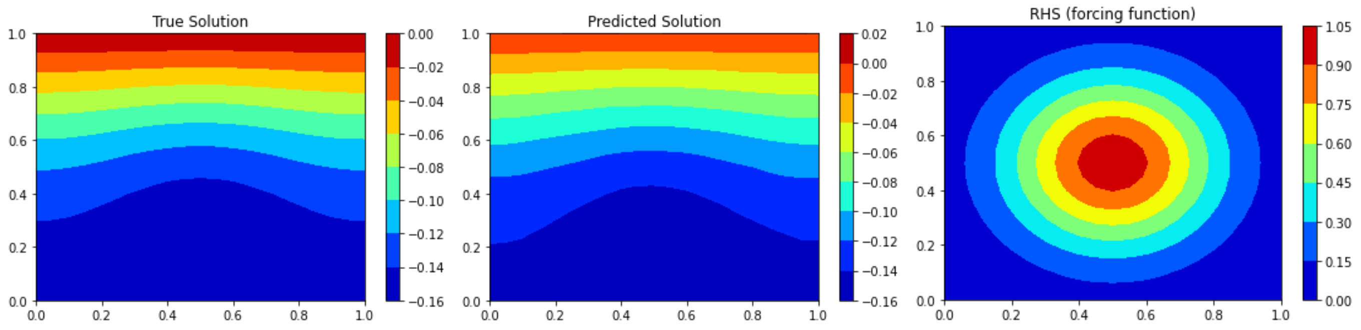


Fig. 5: Pressure Fields from solver after 1.5 secs of simulation

However, contrary to our expectation it was found that the number of iterations required for convergence actually increased as DeepONet was used to precondition the pressure field. The result from the above numerical study suggests that the predictions from the DeepONets were in-fact not helping in speeding up the convergence of iterative solver. Although DeepONet can be used as a Pressure Poisson solver with good enough accuracy, we find that such an ML-based solver does not work well with the iterative algorithms used in CFD solvers. It was found that small errors within the model predictions tends to get amplified over time and the solver eventually diverges

4.4. Limitations of DeepONet



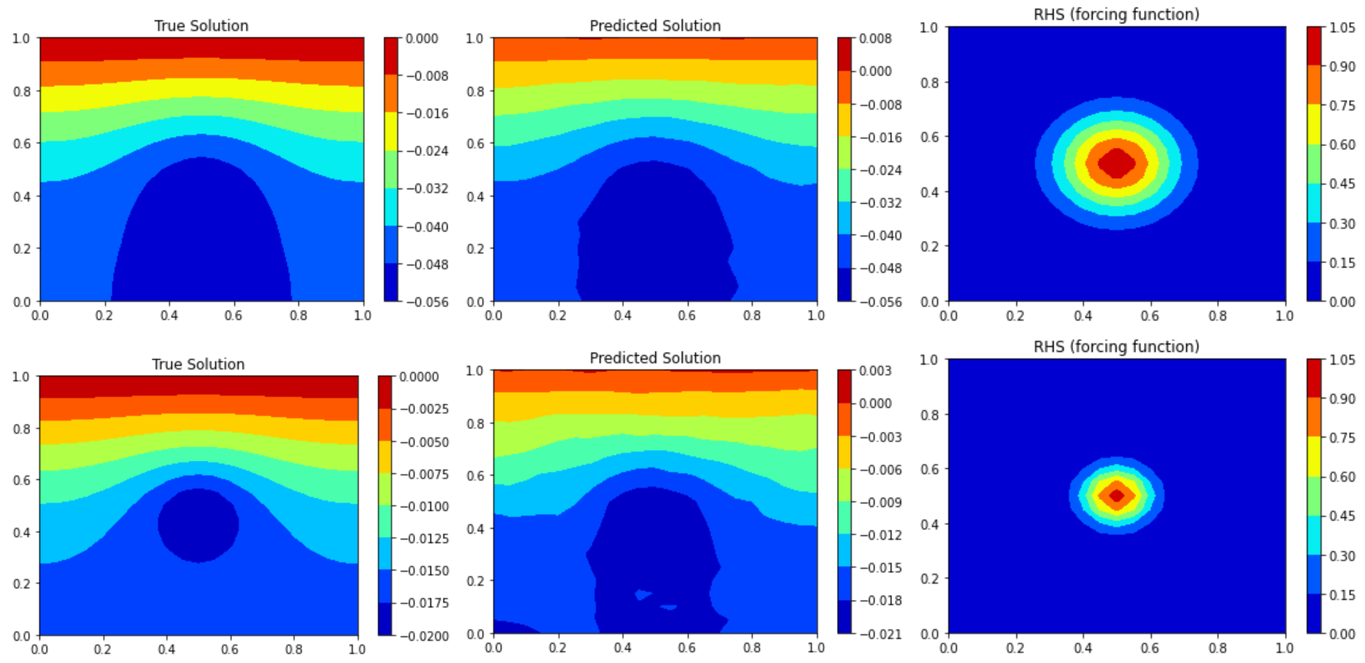


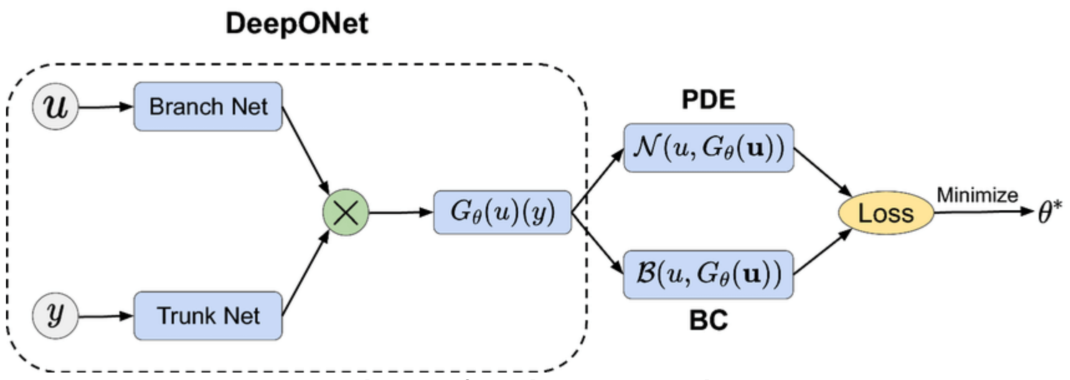
Fig. 6: DeepONet predictions on forcing functions with localised fluctuations.

Upon further numerical investigations on why the DeepONet based CFD simulation fails revealed that the network predictions on certain fields (especially fields with localized fluctuations) are inaccurate. The Network predictions becomes worse as the fluctuations becomes more localized. One contributor to this behaviour might be that the DeepONet takes in this forcing function as an input to branch net at a lower resolution(10x10). This could be causing the network to lose essential information regarding these localized fluctuations which in turn results in bad predictions on such fields.

In order to rectify the above-mentioned limitation, we tried the following methods:

1. Increase the resolution of the input function fed into the branch net. This could potentially solve the problem of bad predictions on fields with localized fluctuations.
2. Incorporate Physics loss into the loss function as it could potentially help the network to generalize.

4.5. Physics Informed DeepONet


DeepONet
u
y
Branch Net
Trunk Net
$G_\theta(u)(y)$
PDE
$\mathcal{N}(u, G_\theta(\mathbf{u}))$
$\mathcal{B}(u, G_\theta(\mathbf{u}))$
BC
Loss
Minimize
θ^*
Fig. 7: Physics Informed DeepONet Architecture

Physics informed DeepONet is a model proposed by Wang et al. in 2021. The model takes its inspiration from the Physics Informed Neural network or PINNs proposed by Raissi et al. The architecture of Physics informed DeepONet is is essentially same as that of a plain DeepONet. The only difference between these two methods is that fact that physics informed DeepONet has an additional term in its loss function called as physics loss. This method is particularly useful in in applications wherein we know the underlying governing equations and lot of data is not available.

Consider an operator G

$$G: f(x,y) \rightarrow u(x,y) \tag{7}$$

Let $G_\theta(f(x,y),x,y)$ represent the DeepONet prediction for a given f(x,y), x, y.

$$G_\theta(f(x,y),x,y) = \sum_{k=1}^{p} b_k(f_1, f_2, f_3 \dots f_m).\, t_k(x,y) \tag{8}$$

where θ denotes the collection of all trainable weight and bias parameters in the branch and trunk networks. These parameters can be optimized by minimizing the following mean square error loss.

$$L = L_{data} + \alpha L_{physics} \tag{9}$$

$$L_{data} = \sum_i (G_\theta(f(x,y),x,y)_i - G(f(x,y),x,y)_i)^2 \tag{10}$$

L_{data} is the same loss function that we use to train a simple DeepONet. For physics informed DeepONet in addition to data loss we have a loss that is calculated based on how much the approximated operator deviates from the known physics/governing equations. The physics loss is essentially the residue loss of the governing equation. As the physics loss reduce, the approximated operator starts following the underlying governing equations.

For example, let us assume that the underlying governing equation is given by a Poisson equation.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \tag{11}$$

Then the physics loss is given by the following relation

$$L_{physics} = \sum_i \left(\frac{\partial^2 G_\theta}{\partial x^2}_i + \frac{\partial^2 G_\theta}{\partial y^2}_i - f(x,y)_i \right)^2 \tag{12}$$

Studies within literature shows that this method can be extremely powerful and, in some cases, it can allow the network to learn the operators without the use of any data, by optimizing the network using physics loss alone.

5. Results and Discussion

Table 1: Summary of Results

Model	Branch Net	Trunk Net	Test Loss	Number of Parameters
DeepONet	[100,100,100,100,100]	[2,100,100,100,100]	1.262e-5	91,201
DeepONet	[2500,100,100,100,100]	[2,100,100,100,100]	2.67e-5	350,201
DeepONet	[2500,1200,600,300,200,100]	[2,100,100,100,100]	2.485e-5	4,030,201
PI-DeepONet	[2500,100,100,100,100]	[2,100,100,100,100]	3.846e-4	350,201

From the series of numerical studies conducted, following observations were obtained. It was found that incorporating physics loss did not result in any further improvement in the network performance. The network performance decreased after introducing physics loss. It was noted that increasing the resolution of the forcing function input to the branch net did not yield better results. Interestingly increasing the resolution, which in effect is providing the network with more information, results in slightly worse performance. As the input function resolution is increased the network size increases which in turn increase the model complexity and the model might be trying to overfit the training data which could be the reason for poor performance. It is also observed that increasing the data also did not produce any improvement in the results. This shows that the lack of data is not the cause of poor generalization of DeepONets. The above results point out to the fact that although DeepONets have shown lot of good results as a PDE solver, it still lacks in terms of generalization across function spaces other than the one it was trained in. Most of the research studies conducted on DeepONet are only limited to testing these networks within the same function space. This feasibility study shows potential benefits of using ML-based techniques to speed up CFD solvers and also opens up several avenues for future research in this area.

5. Conclusion

In conclusion, this study explored the feasibility of using a machine learning model to build a fast computational fluid dynamics (CFD) solver. The results of the study points to the fact that while machine learning models can be effective for interpolation tasks, they struggle to generalize and perform poorly for extrapolation tasks. In the context of operator learning this means that the network would show good performance for predictions on functions that are sampled from the same function space as the training set. This was clearly evident in the numerical studies conducted. It was observed that the model performs very well (<5% error) on fields that are samples from the same function space as the training set (i.e. Gaussian Random Fields). However, the model performance on fields that are from different function spaces (i.e. an extrapolation task) is very poor. This limitation of machine learning models suggests that this field is still in its nascent stage and more research is needed to improve their performance for such applications. The findings of this study have important implications for the development of CFD solvers, as it highlights the need for a hybrid approach that combines the strengths of both traditional CFD solvers and machine learning models to create more accurate and efficient simulations. Overall, this study provides valuable insights into the potential and limitations of machine learning in CFD simulations, and underscores the importance of continued research in this area.

References

[1] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. “Learning nonlinear operators via deeponet based on the universal approximation theorem of operators”. Nature machine intelligence, 3(3):218– 229, 2021.

[2] Wang, Sifan, Hanwen Wang, and Paris Perdikaris. "Learning the solution operator of parametric partial differential equations with physics-informed DeepONets." Science advances 7, no. 40 (2021): eabi8605.

[3] Raissi, Maziar, Paris Perdikaris, and George 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 (2019): 686-707.

[4] Tompson, Jonathan, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. "Accelerating eulerian fluid simulation with convolutional networks." In International Conference on Machine Learning, pp. 3424-3433. PMLR, 2017.

[5] Kovachki, Nikola, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. "Neural operator: Learning maps between function spaces." arXiv preprint arXiv:2108.08481 (2021).

[6] Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations." IEEE transactions on neural networks 9, no. 5 (1998): 987-1000.

[7] Lanthaler, Samuel, Siddhartha Mishra, and George E. Karniadakis. "Error estimates for deeponets: A deep learning framework in infinite dimensions." Transactions of Mathematics and Its Applications 6, no. 1 (2022): tnac001.

[8] Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. "Fourier neural operator for parametric partial differential equations." arXiv preprint arXiv:2010.08895 (2020).