Introduction to TurbFlowPINN: A framework to integrate CFD Simulations with PINNs for Efficient Flow Prediction

A personal project by Bob Bu, sbu1@ualberta.ca

December 2024, Edmonton, AB, Canada

Computational Fluid Dynamics (CFD) is a key tool for analyzing fluid flow in engineering, with applications ranging from aircraft design to biomedical processes. It provides essential insights into flow characteristics such as velocity, pressure, and turbulence, enabling optimization of performance and energy efficiency. However, high-fidelity CFD simulations are computationally demanding, especially for turbulent flows requiring detailed temporal and spatial resolution.

Physics-Informed Neural Networks (PINNs) offer a promising solution to reduce these computational costs. By embedding physical laws like the Navier-Stokes equations directly into the training process, PINNs balance data-driven learning with governing dynamics, allowing them to generalize effectively even with limited or noisy data.

This project develops an end-to-end pipeline that integrates CFD and PINNs for efficient fluid flow prediction. The workflow includes the following steps:

1. **CFD Simulation**: Flow fields are simulated using turbulence model, k-omega SST-IDDES to capture essential variables such as velocity, turbulence energy, and pressure.
2. **Data Processing**: Simulation outputs are extracted and structured into machine learning datasets, retaining relevant spatial and physical parameters.
3. **Model Training**: The PINN is trained on the dataset, combining a data-driven loss with a physics loss to enforce consistency with fluid dynamics equations.
4. **Model Testing**: The trained model is evaluated on a Tesla valve flow scenario, where model predictions will be mapped to real-time visualizations.

This project demonstrates that combining CFD and PINNs can significantly reduce computation times without compromising accuracy, making this approach suitable for real-time predictions and iterative engineering design.

CFD Simulation

I chose to have my own CFD simulation results to collect test dataset and train custom models. To generate the dataset, I used the open-source CFD platform OpenFOAM to simulate a plane landing scenario. This scenario was specifically designed to capture complex flow dynamics, including vortices and turbulence, making it an ideal test case. The setup models a Boeing 737 MAX 8 during its landing approach, configured for touchdown with a simulated wind speed of 67 m/s—typical for this aircraft's landing performance.

The simulation employed the k-omega SST-IDDES turbulence model, which combines Large Eddy Simulation (LES) and Reynolds-Averaged Navier-Stokes (RANS) approaches. This hybrid method balances accuracy and computational efficiency by focusing on surface flow details where LES is computationally expensive, while RANS handles the rest. Pure LES simulations, while highly accurate, would have required an impractical amount of computational resources, especially for a personal workstation. This hybrid approach represents a reasonable engineering compromise.

Even with reduced cell counts, running the simulation for five seconds required 24 hours of core time—an indication of how time-intensive CFD simulations can be. This limitation made generating high-fidelity LES data for training impractical, further justifying the use of IDDES as an efficient alternative.

For the machine learning model, I designed a Physics-Informed Neural Network (PINN) to take spatial coordinates and time as inputs, and output velocity, pressure, turbulence kinetic energy (k), and omega (the rate of dissipation of k into thermal energy). This choice is widely used and well-suited for modeling turbulent flows. Among the models tested: PINN, FNN, and Linear Regression, PINN achieved the best validation loss.

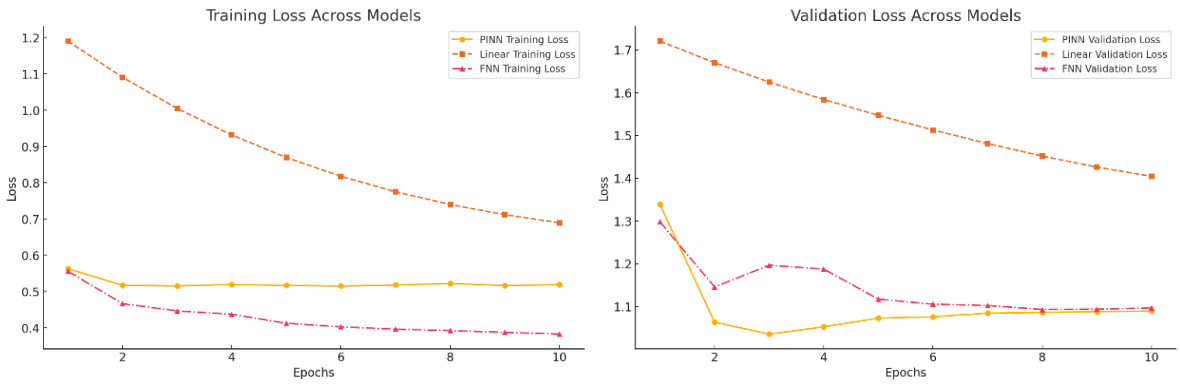

Figure 1 and 2. The training and validation losses for the three models—PINN, Linear, and FNN—across 10 epochs

Since the training data isn't purely derived from a single method like LES, and precision has been intentionally reduced to ensure the simulations can run efficiently on a personal computer, it might seem counterintuitive that the PINN exhibits a higher training loss. However, the validation loss ultimately demonstrates the model's effectiveness, showing that the PINN is the best choice in this scenario. Physics-informed regularization helps the PINN generalize well, making it robust to unseen data and capable of capturing essential physical trends that simpler models fail to represent.

Physics background

With this idea in mind, I can then use PINN for the model schema. First, I need some constraining equations that can capture the physics background of the training data for the best performance. I know that the density of air ρ is 1.225 kg/m^3, which is measured at sea level. Based on this, I derive the dynamic viscosity μ using the relationship:

$$\mu = \rho \cdot \nu$$

where ν is the kinematic viscosity of air at standard conditions, approximately 1.5×10^{-5} m^2/s, this yields $\mu = 1.225 \cdot 1.5 \times 10^{-5} = 1.8375 \times 10^{-5}$ $\text{kg/m} \cdot \text{s}$. These two parameters govern how air behaves as it flows around objects. The Navier-Stokes equations, the backbone of fluid dynamics, ensure the conservation of mass (continuity

equation) and momentum. Additionally, I use the SST k-ω turbulence model [Menter 1993] to approximate chaotic turbulent flows. For simplicity, I directly adopt common values widely used in the CFD community[1]:

$$\alpha = \frac{5}{9}, \beta = 0.075, \sigma_k = 0.85 \text{ and } \sigma_\omega = 0.5$$

Also, I chose to omit the IDDES turbulence model for simplicity, as its hybrid approach involves additional spatial and temporal filtering steps that are unnecessary for our current implementation. Instead, the SST k-ω model provides a practical balance between computational efficiency and accuracy.

Next, consider the flow of air over an airplane wing: as the air moves rapidly, it creates varying pressures on the top and bottom surfaces of the wing. This pressure difference is what generates lift. To ensure the consistency of this flow, I rely on the continuity equation, which states that the amount of air entering a given region must equal the amount leaving it, thereby maintaining a balance. This principle is mathematically expressed as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

Here, u, v, and w are the velocity components of the air in the x, y, and z directions. Then, we need to describe how the air's velocity changes due to forces like pressure and viscosity. For instance, when the air flows around the wing, it slows down near the surface due to friction (viscosity). And this is the momentum equation:

$$\rho \frac{\partial u}{\partial t} + \rho(u \cdot \nabla u) = -\nabla p + \nabla \cdot (\mu \nabla u)$$

Which simply says change in air velocity ($\frac{\partial u}{\partial t}$) depends on forces like pressure (∇p) and viscous effects ($\mu \nabla u$).

Beyond just understanding how the air moves, turbulence adds another layer of complexity. Turbulence is chaotic and unpredictable, but it can be approximated using two additional quantities: k, represents the energy in the turbulent motions; and ω, describes how quickly this energy dissipates into heat. The k equation:

[1] https://www.cfd-online.com/Wiki/SST_k-omega_model

$$\frac{\partial k}{\partial t} + u \cdot \nabla k = P_k - \beta \rho k \omega + \nabla \cdot \left[\left(\mu + \frac{\sigma_k \mu_t}{\rho} \right) \nabla k \right]$$

Governs turbulent energy production and dissipation. P_k is the production of turbulent energy, which depends on how the flow shears and swirls. Similarly, for ω, a related equation tracks its evolution:

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \alpha \frac{P_k}{\nu_t} - \beta \rho \omega^2 + \nabla \cdot \left[\left(\mu + \frac{\sigma_k \mu_t}{\rho} \right) \nabla \omega \right]$$

And this equation governs the rate of dissipation of turbulent kinetic energy. Lastly, all loss terms are computed accordingly.

Data processing and model training

I first preprocess raw CFD simulation data stored in CSV format into HDF5, a more efficient data format. Then it is normalized by computing the mean and standard deviation for both inputs and outputs across the entire dataset.

My dataset is split into training and validation subsets once prepared. The first 80% of the dataset is allocated for training, ensures that the model to learn generalizable patterns. The last 20% of the dataset is reserved for validation, allowing for performance monitoring on unseen data.

Again, the dataset is too large to load entirely into memory for training and evaluation. A practical decision was made to use only a subset of the data for training and validation. This subset is selected sequentially from the grand dataset.

A batch size of 4096 is used for both the training and validation data loaders. This specific batch size was determined through empirical testing, as it strikes the right balance between GPU memory usage, training speed, data fitting and model convergence.

The training process employs several dynamic mechanisms to optimize learning:

1. Learning Rate Scheduler: The learning rate is managed using ReduceLROnPlateau, which reduces the learning rate by a factor of 0.5 whenever the validation loss plateaus.

2. PDE Weight Adjustment: The weight of the physics-informed loss term is gradually increased over epochs, transitioning the focus from fitting data to obeying physical constraints.

At the end of each epoch, the model is evaluated on the validation set to compute the validation loss. This loss is used to save the best-performing model. When the code reloads a previously trained model, it will continue the training or evaluation.

Model Testing

Use trained models to predict fluid flow in a real-world test scenario. The Tesla valve is specifically designed to allow fluid to flow smoothly in one direction while significantly impeding flow in the opposite direction. This unique characteristic makes it an ideal test case to evaluate model performance. Several trained models are provided, each with varying levels of accuracy and capability. This allows you to experiment and compare their performance in this scenario.

Follow the README on the GitHub directory to run and create your own models!

https://github.com/BobOnTheWayCA/TurbFlowPINN.git