
Graph-Driven Surrogate Models for Efficient Fluid Flow Prediction

Nora Ngo

Heinz College
Carnegie Mellon University
Pittsburgh, PA 15213
nngongoc@andrew.cmu.edu

Aditya Sannabhadti

Heinz College
Carnegie Mellon University
Pittsburgh, PA 15213
asannabh@andrew.cmu.edu

Zhuoqi Li

College of Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
zhuoqili@andrew.cmu.edu

Cheick Dosso

College of Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
cdosso@andrew.cmu.edu

Abstract

Accurate and rapid flow field prediction is crucial in many engineering applications, but traditional computational fluid dynamics (CFD) solvers often require a long computational time, particularly for complex geometries [6]. In scenarios like real-time analysis and design optimization, for example, quick access to reliable flow data is essential. Surrogate modeling offers a promising alternative by approximating CFD results with significantly reduced computational cost [7]. Our project proposes developing and testing a graph convolutional network (GCN)-based surrogate model that predicts steady and incompressible 2D flow fields in rectangular channels of different dimensions. By comparing the accuracy of our GCN surrogate with standard CFD results, we aim to demonstrate that our approach can deliver fast and reliable predictions, thereby offering valuable benefits for a wide range of engineering challenges that need real-time design iterations and control of fluid systems. The results presented in this study show that the developed model can extrapolate to unseen rectangular channels while keeping node-wise velocity errors within $\pm 2\%$ of the high-fidelity solution.

1 Overview and Context

Motivation

High-fidelity CFD remains the gold standard for analyzing aerodynamics, cooling channels, and microfluidic devices, but each new geometry can require several hours or days of CPU time with traditional finite-volume or finite-element solvers (methods used to solve CFD simulations) [6]. These long running times prevent rapid prototyping, limit shape optimization, and avoid closed-loop control in settings that require millisecond-level feedback, such as battery thermal management [2] or adaptive wing morphing [4].

Recent work shows that graph-based neural surrogates can reduce the evaluation time by $\mathcal{O}(10^3)$ while keeping errors below a few percent [8, 7]. Deploying such surrogates would unlock the following:

- **Fast design sweeps:** thousands of candidate channels or air-foils evaluated in minutes instead of days
- **Digital twins:** real-time flow estimates streamed into simulators for fault detection and control
- **Lower cost & carbon:** drastic reductions in expensive High Performance Computing usage.

Therefore, we investigate a Graph Convolutional Network that predicts steady, incompressible 2-D channel flows from mesh data, aiming to deliver CFD-level accuracy without the computational overhead.

Objective

To demonstrate that a GCN can learn from a limited set of fluid flow simulations and then generalize to predict velocity fields in unseen or varied geometries with minimal loss of accuracy. Our goal is to limit the error of the expected value to plus or minus 2

2 Literature Review

Traditional CFD approaches, such as Finite Volume (the flow domain is discretized into small 'boxes', and the solver keeps track of how much fluid -mass, momentum, energy- enters and leaves each box so that nothing is lost or created) and Finite Element methods (minimize the residual of the governing partial differential equations over a mesh of interpolation functions), are well-established but become computationally expensive with increased mesh resolution.

Recent advances in deep learning have shown that neural networks can approximate flow fields. Graph-based neural surrogates have emerged as a powerful way to accelerate CFD. Kipf et al. first generalized convolution to irregular graphs, laying the foundation for operating directly on unstructured meshes [5]. Using regular grids, Guo et al. trained a CNN that predicts steady bluff body flow (airflow or fluid flow around objects with blunt, non-streamlined shapes like a circular cylinder, a building or a car) 700 times faster than a finite volume solver (method used in traditional CFD software algorithms) [3].

To enable learning that generalizes across meshes of varying resolution and topology, Chen et al. introduced a Graph Convolutional Network (GCN) that achieves $< 3\%$ mean absolute error on laminar flows (smooth, orderly motion of a fluid in which its layers slide past one another like evenly stacked sheets, without swirling or mixing) around arbitrary 2D shapes, evaluating each case in milliseconds [1].

Most recently, Peng et al. embedded a residual fluid motion governing equation (the Navier-Stokes equation) into a physics-informed GCN (PINN-GCN), achieving mean accuracy over 98% while realizing a speed up $\mathcal{O}(10^3)$ over a finite-volume baseline [7].

Previous studies report two reference approaches: a purely data-driven GCN and a physics-informed GCN, where each is evaluated via node-wise mean squared error (MSE) against CFD solutions.

Rather than blending data and physics terms, we concentrate on the data-driven baseline alone and ask whether a pure GCN can still generalize across rectangular channels of varying aspect ratio, targeting $< 2\%$ MSE while preserving the speed-ups demonstrated in the literature.

3 Methodology

3.1 Data Generation

To build our data set, we run steady-state 2D computational fluid dynamics (CFD) simulations of flow through a rectangular channel, as illustrated in Figure 1. Each simulation provides node positions, mesh connectivity, and velocity values, which form the graph data used by our GCN model. In this initial experiment, we focus on a single channel geometry (fixed height and length) to validate both the feasibility of the overall framework and the accuracy of the GCN predictions.

Figure 1 shows the mesh generated by the CFD model. By refining the mesh, we can increase the number of nodes, thereby creating a denser graph that enriches the dataset. After each simulation, we

compile the coordinate and velocity information for every node; these serve as input features and labels to train our model.

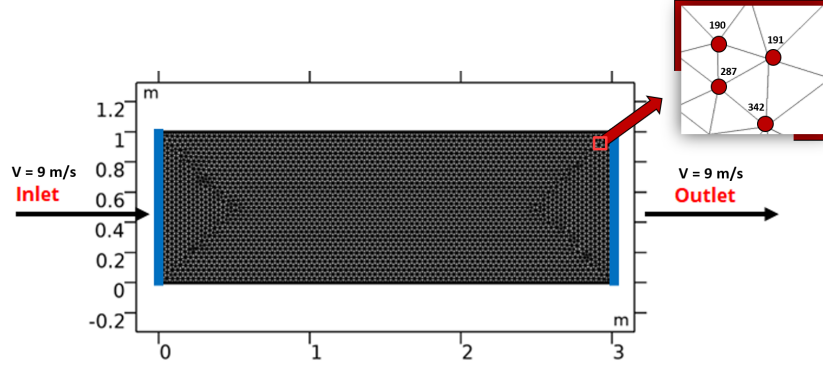


Figure 1: 2D CFD mesh structure and node number obtained through COMSOL.

3.2 Data Preprocessing

After running the CFD simulations, we convert the simulation grid into a graph by treating each point on the grid as a node. This allows us to build a feature matrix in which each row corresponds to a single node, storing the x- and y-coordinate values of that node, along with its corresponding velocity label value in m/s. The process is illustrated in Figure 2.

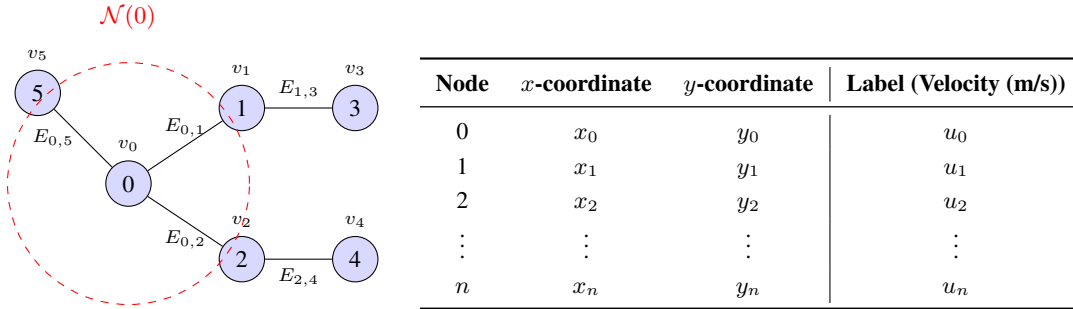


Figure 2: Example of a graph-based mesh and its corresponding feature matrix, including node x-y coordinates and velocity. $\mathcal{N}(0)$ corresponds to the neighbor set of Node 0-w, ch are all the nodes directly connected.

fig:graph-mesh

The CFD simulation will generate two different files to record the information about the mesh nodes and the velocity attribute of each node. By extracting the information from these files, we can divide the content into different CSV files: `Edge_Elements_Data.csv` stores edge connection information between different nodes, `Mesh_Node_From_Mesh_File.csv` stores the coordinates of different nodes, `Output_Var.csv` stores the velocity attribute of each node, and `Triangle_Element_Data.csv` or `Quad_Element_Data.csv` stores the index of nodes that represent the geometric relationship between various nodes.

3.3 Dataset Structure

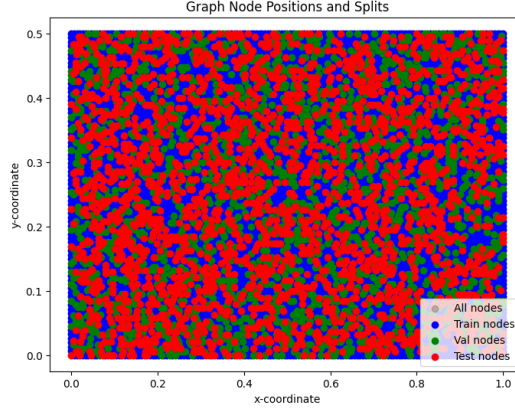


Figure 3: Graph Nodes Positions and Splits

After extracting the simulation results, we combine the mesh connectivity and node information to build our training dataset. First, we use the connectivity of the elements (including edges, triangles, and quadrilaterals) to map out how each node is linked to its neighbors. Next, we merge the node information (which contains the x and y coordinates of each node) with the velocity output from the simulation, producing a table in which each row corresponds to a node and lists the coordinates of that node (x, y) and the velocity.

We treat the coordinates (x,y) as our input features and the velocity at each node as the label (the target variable). Finally, because we have a single graph, we split its nodes into training, validation, and test sets by randomly selecting a subset of nodes for each - the process is illustrated in Figure 3. This ensures that each subset of nodes is used exclusively for training, hyperparameter tuning, or final evaluation.

3.4 Model Architecture

Table 1: Model Architecture

Block	Layers	Hidden Size
Block-1	6x{GCN, Norm, tanh}	HiddenSize {5, 128, 128}
Block-2	6x{GCN, Norm, tanh}	HiddenSize {128, 256, 512}
Block-3	6x{GCN, Norm, tanh}	HiddenSize {512, 1024, 2048}
Block-4	6x{GCN, Norm, tanh}	HiddenSize {2048, 2048, 2048}
Block-5	6x{GCN, Norm, tanh}	HiddenSize {2048, 2048, 2048}
Block-6	6x{GCN, Norm, tanh}	HiddenSize {2048, 1024, 512}
Block-7	6x{GCN, Norm, tanh}	HiddenSize {512, 256, 128}
Block-8	6x{GCN, Norm, tanh}	HiddenSize {128, 64, 32}
LinearOutput	1x{LinearLayer}	HiddenSize {32, 1}

The structure of the model is based on the model proposed by Kipf and Welling[5]. Similarly, we also used critical GCN convolutional layers, regularization layers, and Tanh activation functions, but we increased the depth of this model compared to the original model and used blocks to encapsulate these layers so that we could better adjust the parameters. As Table 1 shows, this model is composed of a total of eight blocks and one linear layer. In each block, we used three different layers: GCN convolution layer, normalization layer, and Tanh activation layer. These three layers are used together

as a group and will be reused six times in one block. The parameters of each GCN layer will depend on the preset parameter, which can be modified by the user.

For the parameters we used, we initially set the input feature dimension to 5, because our dataset has a total of 5 features: position x, position y, the length of the graph, the width of the graph, and the binary symbol "is edge". For the feature "is edge", if the node is on the boundary of the graph, the value will be set to 1. Otherwise, the value will be set to 0. Meanwhile, in addition to features, the model also requires the edge index matrix, which shows the connectivity of the nodes on a graph. Based on the trade-offs among the training duration, model effect, and model depths, we chose to use the hourglass architecture for the model. With an increase in the number of layers, the parameters of the GCN convolutional layer gradually increase from 5 to 2048 in the first block to the third block. The fourth and fifth blocks will continue to use 2048. Finally, from the sixth block to the eighth block, the feature dimensions gradually decrease from 2048 to 32. Also, since we need to predict the velocity in the end, we have to use a linear layer to set the output feature dimension to 1.

3.5 Evaluation Metric and Loss Calculation

For evaluation indicators and standard functions, our ultimate goal is to predict the speed of a certain position. The Mean Square Error (MSE) is used to verify the overall performance of the model. As a goal, we need to lower the MSE value so that the prediction results are as close to the label as possible.

The Mean Squared Error (MSE) is widely used in regression tasks due to its mathematical simplicity and strong penalization of larger errors. It is defined as:

The metric used for the evaluation was MSE. The given metric is especially good and is utilized for the prediction task, which in the given case is being used to predict the velocity of the given point of the nodes. Some more metrics were hypothesized to be effective, RMSE and MAE.

Table 2: Comparison of regression loss functions for CFD task

Loss Function	Formula	Key Property
MSE	$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	Penalizes large errors heavily
MAE	$\frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $	Less sensitive to outliers
RMSE	$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$	Same units as target, interpretable

The reason for using MSE over the two other evaluation criteria is that in a CFD solver-type neural network, RMSE is computationally intensive, and MAE is not suitable for such physics-driven tasks.

The model is trained to minimize this loss using the Adam optimizer, an adaptive gradient descent algorithm known for fast convergence and robustness across a wide range of deep learning tasks. The optimizer updates the parameters θ according to:

Table 3: Comparison of Adam and AdamW Optimizers for CFD Task

Optimizer	Formula	Key Property
Adam	$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$	Efficient for simple tasks with limited regularization needs, well-suited for iterative experiments
AdamW	$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} - \lambda \cdot \theta_t$	Includes weight decay for regularization, more suitable for complex models with larger datasets and prone to overfitting

For our GCN-based CFD surrogate model, the Adam optimizer is preferred over AdamW because of several factors aligned with our task requirements. The fluid velocity prediction task is a straight-forward regression problem with clean, physics-based data from simulations. Given the relatively small meshes (about 15,000 nodes per channel) and limited features (only x, y coordinates), the risk of overfitting is minimal, making regularization in AdamW unnecessary. Adam’s computational efficiency suits our iterative research approach, enabling quick experimental iterations and rapid convergence to validate our basic approach before scaling to more complex geometries or features. Additionally, Adam requires less hyperparameter tuning than AdamW, allowing us to focus on model architecture and feature selection. The regularization provided by Adam aligns better with the physics-driven nature of our problem, as the CFD-generated data already incorporates physical constraints like no-slip boundary conditions, making AdamW’s decoupled weight decay potentially counterproductive. If future work extends to more complex geometries or larger models, AdamW may become more appropriate.

4 Baseline and Extensions

4.1 Baseline Choice

The baseline for the given paper is to achieve a basic working prototype Graph Convolutional Network (GCN) computational fluid dynamics (CFD) solver to simulate fluid flows in 2D geometries. This choice is based on its credibility, widespread adoption in the field, and strong performance documented in previous studies [7]. Moreover, the CFD solver can provide a robust, physics-informed benchmark against which we can assess the performance of surrogate models, such as our neural network.

The Mean Squared Error (MSE) was chosen because it provides a direct measure of prediction accuracy and emphasizes larger deviations, which are critical in fluid dynamics applications where small changes can have significant physical implications. This is important in scientific applications where precision is critical. Additionally, we used the Adam optimizer for training due to its adaptive learning rate and efficiency in converging on optimal solutions, especially in scenarios involving noisy gradients or sparse data. In general, this combination balances the expressiveness, efficiency of training, and interpretability of the model, which makes it suitable for our regression-based mesh node prediction task.

4.2 Baseline Evaluation

To rigorously assess the performance of our surrogate model, we adopt as a baseline a data-driven Graph Convolutional Network (GCN) trained to predict node-wise velocity fields from mesh-based CFD data, following established practice in the recent literature.[7, 1] The evaluation protocol involves randomly splitting the nodes of each mesh into training sets (80%), validation (10%), and test (10%) to ensure impartial assessment and prevent data leakage.

The accuracy of the model is quantified using the mean squared error (MSE) between predicted and reference CFD velocities, both as the loss function during training and as the primary evaluation metric on the test set. This choice reflects the importance of penalizing larger errors in fluid dynamics

applications, where local inaccuracies can propagate and compromise physical fidelity. The baseline GCN is considered successful if it achieves an average node-wise MSE within ± 2 of the high-fidelity CFD solution in all test cases, according to the benchmarks reported in previous work. Results are reported as the average MSE test over multiple geometries, supplemented by qualitative comparisons of predicted and true velocity fields to highlight model strengths and limitations. This evaluation framework provides a transparent and reproducible standard for comparing surrogate models in data-driven fluid simulation.

5 Results and Analysis

5.1 Training Process

In the training process, since we hope that the model can generalize images of different sizes, we will load different pictures as datasets for training. During data pre-processing, each loaded image will be processed using the same methods to obtain the feature index matrix and the edge index. After the model receives these two inputs, we will train the model for 100 epochs. Meanwhile, Adam optimizer and ReduceLROnPlateau scheduler will be used during the training process to help reduce the loss value. For the loss function, since we need to predict the velocity corresponding to each node, we adopt MSE as the criterion for evaluating the error between the predicted value and the actual value.

During the training process, we faced a challenge at first. Due to the limited GPU memory, when we trained five pictures, the problem of CUDA memory allocation errors often occurred. Therefore, we have to train the pictures separately. After the model has completed 100 epochs of training on an image, we save the weights of the model. Subsequently, after the new dataset is loaded, we will reload the weights of the model to continue training the model.

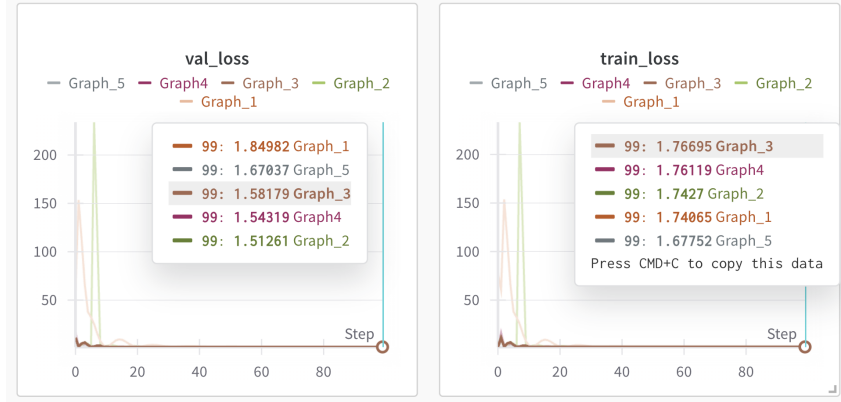


Figure 4: Train Loss and Valid Loss

For example, as shown in Figure 4, our model was trained using five graphs with different lengths and widths. The X-axis of the two plots represents the number of training epochs, and the Y-axis represents the MSE loss value corresponding to the current epoch. According to the results, after training each image, the corresponding training and validation MSE loss values were approximately 2. Meanwhile, as the model trained on more and more pictures, the value of the MSE validation loss gradually decreased from nearly 1.8 to a value between 1.5 and 1.6. For each graph's dataset, we used 80% of the total data in the training process and 10% of it in the verification process. For the last 10 percent, we will use it to test and verify the performance of the model in the current training stage.

5.2 Test Process

Table 4: Test Results

Test Results After Training : MSE	Test Results Of A Whole Graph: MSE
Graph-1: 1.47377	(Height:1,Length:4):1.72215
Graph-2: 1.67772	(Height:1,Length:3):1.71793
Graph-3: 1.71465	(Height:0.25,Length:3):1.75324
Graph-4: 1.34569	
Graph-5: 2.03928	

According to Table 2, it shows the test results of the model during and after training. As mentioned before, we used ten percent of the dataset corresponding to each graph for testing to ensure the effectiveness of the model learning in the training process. The test results are shown on the left side of the table. Although the test MSE of different pictures fluctuates, the overall average MSE can be maintained at around 2.

After completion of the training, we also selected three graphs of different sizes to test the model. To ensure the stability of the test, this part will be tested using all nodes in the graph. After conducting three rounds of tests, the results show that the model’s predicted MSE for each of these graphs is around 1.7. Therefore, the overall MSE of the model was ± 2 in this test.

5.3 Discussion

Our results demonstrate that a purely data-driven Graph Convolutional Network (GCN) surrogate model can achieve node-wise velocity errors within ± 2 of high-fidelity Computational Fluid Dynamics (CFD) solutions. This is a significant achievement, as it highlights the potential of GCN-based models to provide near-CFD accuracy with dramatically faster inference times, making them suitable for real-time design optimization and rapid prototyping in engineering workflows. However, the success of the model is highly sensitive to the quality, diversity, and resolution of the training data. For instance, limited geometric variation or coarse meshes degrade generalization, particularly near domain boundaries. Moreover, the lack of explicit physical constraints, such as the no-slip boundary condition, initially led to substantial edge errors. Enforcing zero velocity at boundary nodes post hoc helped mitigate this issue, emphasizing that the reliability of data-driven surrogates critically depends on physically relevant training signals and domain-aware preprocessing.

6 Future Directions

While the present study shows that a purely data-driven GCN can approximate laminar velocity fields in simple 2-D rectangular channels, several limitations must be addressed before the surrogate can replace a conventional CFD solver.

First, we do not yet enforce the no-slip condition at the wall boundaries, so the network often predicts non-zero wall velocities, which increases the mean squared error. Incorporating a prior physical feature, such as an additional loss term that penalizes the speed of the boundary node, or a hybrid PINN–GCN layer that analytically satisfies "velocity = 0 m/s" will be our next step.

Second, the mesh contains no internal obstacles, while we know that practical channels include cylinders, bends (90° turns), or ribs (small ridges/fins) whose wakes dominate the flow. We plan to augment the COMSOL data with obstacle cases and supply each node with a 'solid' flag (as a feature), for example, so the GCN can generalize to arbitrary layouts.

Third, the model is trained only on a laminar run, while most engineering flows are turbulent (much more complex flow patterns). To handle such systems, we can first generate training data by solving turbulent-flow CFD simulations. Then, instead of the current GCN layers that only look directly around one node, we can switch to attention-based graph layers that let each node 'see' and weigh information from distant neighbors [8]. This should help the surrogate to pick up the larger-scale eddies that drive turbulence.

Fourth, we used a single grid resolution automatically generated by COMSOL (CFD software). Training in multiresolution meshes, together with graph pooling/unpooling, will make the network consistent across coarse and fine discretizations.

Fifth, the work is strictly two-dimensional, and extending the current study would require generating data from 3D CFD models.

Beyond these modeling issues, our outputs are limited to velocity. Other quantities, such as pressure, wall shear, and other derived forces, are equally important for design optimization and must be added as additional prediction channels.

Finally, the geometric parameter space explored so far is narrow. The channel length varied only between 1 and 4 m, and the height between 0.25 and 1 m. Expanding the training set to cover a wider range of aspect ratios, inflow conditions, and fluid properties will test the true generality of the surrogate.

Addressing these items should push the model toward engineering-grade fidelity while retaining fast inference for real-time control and optimization.

7 Conclusion

Our objective was to show that a purely data-driven GCN, trained in a very limited set of CFD simulations, can extrapolate to unseen rectangular channels while keeping node-wise velocity errors within ± 2 of the high-fidelity solution. The experiments met that goal. A surrogate, trained on a single reference mesh, can predict the velocity field in areas of the domain that were never shown during training, and the model was then successfully improved to predict velocities in channels of different dimensions. In these geometries, the network attains a mean-squared error of 1.8, thereby satisfying the < 2 target. This result demonstrates a full surrogate workflow that can be used in real-time design or optimization loops. In general, we confirm that graph-based neural surrogates are a practical alternative to expensive CFD for steady-state laminar channel flows and provide a solid foundation for future work on turbulence, pressure prediction, and more complex 2-D/3-D geometries.

8 Team Contributions

- **Nora:**
 - Testing to implement physics-based loss functions.
 - Deployment, training, and testing of the model.
- **Zhuoqi:**
 - Supplement the data features in the data preprocessing.
 - Deployment, training, and testing of the model.
 - Adjust and revise data preprocessing and training parts to guarantee the compatibility of the project running.
- **Aditya:**
 - Data conversion from raw COMSOL CSV data to processed CSV data.
 - Testing to implement Navier Stokes Loss for GCN.
 - Testing to implement PINNs.
- **Cheick:**
 - Built COMSOL CFD model and exported reference data.
 - Wrote preprocessing scripts and custom Dataset class for PyG.
 - Implemented the initial baseline GCN surrogate.
 - Provided fluid-dynamics expertise to align ML choices with physics.

References

- [1] J () Chen, E Hachem, and J Viquerat. “Graph neural networks for laminar flow prediction around random two-dimensional shapes”. In: *Physics of Fluids* 33.12 (2021), p. 123607. ISSN: 1070-6631. DOI: 10.1063/5.0064108. URL: <https://doi.org/10.1063/5.0064108>.
- [2] Quanyi Chen et al. “A fast thermal simulation and dynamic feedback control framework for lithium-ion batteries”. In: *Applied Energy* 350 (2023), p. 121739. ISSN: 0306-2619. DOI: <https://doi.org/10.1016/j.apenergy.2023.121739>. URL: <https://www.sciencedirect.com/science/article/pii/S0306261923011030>.
- [3] Xiaoxiao Guo, Wei Li, and Francesco Iorio. “Convolutional Neural Networks for Steady Flow Approximation”. In: 2016, pp. 481–490. DOI: 10.1145/2939672.2939738.
- [4] Simon Luis Jeger et al. “Adaptive morphing of wing and tail for stable, resilient, and energy-efficient flight of avian-inspired drones”. In: *npj Robotics* 2.1 (2024), p. 8. ISSN: 2731-4278. DOI: 10.1038/s44182-024-00015-y. URL: <https://doi.org/10.1038/s44182-024-00015-y>.
- [5] Thomas N. Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2017), p. 2907. arXiv: 1609.02907.
- [6] Tianyu Li et al. “Predicting unsteady incompressible fluid dynamics with finite volume informed neural network”. In: *Physics of Fluids* 36.4 (2024), p. 10050. ISSN: 10897666. DOI: 10.1063/5.0197425. arXiv: arXiv:2309.10050.
- [7] Jiang-Zhou Peng et al. “Data and physics-driven modeling for fluid flow with a physics-informed graph convolutional neural network”. In: *Ocean Engineering* 301 (2024), p. 117551. ISSN: 0029-8018. DOI: <https://doi.org/10.1016/j.oceaneng.2024.117551>. URL: <https://www.sciencedirect.com/science/article/pii/S0029801824008886>.
- [8] Tobias Pfaff et al. *Learning Mesh-Based Simulation with Graph Networks*. 2020. DOI: 10.48550/arXiv.2010.03409.

GitHub

https://github.com/AlanLiZQ6/CMU_11785_Project_IDL#