A Comprehensive Two-Week Project Plan for Developing Graph Neural Network Surrogates for Aerodynamic Analysis

Week 1: Foundations, Data Processing, and Baseline Modeling

The first week is dedicated to establishing a solid foundation for the project. This includes setting up a reproducible environment, gaining a deep, intuitive understanding of the dataset through rigorous exploratory analysis, building a robust data processing pipeline, and implementing a baseline model. This baseline will serve as a critical scientific control against which all subsequent, more advanced models will be measured.

Day 1: Project Scoping and Environment Setup

Objective: Establish a robust and reproducible development environment and gain a comprehensive understanding of the AirfRANS dataset's nuances to finalize the project scope.

A meticulously planned project begins with a well-defined scope and a stable development environment. This initial phase is not merely administrative; it is a strategic step to mitigate future technical debt and ensure that the project's goals are aligned with the capabilities and limitations of the chosen dataset. The focus of this day is to move from abstract goals to a concrete, actionable plan grounded in the specifics of the AirfRANS dataset.

Environment Configuration

A reproducible environment is the cornerstone of credible machine learning research and development. It ensures that results can be independently verified and that the project can be seamlessly transferred between different machines or collaborators.

A dedicated environment will be configured using Conda or Python's native venv. This isolates project dependencies, preventing conflicts with other projects. The following core libraries will be installed:

- **Deep Learning Framework:** PyTorch will serve as the primary deep learning framework. Built upon PyTorch, PyTorch Geometric (PyG) will be installed as the specialized library for implementing Graph Neural Networks. PyG offers a vast, well-maintained collection of GNN layers, operators, and benchmark datasets, making it an ideal choice for this project.[1] Essential PyG dependencies, including
  torch-scatter and torch-sparse, will also be installed to enable efficient sparse matrix operations on CPU and GPU.[1]
- **Scientific Computing and Visualization:** A suite of libraries for data manipulation and visualization is required. This includes numpy for numerical operations and scikit-learn for standard machine learning utilities like data scaling and evaluation metrics. Crucially, **PyVista** will be installed for advanced 3D mesh analysis and visualization. The AirfRANS documentation explicitly recommends PyVista for manipulating and visualizing the raw simulation data, highlighting its suitability for handling the unstructured CFD meshes central to this project.[2]
  matplotlib will be used for generating 2D plots and graphs.[5]
- **Dataset Helper:** The airfrans Python library, provided by the dataset creators, will be installed. This library contains utility functions for programmatically downloading and loading the dataset, which is a best practice for reproducibility.[6]

Finally, a standardized project directory structure will be created. This organization, common in professional ML projects, enhances clarity and maintainability [8]:

- data/: For storing raw and processed datasets.
- notebooks/: For exploratory data analysis and results visualization.
- src/: For all source code, including data processing, model definitions, and training scripts.
- models/: For saving trained model checkpoints.
- results/: For storing quantitative results, plots, and animations.

In-Depth Documentation Review

A thorough understanding of the dataset's provenance, structure, and intended use is critical before any code is written. This involves a deep dive into the official AirfRANS paper and its accompanying documentation.[2]

Key areas of focus during this review include:

- **Machine Learning Tasks:** The AirfRANS dataset was designed with four specific generalization challenges in mind: a 'full' data regime (800 training samples), a 'scarce' data regime (200 training samples), a 'reynolds' extrapolation task, and an 'aoa' (angle of attack) extrapolation task.[6] For this portfolio project, the **'scarce' data regime** will be selected. This choice strategically frames the project's narrative. Instead of merely demonstrating the ability to train a GNN, it showcases the ability to develop a model that learns effectively from limited data—a common and critical challenge in scientific and engineering domains where data acquisition (i.e., running high-fidelity simulations) is prohibitively expensive.[10]
- **Data Structure:** The pre-processed dataset provides each simulation as a point cloud. Each point (node) is described by 7 input features: 3D position (though z is constant for these 2D simulations), the two components of the inlet velocity, the distance to the airfoil surface, and the two components of the surface normal vector (zero for non-surface points).[6] The target for regression consists of 4 components: the two components of the velocity field, the pressure divided by the fluid density, and the turbulent kinematic viscosity.[6] Understanding the physical meaning and units of each feature is essential for correct feature engineering and normalization.
- **Pre-processing:** The documentation details the steps taken to convert the raw OpenFOAM simulation output into the provided pre-processed files, including clipping the domain and slicing the 3D mesh to 2D.[11] This information is valuable for understanding the spatial boundaries of the problem and any transformations already applied to the data.

Day 2: Dataset Procurement and Exploratory Data Analysis (EDA)

Objective: Download the dataset and perform a comprehensive EDA to build a strong intuition for the mesh structures, flow field distributions, and graph properties.

EDA is a critical risk mitigation strategy. A common failure mode in machine learning projects is the late discovery of data quality issues or distributional shifts that invalidate modeling assumptions. By proactively investigating the data, potential problems such as inconsistent mesh resolutions or extreme outliers in target values can be identified and addressed early,

de-risking the entire project and demonstrating a rigorous, foresightful approach.

Dataset Procurement

The dataset will be acquired programmatically to ensure the process is repeatable.

- The airfrans.dataset.download() function will be used to download the pre-processed version of the dataset into the data/ directory.[6]
- Subsequently, airfrans.dataset.load() will be used to load the training and test splits for the chosen 'scarce' task into memory as NumPy arrays.[6] The dimensions and data types of the loaded arrays will be verified to ensure the data has been loaded correctly.

Mesh and Geometry Analysis (Qualitative EDA)

Visual inspection of the input data is the most effective way to build an intuition for its structure. A Jupyter notebook (notebooks/01_EDA.ipynb) will be created for this purpose.

- Several diverse simulation samples will be selected from the training set.
- For each sample, **PyVista** will be used to load the point cloud and render it as a 2D mesh. PyVista is the ideal tool for this task, as it is specifically designed for visualizing unstructured grids and is the recommended tool in the dataset's own documentation.[2] Tutorials and examples for PyVista are readily available to guide this process.[13]
- The visualizations will focus on identifying variations in mesh density, particularly around the airfoil's leading edge, trailing edge, and in the wake region. The points can be colored by features like wall distance to visually confirm that the geometry and key physical regions are correctly represented. This qualitative analysis helps answer questions like: "Is the mesh resolution consistent across different airfoil shapes and angles of attack?" Significant variations could pose a generalization challenge for the GNN.

Flow Field Analysis (Quantitative EDA)

Complementing the qualitative analysis, a quantitative study of the target variables will be performed.

- Using the entire training set, histograms and box plots for each of the four target fields

(velocity-x, velocity-y, pressure, turbulent viscosity) will be generated using matplotlib and seaborn.[5]
- These plots will be inspected for signs of skewed distributions, outliers, or multi-modal patterns. For instance, a bimodal pressure distribution might indicate the presence of distinct flow regimes (e.g., attached vs. separated flow) within the dataset.
- Key descriptive statistics (mean, standard deviation, minimum, maximum) will be calculated and recorded for each target field. These statistics are not just for reporting; they are essential inputs for the data normalization step on Day 3.

Graph Property Analysis

As a preliminary step towards graph construction, the basic properties of the graphs that will be generated from the meshes will be analyzed.

- A single representative mesh sample will be converted into a graph structure. This can be achieved by applying Delaunay triangulation to the mesh points to define edges between adjacent nodes.
- For this sample graph, fundamental graph statistics will be computed: the total number of nodes and edges, the average node degree, and the distribution of node degrees. This analysis provides an early estimate of the computational load for the GNN's message-passing steps and informs the design of the model architecture.

Day 3: Graph Construction and Physics-Informed Feature Engineering

Objective: Convert the raw point cloud data into a graph dataset suitable for PyTorch Geometric and engineer rich, physically meaningful features.

The process of converting a mesh into a graph is not merely a technical data transformation; it represents a hypothesis about the underlying physics. The choice of which nodes to connect with edges defines the pathways through which information can flow during the GNN's message-passing phase. By using Delaunay triangulation, which connects adjacent nodes from the original CFD mesh, a strong and physically-grounded inductive bias is introduced: the assumption that physical quantities like pressure and velocity gradients are most influential between neighboring cells. This is a more natural and powerful assumption for a CFD problem than alternatives like a simple radius graph, which assumes isotropic interactions.

Mesh-to-Graph Conversion Pipeline

A reusable Python script (src/data_processing.py) will be developed to automate the conversion of the entire dataset.

- The script will iterate through the NumPy arrays loaded on Day 2. Each sample (a point cloud with features) will be converted into a torch_geometric.data.Data object, the standard data structure in PyG.[16]
- **Node Representation:** The nodes of the graph will directly correspond to the points in the CFD mesh.
- **Edge Connectivity:** The edge_index attribute of the Data object, which defines the graph's topology, will be constructed using Delaunay triangulation on the 2D spatial coordinates of the mesh points. The scipy.spatial.Delaunay library provides an efficient implementation for this. The resulting simplexes (triangles) define faces, from which an edge list can be easily extracted. PyG's FaceToEdge transform can also be utilized for this purpose.[17] Libraries like
  PyMesh also offer utilities for converting meshes to graphs.[19]

Feature Engineering

The Data object for each graph will be populated with carefully engineered features that encode both the geometry and the physics of the problem. This step is crucial for providing the GNN with the necessary information to make accurate predictions.[21]

- **Node Features (data.x):** The node feature matrix will be a concatenation of:
  - Normalized spatial coordinates (x, y).
  - Inlet velocity components (U_inf_x, U_inf_y), which define the boundary conditions for the simulation.
  - **Boundary Condition Flags:** A one-hot encoded vector will be created to explicitly identify the type of each node. This vector will have dimensions for categories like is_airfoil_surface, is_inlet, is_outlet, and is_far_field. This feature explicitly informs the GNN about the physical constraints at different parts of the domain, a critical piece of information that a simple coordinate-based model would have to learn implicitly.
- **Edge Features (data.edge_attr):** The edge feature matrix will encode the relative geometry between connected nodes:
  - The displacement vector (dx, dy) between the two nodes.
  - The scalar Euclidean distance between the nodes. These relative positional

encodings are essential for many GNN convolution operators, as they allow the message-passing function to be conditioned on the local geometry.

- **Global Features (data.u):** For tasks involving varying global parameters like the Reynolds number or angle of attack, these scalar values would be stored as graph-level features. They can be broadcast and concatenated to each node's feature vector before the first GNN layer.

Normalization

All input features and target variables will be normalized to ensure stable and efficient training.

- The statistical moments (mean and standard deviation) calculated during the EDA on Day 2 will be used to define the scaling transformations.
- Spatial coordinates will be scaled using Min-Max scaling to map them to a consistent range (e.g., $$).
- Flow fields and other features will be standardized using Z-score normalization (subtracting the mean and dividing by the standard deviation).
- Crucially, the scaler objects (e.g., from scikit-learn) will be fitted **only** on the training data and then applied consistently to the validation and test sets to prevent data leakage.

Dataset Creation

The final step is to process all samples from the training, validation, and test splits and save the resulting list of Data objects to disk using torch.save. This creates a final, analysis-ready graph dataset that can be loaded quickly during model training. For very large datasets, a custom class inheriting from PyG's InMemoryDataset would be an appropriate structure.[23]

Day 4: Implementation of a Baseline GNN Model

Objective: Implement and train a standard GNN model (GCN or GraphSAGE) to establish a performance baseline and verify the end-to-end data and training pipeline.

The development of a simple baseline model is not a throwaway step; it is a fundamental part

of a disciplined, scientific approach to model development. This baseline serves two critical purposes. First, it acts as an end-to-end test of the entire data pipeline, from loading and processing to batching and feeding into a model. If the baseline trains successfully, it provides confidence that the data is being handled correctly. Second, and more importantly, it provides a quantitative benchmark or a "scientific control." The performance of any subsequent, more complex model is only meaningful in comparison to this baseline. Without it, one cannot rigorously claim that advanced architectural choices or physics-informed techniques provide a tangible benefit.

Model Architecture Definition

A simple yet robust GNN architecture will be defined in a new script, src/models.py, using pre-built layers from PyTorch Geometric.[1] A standard encoder-processor-decoder structure will be used:

- **Encoder:** A multi-layer perceptron (MLP) that takes the initial node features and projects them into a higher-dimensional latent space (e.g., 128 dimensions). This allows the model to learn a rich initial representation for each node.
- **Processor:** A stack of 2-3 message-passing layers. **GraphSAGE** (SAGEConv) is an excellent choice for a baseline due to its robust performance and generalizability.[24] Each SAGEConv layer will be followed by a non-linear activation function (e.g., ReLU) and potentially a normalization layer (e.g., LayerNorm).
- **Decoder:** Another MLP that takes the final node embeddings from the processor and maps them back to the target dimension (4, for the two velocity components, pressure, and turbulent viscosity).

This architecture can be readily implemented by following the numerous examples and tutorials available in the PyTorch Geometric documentation.[16]

Training Script

A comprehensive training script, src/train.py, will be developed to manage the model training and validation process. The structure of this script can be informed by the example scripts provided in the official AirfRANS repository.[26]

- **Data Loading:** PyG's DataLoader will be used to efficiently create mini-batches of graphs. This loader automatically handles the collation of graphs of varying sizes into a single large graph with a batch vector to distinguish nodes from different samples.

- **Optimization:**
  - **Loss Function:** The primary training objective will be to minimize the Mean Squared Error (torch.nn.MSELoss) between the model's predictions and the ground truth flow fields.
  - **Optimizer:** The AdamW optimizer, an improved version of Adam with decoupled weight decay, is a standard and effective choice.
  - **Learning Rate Scheduler:** To improve convergence and avoid local minima, a learning rate scheduler such as ReduceLROnPlateau (which reduces the learning rate when the validation loss plateaus) or a cosine annealing schedule will be implemented.
- **Training and Validation Loops:** A standard PyTorch training loop will be written. This loop will iterate over epochs, and within each epoch, it will iterate over the batches provided by the DataLoader. For each batch, it will perform a forward pass, compute the loss, perform backpropagation to compute gradients, and update the model's weights using the optimizer. After each training epoch, a validation loop will be run on a held-out validation set (a subset of the full training data). This loop computes the loss on unseen data, providing a crucial signal for monitoring overfitting and for model selection (e.g., saving the model checkpoint with the lowest validation loss).
- **Logging:** To monitor the training process effectively, a logging utility such as TensorBoard or Weights & Biases will be integrated. This will allow for real-time visualization of key metrics like training loss, validation loss, and the learning rate throughout the training run.

Day 5: Baseline Model Evaluation and Initial Visualization

Objective: Quantitatively and qualitatively evaluate the trained baseline model to understand its performance and identify areas for improvement.

An evaluation based solely on a single, aggregated metric like the mean squared error across the entire test set can be dangerously misleading. A low average error might conceal a model that performs exceptionally well in the simple, far-field regions of the domain but fails catastrophically near the airfoil surface or in the complex wake region, where the physics is most critical. Therefore, a dual approach of quantitative and qualitative evaluation is essential. The qualitative visualizations serve as powerful diagnostic tools, transforming an abstract error score into a concrete, interpretable understanding of the model's physical deficiencies and successes. This level of detailed analysis is what distinguishes a basic project from an expert-level one.

Quantitative Evaluation

The best-performing baseline model checkpoint (selected based on the lowest validation loss) will be loaded for evaluation on the unseen test set.

- **De-normalization:** A critical first step is to de-normalize the model's predictions. The inverse of the scaling transformation applied on Day 3 will be used to convert the predicted fields back into their original physical units (e.g., m/s for velocity, m2/s2 for pressure/density). All subsequent error metrics will be calculated on these de-normalized values to ensure they are physically interpretable.
- **Field-wise Metrics:** The Mean Squared Error (MSE) and Mean Absolute Error (MAE) will be calculated for the predicted velocity and pressure fields. These metrics will be computed by averaging the error over all nodes in all graphs of the test set, providing a global measure of the model's accuracy on a per-node basis. The scikit-learn metrics module provides standard implementations for these calculations.
- **Global Metrics:** The AirfRANS paper correctly emphasizes that for engineering applications, the accuracy of derived physical quantities is often more important than per-node field accuracy.[10] Therefore, a function will be implemented to compute the integrated lift and drag coefficients from the predicted pressure and velocity fields on the airfoil surface nodes. The relative error in these predicted global coefficients compared to the ground truth will be a key performance indicator, as these values directly relate to the aerodynamic performance of the airfoil.[28]

Qualitative Evaluation (Visualization)

A new Jupyter notebook (notebooks/02_baseline_evaluation.ipynb) will be created to perform a deep visual analysis of the model's predictions.

- A few representative cases will be selected from the test set for detailed inspection, including a case with average performance, a case where the model performed poorly, and a case where it performed exceptionally well.
- For each selected case, a series of side-by-side comparison plots will be generated using matplotlib and PyVista:
    - **Pressure and Velocity Contour Plots:** PyVista's plotting capabilities will be used to generate contour plots of the ground truth and predicted flow fields on the 2D mesh.[29] These visualizations will provide an immediate qualitative assessment of whether the model is capturing the overall flow structure, such as the high-pressure stagnation point at the leading edge and the low-pressure region on the upper surface.

- **Spatial Error Maps:** The predicted field will be subtracted from the ground truth field to generate a spatial error map. This error field will then be plotted on the mesh. This is a powerful diagnostic tool that will highlight specific regions of the domain where the model struggles, such as in the boundary layer, at the trailing edge, or in the turbulent wake.
- **Pressure Coefficient (Cp) Plots:** A standard and critical visualization in aerodynamics is the plot of the pressure coefficient (Cp) along the airfoil's surface versus the normalized chordwise position (x/c).[32] A matplotlib plot will be created showing the ground truth Cp distribution as a solid line and the model's predicted Cp distribution as a dashed line or points. This plot provides a precise, quantitative measure of the model's ability to predict the surface pressures that are directly responsible for generating aerodynamic lift and drag.

Week 2: Advanced Architectures, In-Depth Analysis, and Final Deliverables

The second week transitions from foundational work to the implementation and analysis of state-of-the-art techniques. The goal is to systematically build upon the baseline model, incorporating advanced architectures, physics-informed constraints, and uncertainty quantification. The week culminates in a comprehensive comparative analysis and the packaging of the entire project into a polished, reproducible portfolio piece.

Day 6-7: Developing an Advanced Hierarchical GNN

Objective: Implement a more powerful GNN architecture inspired by PointNet++ and MeshGraphNets to better capture the multi-scale nature of fluid dynamics.

Fluid dynamics is an inherently multi-scale phenomenon. Small-scale turbulent eddies near the airfoil surface can coalesce and influence the large-scale vortex shedding pattern in the wake. A standard, "flat" GNN architecture struggles to capture these long-range dependencies efficiently. Information propagates only one "hop" per message-passing layer, meaning a very deep network would be required for a node on one side of the domain to be influenced by a node on the other. Such deep GNNs are notoriously difficult to train due to issues like over-smoothing.

A hierarchical architecture, inspired by models like MeshGraphNets and PointNet++, provides

a direct and elegant solution to this problem.[34] By creating progressively coarser representations of the graph, the model establishes "shortcuts" for information to travel across the domain in fewer computational steps. This architectural choice explicitly encodes a physical prior about multi-scale interactions, making it far more suitable and sample-efficient for this class of problems than a simple GCN or GraphSAGE.

Conceptual Foundation

The design of the advanced model will draw from two seminal works:

- **MeshGraphNets:** This framework proposes an **Encode-Process-Decode** architecture. An encoder maps the input graph to a latent space, a "processor" consisting of a deep stack of GNN layers performs message passing in this latent space, and a decoder maps the latent representations back to the physical domain. The deep processor allows for learning complex and long-range interactions.[22]
- **PointNet++:** This model introduces the concept of **hierarchical feature learning** for point clouds. It works by recursively applying a base network (PointNet) on nested, progressively larger partitions of the input point set. This is achieved through a sequence of "set abstraction" layers, each involving sampling, grouping, and pooling operations to create a coarser-resolution representation with richer contextual features.[36]

Implementation Strategy

A new model will be implemented in src/models.py, combining these ideas into a U-Net-like hierarchical GNN using PyTorch Geometric's pooling and unpooling modules.

- **Encoder (Downsampling Path):**
  1. An initial MLP will embed the input node features into a high-dimensional latent space.
  2. This will be followed by a sequence of GNN blocks (e.g., two SAGEConv layers with ReLU and LayerNorm).
  3. After each GNN block, a graph pooling layer (e.g., torch_geometric.nn.pool.voxel_grid_pool) will be used to downsample the graph. This layer groups nodes based on their spatial location and aggregates their features, creating a new, coarser graph with fewer nodes. This process will be repeated 2-3 times, creating a hierarchy of graph resolutions.
- **Processor (Bottleneck):**
  1. At the coarsest level of the hierarchy, a deep stack of GNN blocks (e.g., 10-15 layers,

as in MeshGraphNets) will be applied. Message passing at this low resolution is computationally efficient and allows information to propagate across the entire domain in just a few steps, effectively learning global dynamics.

- **Decoder (Upsampling Path):**
    1. The model will then progressively upsample the graph back to its original resolution using graph unpooling operations.
    2. Crucially, **skip connections** will be used to concatenate the feature maps from the upsampling path with the corresponding feature maps from the encoder path at the same resolution. This U-Net-like structure is vital for preserving high-frequency, fine-grained details that might be lost during the pooling operations.
    3. The final stage will be an MLP that decodes the node embeddings at the original resolution back into the predicted flow fields.

Several open-source implementations of MeshGraphNets can serve as valuable references for this architecture.[39]

Training and Initial Evaluation

The training script from Day 4 will be adapted to handle this more complex model. It is anticipated that this larger model will require longer training times and more careful tuning of hyperparameters such as learning rate, the number of layers at each level, and the hidden feature dimensions. An initial evaluation will be performed using the same quantitative and qualitative metrics from Day 5 to confirm that the hierarchical model is learning effectively and provides a performance improvement over the baseline.

Day 8: Integrating Physics-Informed Constraints

Objective: Enhance the model's physical plausibility and data efficiency by incorporating the governing equations (PDEs) directly into the loss function.

In a low-data regime like the 'scarce' task, a purely data-driven model is highly susceptible to overfitting. It may learn non-physical solutions that happen to fit the sparse training examples but fail to generalize. A physics-informed loss function acts as a powerful, data-free regularizer. It provides a form of supervision at every point in the domain—even where no ground truth data exists—by penalizing the model for violating known physical laws. This constraint forces the model's learned function to lie on or near the manifold of physically plausible solutions, leading to significantly better generalization and physical consistency, a

key challenge in scientific machine learning.[42]

Understanding Physics-Informed Loss

The core principle of a Physics-Informed Neural Network (PINN) is to augment the standard data-driven loss function with a term that measures the model's adherence to governing physical laws, typically expressed as partial differential equations (PDEs).[44]

The total loss function becomes a weighted sum:

$$L_{total} = L_{data} + \lambda \cdot L_{physics}$$

where:

- L_{data} is the supervised Mean Squared Error on the labeled training data.
- $L_{physics}$ is the mean squared residual of the governing PDE(s).
- λ is a tunable hyperparameter that balances the contribution of the data and physics terms.

Implementing the Physics Loss Term

For the AirfRANS dataset, which models incompressible flow, a fundamental physical law is the conservation of mass. For an incompressible fluid, this law simplifies to the divergence-free condition on the velocity field $u=(u,v)$:

$$\nabla \cdot u = \partial_x u + \partial_y v = 0$$

This constraint will be implemented as the physics loss term.

- **Automatic Differentiation:** The key enabling technology is automatic differentiation. PyTorch's torch.autograd.grad function will be used to compute the necessary spatial derivatives of the GNN's output (the velocity components u and v) with respect to its spatial coordinate inputs (x and y). This requires the input coordinates tensor to have requires_grad=True and the create_graph=True flag to be set in the grad call to allow for backpropagation through the derivative calculation.
- **Loss Calculation:** The implementation will be integrated into the training loop:
  1. During the forward pass, the model predicts the velocity field upred.
  2. Using automatic differentiation, the divergence of the predicted field, $\nabla \cdot u_{pred}$, is computed for each node in the graph.

3. The physics loss is then calculated as the mean squared error between the computed divergence and zero: $L_{physics} = MSE(\nabla \cdot u_{pred}, 0)$.
4. This physics loss, weighted by λ, is added to the data loss before the backward pass. The hyperparameter λ will be tuned to achieve the best balance between fitting the data and satisfying the physical constraint.

Collocation Points

The physics loss will be evaluated at the locations of the mesh nodes in the training data. These nodes effectively serve as "collocation points" where the PDE residual is minimized. This approach leverages the existing data structure without the need to sample additional points, providing a dense source of physics-based supervision across the domain.

Day 9: Implementing Multi-Fidelity Learning Strategies

Objective: Improve data efficiency and model performance by leveraging low-fidelity data, simulating a common scenario in engineering where cheap, approximate simulations are abundant.

The concept behind multi-fidelity learning is to train a model on a large corpus of inexpensive, low-fidelity (LF) data to learn general principles, and then refine its knowledge using a small set of expensive, high-fidelity (HF) data.[46] This process is analogous to how a human expert first learns the fundamentals of a field from textbooks before mastering specifics through hands-on experience. The pre-training on LF data allows the model to learn a powerful "physics prior"—a fundamental representation of the system's dynamics. When this pre-trained model is subsequently fine-tuned on the scarce HF data, it is not starting from a random, uninformed state. Instead, it starts from a state that already "understands" the general behavior of the system, allowing it to converge to a high-performance solution with far fewer HF samples.

Conceptual Basis and Implementation

Since the AirfRANS dataset only provides a single fidelity level, this scenario will be simulated using a **transfer learning** methodology, a common and effective approach in multi-fidelity

settings.[48]

- **Source Task (Simulated Low-Fidelity):** The "low-fidelity" data will be the 800 training samples from the AirfRANS 'full' data regime. The advanced hierarchical GNN from Day 7 will be trained to convergence on this large dataset. This pre-training phase allows the model to learn the general features of fluid flow over a wide variety of airfoils and conditions.
- **Target Task (High-Fidelity):** The "high-fidelity" data is the 200-sample 'scarce' training set. The fine-tuning process will proceed as follows:
  1. The weights of the pre-trained model will be loaded.
  2. The initial layers of the model (e.g., the encoder part of the U-Net) may be frozen. This preserves the general-purpose feature extractors learned during pre-training.
  3. The model will then be trained for additional epochs on the 'scarce' dataset, using a smaller learning rate than in the pre-training phase. This allows the model to gently adapt its learned representations to the specific characteristics of the high-fidelity data distribution without catastrophically forgetting the knowledge gained during pre-training.

Evaluation

The performance of the fine-tuned multi-fidelity model will be rigorously compared against the model from Day 7, which was trained from scratch using only the 'scarce' data. The expectation is that the transfer-learned model will achieve significantly higher accuracy on the test set, thereby demonstrating the substantial value of leveraging a larger, albeit simulated, low-fidelity dataset to improve performance in a data-limited scenario.

Day 10: Uncertainty Quantification (UQ)

Objective: Augment the model to not only make predictions but also provide a measure of its confidence in those predictions, which is critical for real-world deployment.

A standard deterministic model provides a single point estimate as its prediction. It has no mechanism to communicate its own confidence; if its prediction is wrong, it is often confidently wrong. Uncertainty quantification (UQ) addresses this critical limitation by enabling the model to produce a predictive distribution rather than a single value. This distribution reveals the model's "known unknowns"—regions of the input space where it is uncertain about its prediction. This information is invaluable for engineering applications. A

prediction with low uncertainty can be trusted for decision-making, while a prediction with high uncertainty can be automatically flagged for further investigation, perhaps by running a full-fidelity CFD solver. This capability elevates the GNN from a simple surrogate to an active component in an intelligent, adaptive simulation workflow.

Methodology: Monte Carlo (MC) Dropout

MC Dropout is a theoretically grounded yet practical Bayesian approximation technique for estimating model uncertainty.[49] It re-purposes the standard dropout layer, typically used as a regularizer during training, as a tool for Bayesian inference.

- The core insight is that by keeping dropout layers **active during inference** and performing multiple forward passes, the model effectively samples from an approximate posterior distribution of the model weights.
- This process generates an ensemble of predictions for a single input, and the statistics of this ensemble (e.g., mean and variance) provide an estimate of the model's predictive mean and uncertainty.

Implementation

The UQ capability will be added to the advanced GNN model.

- It will be ensured that torch.nn.Dropout layers are present within the MLP components of the hierarchical GNN (e.g., in the encoder and decoder).
- A new evaluation script (src/evaluate_uq.py) will be created to perform the MC Dropout procedure.
- During inference, the model will first be set to evaluation mode (model.eval()) to disable non-dropout-related training behaviors like in BatchNorm. Then, the dropout layers will be manually re-activated by iterating through the model's modules and calling .train() on them.
- For each sample in the test set, the model will be run in a loop for T stochastic forward passes (e.g., T=50). In each pass, a different dropout mask is randomly applied, resulting in a slightly different prediction. The T predictions are stored.
- **Uncertainty Calculation:** From the stored set of T predictions, two key quantities are calculated for each node:
    - **Predictive Mean:** The average of the T predictions. This serves as the final, uncertainty-aware prediction.
    - **Predictive Variance:** The variance of the T predictions. This scalar value represents

the model's uncertainty at that specific node.

Initial Analysis

The output of the UQ pipeline will be a mean prediction field and a variance field for each test sample.

- Using PyVista, the variance field will be visualized as a contour plot on the mesh. This "uncertainty map" will be analyzed to identify regions where the model is most uncertain. It is hypothesized that uncertainty will be highest in regions of complex or highly variable flow, such as in the turbulent wake, near flow separation points, or for airfoil geometries that are outliers in the training distribution.

Day 11-12: Comprehensive Evaluation, Visualization, and Ablation Studies

Objective: Systematically evaluate all developed models, generate a suite of high-quality visualizations for the final report, and perform ablation studies to understand the contribution of each architectural component.

Simply presenting the final, most complex model is insufficient for a top-tier portfolio project. A rigorous evaluation must demonstrate *why* the final model is superior and quantify the contribution of each of its components. Ablation studies provide the narrative for the model's design. They form a structured, evidence-based argument that justifies each architectural and methodological choice. For example, a statement like, "Incorporating the hierarchical architecture reduced pressure field MSE by 15% over the baseline, demonstrating the importance of multi-scale feature learning," is a powerful claim backed by empirical data. This analytical depth showcases a sophisticated, scientific understanding of model development.

Final Model Training

A final, "all-in" model will be trained, integrating the most successful techniques developed throughout the week. This model will feature the hierarchical architecture, be trained with the physics-informed loss function, and will be initialized using the multi-fidelity transfer learning

approach.

Systematic Quantitative Evaluation

All models developed during the project will be evaluated on the held-out test set to facilitate a direct comparison. The models include:

1. Baseline GNN (GraphSAGE)
2. Hierarchical GNN
3. Hierarchical GNN + Physics-Informed Loss
4. Hierarchical GNN + Multi-Fidelity (Transfer Learning)
5. Final Model (Hierarchical + Physics + Multi-Fidelity)

For each model, the full suite of quantitative metrics will be computed: field-wise MSE and MAE for pressure and velocity, the percentage error in the predicted lift and drag coefficients, and the R-squared value for the predicted vs. true lift and drag coefficients across the test set.[28] The results will be compiled into a clear, comprehensive table.

Advanced Visualization

A final analysis notebook (notebooks/03_final_analysis.ipynb) will be created to generate a suite of high-quality visualizations.

- **Pressure Coefficient (Cp) Plots with Uncertainty:** For the UQ-enabled model, the Cp vs. x/c plot will be enhanced. The predictive mean from the MC Dropout procedure will be plotted as a solid line, and the uncertainty (e.g., mean ± one standard deviation) will be visualized as a shaded region using matplotlib.pyplot.fill_between.[52] This is a standard and highly effective method for communicating predictive uncertainty in scientific plots.[53]
- **Streamline Comparisons:** PyVista's streamlines filter will be used to visualize and compare the flow paths from the ground truth and predicted velocity fields.[54] This qualitative comparison is excellent for assessing whether the model correctly captures large-scale flow features like recirculation bubbles in the wake or the curvature of flow around the airfoil.
- **Predicted vs. True Scatter Plots:** For the global quantities of lift and drag, scatter plots of the predicted values versus the true values for every sample in the test set will be generated. A perfect model's predictions would fall on the y=x line. The R-squared value will be calculated and displayed on these plots to quantify the correlation.

Ablation Studies

The comparative results table forms the basis of the ablation study.[56] A written analysis will be drafted to interpret these results, systematically quantifying the impact of each component:

- **Effect of Hierarchical Architecture:** Performance of Model 2 vs. Model 1.
- **Effect of Physics-Informed Loss:** Performance of Model 3 vs. Model 2.
- **Effect of Multi-Fidelity Learning:** Performance of Model 4 vs. Model 2.
- **Synergistic Effects:** Performance of the final Model 5 vs. all others.

The improvements will be quantified in terms of the percentage reduction in key error metrics, providing concrete evidence for each design decision.

Key Table: Comparative Model Performance on AirfRANS 'Scarce' Test Set

The following table structure will be used to summarize the central quantitative findings of the project. It provides a clear, at-a-glance comparison of all developed models across a range of relevant metrics, serving as the quantitative backbone for the ablation study analysis.

Model Configuration	Pressure MSE (field)	Velocity MAE (field)	Lift Coeff. Error (%)	Drag Coeff. Error (%)	R^2 (Lift)
1. Baseline (GraphSAGE)					
2. Hierarchical GNN					
3. Hierarchical + Physics Loss					

4. Hierarchical + Multi-Fidelity					
5. Final Model (All-In)					

Day 13-14: Code Refactoring, Documentation, and Project Packaging

Objective: Transform the collection of scripts and notebooks into a polished, reproducible, and well-documented portfolio piece.

A project with outstanding results but with code that is undocumented and cannot be run by others is of limited value. In both industrial and academic settings, reproducibility is the standard of excellence. This final phase of the project is dedicated to transforming the research code into a high-quality, reusable software artifact. A well-documented and easy-to-run repository demonstrates a professional skill set that is as crucial as the ability to design a neural network. It signals a developer who is a careful, considerate, and effective collaborator, capable of producing not just models, but maintainable and usable tools.

Code Cleaning and Refactoring

All source code in the src/ directory will be reviewed and refined.

- **Code Quality:** The code will be checked for clarity, and comments will be added where necessary to explain complex logic. Adherence to the PEP 8 style guide will be enforced.
- **Modularity:** Any large, monolithic scripts will be refactored into smaller, reusable functions with clear, single responsibilities.
- **Documentation:** Docstrings will be added to all functions and classes, explaining their purpose, parameters, and return values in a standard format.
- **Dependency Management:** A requirements.txt file will be generated using pip freeze > requirements.txt to lock the exact versions of all project dependencies, ensuring that the environment can be perfectly replicated.

Comprehensive Documentation (README.md)

The README.md file is the front page of the project repository and must be comprehensive and clear. It will be structured to guide a visitor from a high-level overview to detailed usage instructions. High-quality machine learning repositories on GitHub will be used as templates for structure and content.[8] The README will include:

- **Project Title and Summary:** A concise overview of the project's goal and key achievements.
- **Motivation:** A brief explanation of why GNN-based surrogate modeling for CFD is an important and challenging problem.
- **Dataset:** A description of the AirfRANS dataset with a link to the official source.
- **Methodology:** A high-level explanation of the different models developed, from the baseline to the final advanced architecture.
- **Key Results:** The most compelling visualizations (e.g., a side-by-side prediction vs. ground truth, the Cp plot with uncertainty) and the final comparative results table will be embedded directly in the README.
- **Repository Structure:** A brief explanation of the purpose of each directory (src, notebooks, etc.).
- **Setup and Usage:** Clear, numbered, step-by-step instructions on how to clone the repository, create the environment from requirements.txt, and run the key scripts (e.g., python src/train.py --model hierarchical_physics --config params.yaml).

Finalizing Notebooks

The Jupyter notebooks will be cleaned and polished to serve as narrative documents that walk a reader through the project's analysis.

- Extraneous code cells and intermediate outputs will be removed.
- Markdown cells with clear headings and explanatory text will be added to describe each step of the analysis, the rationale behind it, and the interpretation of the results.
- All plots will be ensured to have clear titles, axis labels, and legends.

Repository Finalization

Final touches will be put on the GitHub repository to prepare it for public viewing.

- A .gitignore file will be configured to exclude unnecessary files and directories from version control, such as __pycache__, large data files, and environment folders.
- The entire cleaned and documented project will be pushed to a public GitHub repository with a professional and descriptive name (e.g., gnn-cfd-surrogate-airfrans).

Works cited

1. pyg-team/pytorch_geometric: Graph Neural Network Library for PyTorch - GitHub, accessed on August 19, 2025, https://github.com/pyg-team/pytorch_geometric
2. Introduction - AirfRANS documentation, accessed on August 19, 2025, https://airfrans.readthedocs.io/en/latest/notes/introduction.html
3. Introduction to PyVista in Python - GeeksforGeeks, accessed on August 19, 2025, https://www.geeksforgeeks.org/python/introduction-to-pyvista-in-python/
4. PyVista — PyVista 0.46.1 documentation, accessed on August 19, 2025, https://docs.pyvista.org/
5. Matplotlib — Visualization with Python, accessed on August 19, 2025, https://matplotlib.org/
6. airfrans.dataset, accessed on August 19, 2025, https://airfrans.readthedocs.io/en/latest/modules/dataset.html
7. AirfRANS documentation — AirfRANS documentation, accessed on August 19, 2025, https://airfrans.readthedocs.io/
8. ml-project · GitHub Topics, accessed on August 19, 2025, https://github.com/topics/ml-project
9. machine-learning-project · GitHub Topics, accessed on August 19, 2025, https://github.com/topics/machine-learning-project
10. AirfRANS: High Fidelity Computational Fluid Dynamics Dataset for Approximating Reynolds-Averaged Navier-Stokes Solutions - arXiv, accessed on August 19, 2025, https://arxiv.org/abs/2212.07564
11. Dataset - AirfRANS documentation - Read the Docs, accessed on August 19, 2025, https://airfrans.readthedocs.io/en/latest/notes/dataset.html
12. AirfRANS: High Fidelity Computational Fluid Dynamics Dataset for Approximating Reynolds-Averaged Navier–Stokes Solutions | OpenReview, accessed on August 19, 2025, https://openreview.net/forum?id=Zp8YmiQ_bDC&trk=public_post_comment-text
13. PyVista SciPy 2022-2025 Tutorial - GitHub, accessed on August 19, 2025, https://github.com/pyvista/pyvista-tutorial
14. PyVista Tutorial — PyVista Tutorial, accessed on August 19, 2025, https://tutorial.pyvista.org/
15. Getting Started — PyVista 0.45.2 documentation, accessed on August 19, 2025, https://docs.pyvista.org/getting-started/index.html
16. Introduction by Example — pytorch_geometric documentation - PyTorch Geometric, accessed on August 19, 2025, https://pytorch-geometric.readthedocs.io/en/2.6.1/get_started/introduction.html

17. pytorch-geometric.readthedocs.io, accessed on August 19, 2025, https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.GeometricShapes.html#:~:text=To%20convert%20the%20mesh%20to,point%20cloud%2C%20use%20the%20torch_geometric.
18. torch_geometric.datasets.GeometricShapes - PyTorch Geometric - Read the Docs, accessed on August 19, 2025, https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.GeometricShapes.html
19. PyMesh — Geometry Processing Library for Python — PyMesh 0.2.1 documentation, accessed on August 19, 2025, https://pymesh.readthedocs.io/
20. PyMesh/python/pymesh/meshutils/mesh_to_graph.py at main - GitHub, accessed on August 19, 2025, https://github.com/PyMesh/PyMesh/blob/main/python/pymesh/meshutils/mesh_to_graph.py
21. MeshGraphNet for Reduced-Order cardiovascular simulations - NVIDIA Docs, accessed on August 19, 2025, https://docs.nvidia.com/deeplearning/physicsnemo/physicsnemo-core/examples/healthcare/bloodflow_1d_mgn/readme.html
22. MeshGraphNet for transient vortex shedding - NVIDIA Docs, accessed on August 19, 2025, https://docs.nvidia.com/deeplearning/physicsnemo/physicsnemo-core/examples/cfd/vortex_shedding_mgn/readme.html
23. torch_geometric.datasets.geometry — pytorch_geometric documentation - PyTorch Geometric, accessed on August 19, 2025, https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/datasets/geometry.html
24. torch_geometric.nn.models.GraphSAGE — pytorch_geometric documentation - PyTorch Geometric, accessed on August 19, 2025, https://pytorch-geometric.readthedocs.io/en/2.5.0/generated/torch_geometric.nn.models.GraphSAGE.html
25. Graph Neural Networks using Pytorch | by Andrea Rosales | Medium, accessed on August 19, 2025, https://medium.com/@andrea.rosales08/introduction-to-graph-neural-networks-78cbb6f64011
26. In this repository, you will find the different python scripts to train the available models on the AirfRANS dataset proposed at the NeurIPS 2022 Datasets and Benchmarks Track conference. - GitHub, accessed on August 19, 2025, https://github.com/Extrality/AirfRANS
27. AirfRANS: High Fidelity Computational Fluid Dynamics Dataset for Approximating Reynolds-Averaged Navier–Stokes Solutions, accessed on August 19, 2025, https://neurips.cc/media/neurips-2022/Slides/55720.pdf
28. An Investigation of Surrogate Models for Efficient Performance-Based Decoding of 3D Point Clouds, accessed on August 19, 2025, https://asmedigitalcollection.asme.org/mechanicaldesign/article/141/12/121401/975226/An-Investigation-of-Surrogate-Models-for-Efficient

29. Plot CFD Data — PyVista 0.46.1 documentation, accessed on August 19, 2025, https://docs.pyvista.org/examples/99-advanced/openfoam_tubes.html
30. Contouring - PyVista Tutorial, accessed on August 19, 2025, https://tutorial.pyvista.org/tutorial/04_filters/solutions/d_contouring.html
31. Creating a Contour Map Using Python PyVista - GeeksforGeeks, accessed on August 19, 2025, https://www.geeksforgeeks.org/python/creating-a-contour-map-using-python-pyvista/
32. airfoil profile geometry plotting - python - Stack Overflow, accessed on August 19, 2025, https://stackoverflow.com/questions/25958883/airfoil-profile-geometry-plotting
33. How to plot the pressure distribution over an airfoil? - Aviation Stack Exchange, accessed on August 19, 2025, https://aviation.stackexchange.com/questions/5230/how-to-plot-the-pressure-distribution-over-an-airfoil
34. [PDF] MultiScale MeshGraphNets - Semantic Scholar, accessed on August 19, 2025, https://www.semanticscholar.org/paper/MultiScale-MeshGraphNets-Fortunato-Pfaff/b6841ed68e0e6d9cabeb28e9dfca22ffa853f30f
35. Learning mesh-based simulations - Google Sites, accessed on August 19, 2025, https://sites.google.com/view/meshgraphnets
36. charlesq34/pointnet2: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space - GitHub, accessed on August 19, 2025, https://github.com/charlesq34/pointnet2
37. Learning Mesh-Based Simulation with Graph Networks - OpenReview, accessed on August 19, 2025, https://openreview.net/forum?id=roNqYL0_XP
38. Graph Residual Point Cloud Network for 3D Object Classification and Segmentation - arXiv, accessed on August 19, 2025, https://arxiv.org/html/2412.03052v1
39. CCSI-Toolset/MGN: MeshGraphNets (MGN) - GitHub, accessed on August 19, 2025, https://github.com/CCSI-Toolset/MGN
40. echowve/meshGraphNets_pytorch: PyTorch implementations of Learning Mesh-based Simulation With Graph Networks - GitHub, accessed on August 19, 2025, https://github.com/echowve/meshGraphNets_pytorch
41. MeshGraphNets Implementation in PyG for Stanford CS224W (Machine Learning with Graphs) 2021 - GitHub, accessed on August 19, 2025, https://github.com/rclupoiu/meshgraphnets
42. Physics-Informed Graph Neural Networks for Attack Path Prediction - MDPI, accessed on August 19, 2025, https://www.mdpi.com/2624-800X/5/2/15
43. Physics-Informed Graph Neural Operator for Mean Field Games on Graph: A Scalable Learning Approach - MDPI, accessed on August 19, 2025, https://www.mdpi.com/2073-4336/15/2/12
44. Physics-Informed Neural Networks with MATLAB - Conor Daly | Deep Dive Session 5, accessed on August 19, 2025, https://www.youtube.com/watch?v=RTR_RklvAUQ

45. What Are Physics-Informed Neural Networks (PINNs)? - MATLAB & Simulink - MathWorks, accessed on August 19, 2025, https://www.mathworks.com/discovery/physics-informed-neural-networks.html
46. A Multi-Fidelity Graph U-Net Model for Accelerated Physics Simulations - arXiv, accessed on August 19, 2025, https://arxiv.org/html/2412.15372v1
47. MehdiTaghizadehUVa/MFGNN_Power: This repository contains the implementation and data for our study on multi-fidelity graph neural networks (MF-GNN) for efficient power flow analysis under demand and renewable generation uncertainty. - GitHub, accessed on August 19, 2025, https://github.com/MehdiTaghizadehUVa/MFGNN_Power
48. davidbuterez/multi-fidelity-gnns-for-drug-discovery-and-quantum-mechanics - GitHub, accessed on August 19, 2025, https://github.com/davidbuterez/multi-fidelity-gnns-for-drug-discovery-and-quantum-mechanics
49. What is Monte Carlo (MC) dropout? - GeeksforGeeks, accessed on August 19, 2025, https://www.geeksforgeeks.org/deep-learning/what-is-monte-carlo-mc-dropout/
50. Uncertainty in Graph Neural Networks: A Survey - arXiv, accessed on August 19, 2025, https://arxiv.org/html/2403.07185v1
51. Evaluation of supervised machine learning regression models for CFD-based surrogate modelling in indoor airflow field reconstruction | Request PDF - ResearchGate, accessed on August 19, 2025, https://www.researchgate.net/publication/385219359_Evaluation_of_supervised_machine_learning_regression_models_for_CFD-based_surrogate_modelling_in_indoor_airflow_field_reconstruction
52. Visualizing ranges and uncertainty - Practical Data Science with Python, accessed on August 19, 2025, https://www.practicaldatascience.org/notebooks/class_5/week_1/1.2.8_errorbars.html
53. 16 Visualizing uncertainty - Fundamentals of Data Visualization, accessed on August 19, 2025, https://clauswilke.com/dataviz/visualizing-uncertainty.html
54. 2D Streamlines — PyVista 0.45.2 documentation, accessed on August 19, 2025, https://docs.pyvista.org/examples/01-filter/streamlines_2d
55. Streamlines — PyVista 0.45.2 documentation, accessed on August 19, 2025, https://docs.pyvista.org/examples/01-filter/streamlines.html
56. Ablation (artificial intelligence) - Wikipedia, accessed on August 19, 2025, https://en.wikipedia.org/wiki/Ablation_(artificial_intelligence)
57. machine-learning-projects · GitHub Topics, accessed on August 19, 2025, https://github.com/topics/machine-learning-projects?l=python
58. machine-learning-projects · GitHub Topics, accessed on August 19, 2025, https://github.com/topics/machine-learning-projects?o=desc&s=updated
59. machinelearningprojects · GitHub Topics, accessed on August 19, 2025, https://github.com/topics/machinelearningprojects