**1. Introduction**

Finite Element Analysis (FEA) is an indispensable tool in modern engineering, providing high-fidelity simulations for the design and analysis of complex mechanical systems. Despite its power, the computational cost of FEA remains a significant bottleneck, particularly in applications requiring numerous sequential simulations, such as design optimization, uncertainty quantification, and the development of real-time digital twins. This computational demand often restricts engineers to simplified models or limits the exploration of the design space, creating a need for computationally efficient, high-accuracy alternatives.

To mitigate these costs, the engineering community has historically relied on techniques such as Reduced-Order Models (ROMs), most notably those based on Proper Orthogonal Decomposition (POD), as well as response surface methodologies and Kriging. These methods seek to project the high-dimensional solution space onto a low-dimensional manifold, allowing for rapid approximation of new solutions. However, despite their success in certain domains, traditional ROMs have notable limitations. Many are based on a linear subspace assumption, which limits their effectiveness for highly non-linear phenomena. Furthermore, methods like POD-Galerkin are often "intrusive," requiring modification of the underlying solver code, which is impractical for commercial FEA software. These methods can also struggle to generalize to designs that lie far from the initial training data and can be challenged by high-dimensional parameter spaces.

These limitations have motivated the search for more flexible, non-linear, and non-intrusive methods, creating a fertile ground for the application of deep learning. Deep learning-based surrogate models have emerged as a promising solution, learning to approximate the input-output mapping of expensive simulators with inference times that are orders of magnitude faster. For problems in structural mechanics, which are naturally defined on unstructured meshes, Graph Neural Networks (GNNs) offer a particularly powerful inductive bias. By representing the FEA mesh as a graph, GNNs can leverage the inherent topology of the structure, allowing for a more natural and efficient learning of physical phenomena like stress and strain propagation compared to grid-based methods like Convolutional Neural Networks (CNNs).

However, the development of robust surrogate models is not without its challenges. Purely data-driven models may struggle to generalize to out-of-distribution scenarios and can produce physically implausible results, as they lack any intrinsic understanding of the system's governing laws. Physics-Informed Neural Networks (PINNs) offer a compelling solution by embedding the governing partial differential equations (PDEs) directly into the neural network's loss function. This approach acts as a strong regularizer, guiding the model toward solutions that are not only accurate with respect to the training data but are also consistent with fundamental physical principles.

This paper presents a comprehensive investigation into the development of GNN-based surrogate models for accelerating FEA of I-beam structures, with a specific focus on the enhancement of these models with a PINN framework. Our key contributions are:

1. **A rigorous architectural comparison:** We provide a systematic evaluation of multiple GNN architectures (GCN, GAT, MPNN, Graph Transformer) against a 3D U-Net baseline on both specialized (unimodal) and generalized (multimodal) load scenarios.
2. **Successful PINN implementation for structural mechanics:** We demonstrate the successful integration of the Navier-Cauchy equations for linear elasticity into the GNN training process to measurably improve model generalization.
3. **A robust PINN training strategy:** We introduce and validate a curriculum learning approach with loss weight annealing, proving it is a critical strategy for stabilizing the training of PINN models and ensuring their convergence.
4. **A practical performance-efficiency analysis:** We identify the Graph Transformer as the most accurate architecture, but highlight the PINN-enhanced MPNN (MPNN-PINN) as the superior solution for practical deployment, offering an optimal balance of predictive accuracy, model size, and inference speed.

**3. Methodology**

3.1. Problem Formulation and Data Generation

The foundation of any surrogate modeling task is a comprehensive and well-structured dataset. We developed a robust data generation pipeline to simulate a parametric I-beam, a ubiquitous structural element in civil and mechanical engineering. The I-beam was chosen as it represents a canonical yet non-trivial problem: its geometry is simple enough to be parameterized efficiently, but its anisotropic bending stiffness presents a meaningful challenge for learning models. All ground truth data was generated using a high-fidelity Finite Element Analysis (FEA) pipeline built with the open-source gmsh mesh generator and the dolfinx finite element solver.

Parametric Design Space and Mesh Consistency:  
To ensure the surrogate models can generalize, we defined a multi-dimensional parameter space encompassing geometric properties, material characteristics, and loading conditions (detailed in Table 1). These parameters were sampled using Latin Hypercube Sampling (LHS) to ensure an efficient and uniform exploration of the space.

A critical decision in our data generation strategy was to maintain a consistent mesh topology and element size across all 2500 simulations. While the node coordinates were updated to reflect changes in the beam's geometric parameters (e.g., flange\_width, beam\_depth), the number of nodes and their connectivity (i.e., the graph structure) remained constant. This approach was chosen to isolate the learning task to the underlying physics, forcing the models to learn the effects of changing geometric, material, and load parameters without the confounding variable of a changing mesh discretization. This ensures a consistent basis for comparison across all models, particularly the GNNs which operate directly on this graph structure.

*Table 1: Parametric space for the I-beam FEA simulations.*  
| Parameter | Description | Type | Range / Values |  
| :--- | :--- | :--- | :--- |  
| beam\_length | The length of the beam along the Z-axis. | Continuous | [280.0, 320.0] mm |  
| flange\_width | The total width of the top and bottom flanges. | Continuous | [90.0, 110.0] mm |  
| flange\_thickness | The thickness of the flanges. | Continuous | [13.0, 17.0] mm |  
| web\_thickness | The thickness of the central vertical web. | Continuous | [8.0, 12.0] mm |  
| beam\_depth | The total height of the I-beam. | Continuous | [140.0, 160.0] mm |  
| fillet\_radius | The radius of the fillets at the web-flange junctions. | Continuous | [10.0, 14.0] mm |  
| youngs\_modulus | The Young's Modulus of the material (variations of steel). | Continuous | GPa |  
| poissons\_ratio| The Poisson's Ratio of the material. | Continuous | [0.28, 0.32] |  
| force\_magnitude | The total magnitude of the force applied to the free end. | Continuous | kN |  
| load\_type | The nature of the applied load. | Categorical | {bending\_y, bending\_x, torsion} |  
| load\_distribution | The spatial distribution of the applied load. | Categorical | {uniform, linear\_y} |

**Boundary and Loading Conditions:**  
The simulations model a cantilever beam configuration, as depicted in Figure 1. One end of the beam (at Z=0) is fully fixed, representing a clamped boundary condition. A distributed traction force is applied to the surface at the free end (at Z=L). The nature of this force is determined by the load\_type parameter: bending\_y (vertical), bending\_x (horizontal), or torsion (twisting moment).

**Figure 1: FEA Problem Formulation and Sample Data.** (a) The I-beam geometry with boundary conditions, showing the fixed surface (red) and the load application surface (blue). (b) A visualization of the unstructured tetrahedral mesh used for the FEA simulations. (c) A sample ground truth displacement field for a bending\_y load case from the High Signal dataset, showing the magnitude of displacement.  
*(You would insert a 3-panel image here corresponding to this caption)*

**Dataset Generation: Low Signal vs. High Signal Regimes:**  
A key hypothesis of this work is that a model's performance is highly dependent on the "signal strength" of the physical response in the training data. A high-magnitude force produces large, clear deformations (a "high signal"), making the relationship between inputs and outputs easier to learn. Conversely, a low-magnitude force produces subtler deformations (a "low signal"), which can be more challenging for a network to learn accurately. To investigate this, we generated two distinct datasets:

1. **Low Signal Dataset:** A set of **1500 simulations** generated with force\_magnitude sampled from a lower range of **[50,000 N to 100,000 N]**. This dataset contains a random mixture of all three load types (bending\_y, bending\_x, and torsion) and represents a scenario with subtle physical responses.
2. **High Signal Dataset:** A second, distinct set of **1000 simulations** generated with force\_magnitude sampled from a significantly higher range of **[200,000 N to 250,000 N]**. This dataset was generated **exclusively for the bending\_y load case** and represents a scenario where the input-output relationship is much more pronounced.

**Task Formulation:**  
These two datasets allow us to formulate three distinct and progressively challenging learning tasks, which correspond directly to the model evaluations presented in our results:

* **Generalist (Low Signal) Task:** A single model is trained on the complete 1500-sample **Low Signal Dataset**. This is the most complex task, as the model must learn to generalize across three qualitatively different physical responses (bending\_y, bending\_x, torsion) under challenging low-signal conditions.
* **Specialist (Low Signal) Task:** A model is trained *only* on the subset of the **Low Signal Dataset** corresponding to the bending\_y load case. This task assesses the model's ability to learn a single, specialized physical response under the same low-signal conditions.
* **Specialist (High Signal) Task:** A model is trained *only* on the 1000-sample **High Signal Dataset**. This task evaluates the model's performance on the same specialized bending\_y task but under ideal high-signal conditions, providing a benchmark for the best possible performance on this specific load case.

Each simulation was run to completion, and the full field data—including node coordinates, element topology, and the resulting displacement field—was saved to a dedicated HDF5 (.h5) file. This collection serves as the primary ground truth dataset.

**3.2. Data Representation and Preprocessing**

The output of the FEA solver is a high-dimensional field representing physical quantities such as displacement and stress. We chose to represent these fields by storing their pointwise values at each node of the simulation mesh. This approach was selected as it preserves the maximum possible fidelity of the simulation output, directly corresponding to the degrees of freedom solved for in the FEA process. Alternatives, such as storing element-wise averages, would result in a loss of resolution, while representations based on basis functions would introduce an additional layer of approximation. All geometric and field data were defined and stored using a standard 3D Cartesian coordinate system (X, Y, Z), which naturally aligns with the principal axes of the I-beam geometry.

While the unstructured HDF5 data is directly suitable for GNNs, the grid-based 3D U-Net architecture requires a structured, voxelized input. To accommodate this, we implemented a dedicated preprocessing pipeline to convert the unstructured data into a uniform grid representation, generating a corresponding compressed NumPy (.npz) file for each simulation. This process involves:

1. Grid Definition: To analyze the impact of input resolution on performance, we defined two distinct Cartesian grids: a low-resolution grid of 64x32x32 and a high-resolution grid of 96x48x48 voxels (Z, Y, X). A single, consistent bounding box was used for all simulations to ensure spatial alignment.
2. Field Interpolation: The unstructured displacement vectors were interpolated onto these regular grids using trilinear interpolation, resulting in a 3-channel (dx, dy, dz) tensor. Voxels outside the convex hull of the original mesh were assigned a fill value of zero.
3. Geometry Mask Creation: A binary geometry mask was created using nearest-neighbor interpolation to inform the U-Net of the beam's location within the voxel space.

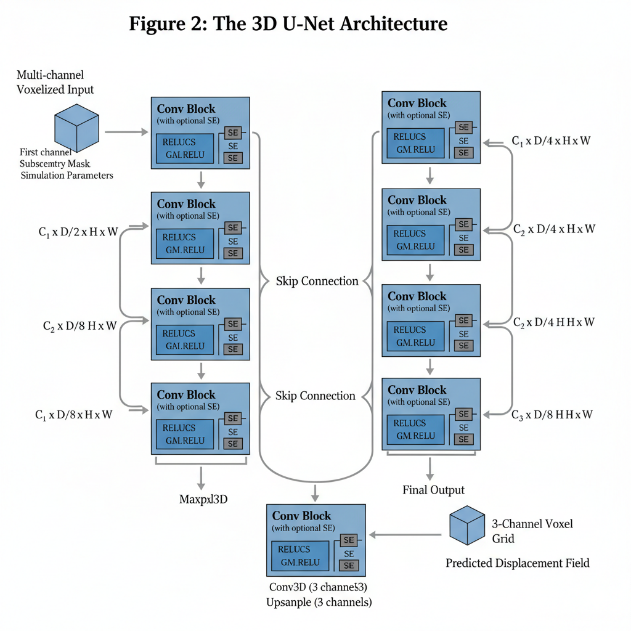
To ensure stable and efficient training, all input and output data were normalized. Deep learning models are sensitive to the scale of input features; without normalization, large-valued parameters (like Young's Modulus) could dominate the learning process and lead to unstable gradients. Similarly, normalizing the target output field sets an appropriate scale for the loss function. To this end, all scalar input parameters and the output displacement fields were scaled to a range of approximately [-1, 1] using min-max scaling, based on the global minimum and maximum values observed across the entire training dataset. The same scaling factors were stored and used to apply the inverse transformation during inference to return model predictions to their original physical units.

**3.3. Model Architectures**

To systematically evaluate the most effective approach for learning FEA surrogates, we implemented and compared two distinct classes of neural network architectures: grid-based Convolutional Neural Networks (CNNs) and mesh-based Graph Neural Networks (GNNs). Each class contains several variants to allow for a thorough analysis of performance, efficiency, and the impact of specific architectural features.

**3.3.1. Grid-Based Architecture: 3D U-Net**

To provide a strong baseline from the convolutional domain, we adapted the U-Net architecture to our 3D regression problem. The U-Net's encoder-decoder structure with skip connections, illustrated in **Figure 2**, is well-suited for capturing both local features and global context, which is essential for predicting a full displacement field.

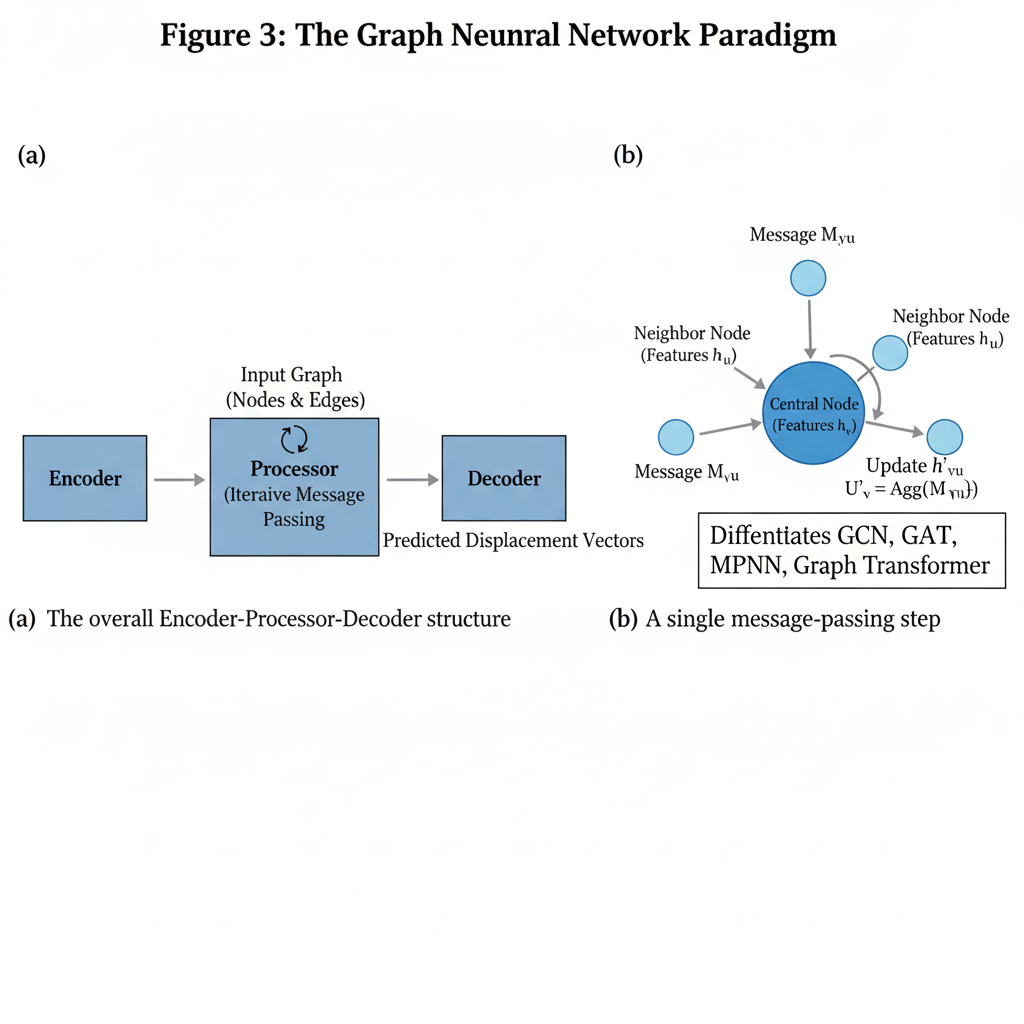


**Figure 2: The 3D U-Net Architecture.** The model takes a multi-channel voxelized input, where the first channel is the geometry mask and subsequent channels are broadcasted simulation parameters. The encoder (left) progressively downsamples the spatial resolution while increasing feature depth. The decoder (right) symmetrically upsamples the features, using skip connections (gray arrows) to re-introduce high-resolution information from the encoder path. Optional Squeeze-and-Excitation (SE) blocks provide channel-wise attention within each convolutional block. The final output is a 3-channel voxel grid representing the predicted displacement field.

* **Core Architecture:** As shown in Figure 2, our 3D U-Net consists of a contracting path (encoder) and an expansive path (decoder). Skip connections concatenate feature maps from the encoder to the corresponding layers in the decoder, which is crucial for preserving high-frequency details.
* **Input Formulation:** The input to the U-Net is a multi-channel 3D tensor. The first channel is the binary geometry mask, which explicitly defines the shape of the I-beam within the voxel grid. Subsequent channels are created by broadcasting each of the normalized scalar simulation parameters (e.g., force magnitude, Young's modulus, flange width) into its own full-resolution 3D channel. This "parameter embedding" technique ensures that every convolutional filter at every location has access to the global physical context of the simulation.
* **Architectural Variants:** As shown in our results, we evaluated two main variants based on the code in unet\_variants.py:
  1. **UNet3D:** A computationally efficient model with a baseline channel count of 32 in the first layer, which doubles with each downsampling step.
  2. **Attention-Enhanced U-Net (UNet3D + Attn):** To test the hypothesis that focusing on salient features can improve performance, we integrated a **Squeeze-and-Excitation (SE) block** into each convolutional layer. The SE\_Block3D is a channel-wise attention mechanism that adaptively recalibrates the feature maps. It "squeezes" global spatial information into a channel descriptor and then uses this to compute channel-wise attention weights, effectively allowing the network to emphasize more informative feature channels and suppress less useful ones.

**3.3.2. Mesh-Based Architectures: Graph Neural Networks**

GNNs represent a more natural paradigm for this problem, as they operate directly on the unstructured FEA mesh, thereby preserving the exact geometry and topology without any discretization error from voxelization. The general GNN paradigm we employ is shown in **Figure 3**.

****

**Figure 3: The Graph Neural Network Paradigm.** (a) The overall Encoder-Processor-Decoder structure. The input graph's node features are encoded into a latent space, iteratively refined through multiple message-passing layers in the processor, and finally decoded into the predicted displacement vectors. (b) A conceptual view of a single message-passing step, where a central node aggregates information from its neighbors to update its own feature representation. The specific mathematical formulation of this aggregation and update step is what differentiates the GCN, GAT, MPNN, and Graph Transformer architectures.

* **Graph Representation:** The FEA mesh, composed of tetrahedral elements, was converted into an undirected graph structure suitable for PyTorch Geometric. The nodes of the graph directly correspond to the nodes of the FEA mesh. The graph's edges are derived by extracting all unique edges from the tetrahedral elements.
* **Node Feature Engineering:** Each node in the graph is initialized with a feature vector that encodes both its local position and the global context of the simulation. This vector is constructed by concatenating:
  1. The node's 3D Cartesian coordinates (pos).
  2. The full set of normalized scalar simulation parameters, which are repeated for every node.
  3. A **conditional load-type encoding**: For the *Generalist (multimodal)* model, the categorical load\_type is one-hot encoded into a 3-dimensional vector (e.g., [1,0,0] for bending\_y). For the *Specialist (unimodal)* models, it is encoded as a single scalar. This distinction is critical, as the one-hot encoding provides a clear, non-ordinal signal that allows the generalist model to effectively learn the different physical responses.
* **Architectural Variants:**  All GNNs follow the encoder-processor-decoder design shown in Figure 3a. An input linear layer encodes the node features into a higher-dimensional hidden state. A series of "processor" layers then perform message passing , illustrated in Figure 3b, to iteratively update these hidden states. Finally, a linear decoder maps the final hidden states to the predicted 3D displacement vectors. We evaluated four GNN processor types:
  1. **GCN (Graph Convolutional Network):** Uses GCNConv layers, which perform isotropic aggregation by averaging the features of neighboring nodes. It serves as a foundational GNN baseline. Mathematically, The feature update is a normalized sum of neighboring node features, transformed by a learnable weight matrix **W**<sup>(l)</sup>:  
     **h***v*<sup>(l+1)</sup> = σ ( **W**<sup>(l)</sup> Σ\_{u ∈ N(v) ∪ {v}} (1/c\_{vu}) **h***u*<sup>(l)</sup> )  
     where *c\_vu* is a normalization constant and σ is the activation function.
  2. **GAT (Graph Attention Network):** Employs GATConv layers, which enhance GCN by introducing a self-attention mechanism. This allows the model to learn different weights for different neighbors, focusing on the most relevant information during aggregation. Mathematically, The model learns attention coefficients α\_{vu} that determine the importance of neighbor *u*'s features to node *v*:  
     **h***v*<sup>(l+1)</sup> = σ ( Σ\_{u ∈ N(v) ∪ {v}} α\_{vu} **W**<sup>(l)</sup> **h***u*<sup>(l)</sup> )  
     The attention coefficients α\_{vu} are computed using a learnable attention mechanism, allowing for a weighted, anisotropic aggregation.
  3. **MPNN (Message Passing Neural Network):** Implemented using the expressive MetaLayer framework. This provides a more general form of message passing where separate neural networks (EdgeModel and NodeModel) are explicitly learned to first create "messages" based on pairs of connected nodes, and then update each node based on the sum of its incoming messages. Residual connections are used after each update to improve gradient flow. This provides a more general and expressive form of message passing by using distinct learnable functions (MLPs) for message creation (ψ) and node updates (φ):  
     **m***{vu} = ψ(****h****v<sup>(l)</sup>,****h****u<sup>(l)</sup> )****h****v<sup>(l+1)</sup> = φ(****h****v<sup>(l)</sup>, Σ*{u ∈ N(v)} **m**\_{vu} )  
     This explicit formulation allows the model to learn more complex interactions between nodes.
  4. **Graph Transformer:** Utilizes TransformerConv layers, representing the most powerful architecture in our study. This layer applies multi-head self-attention to the local neighborhood of each node, allowing it to learn highly complex and adaptive aggregation functions, capturing intricate dependencies between nodes.

**3.4. Physics-Informed Learning Framework**

To move beyond a purely data-driven approach and embed physical knowledge into our models, we integrated a Physics-Informed Neural Network (PINN) framework. The primary goal of the PINN component is not to solve the PDE from scratch, but rather to act as a physics-based regularizer, ensuring the model's predictions adhere to the governing laws of solid mechanics and thereby improving generalization.

**3.4.1. Governing Equations and Loss Formulation**  
The physical behavior of a linearly elastic, isotropic solid in static equilibrium is governed by the **Navier-Cauchy equations**. In vector form, the equation is:

μ∇²u + (μ + λ)∇(∇ ⋅ u) + F = 0

where **u** is the displacement vector field, **F** is the body force vector (assumed to be zero in our case), and *μ* and *λ* are the material-specific Lamé parameters, which are derived from the Young's Modulus and Poisson's Ratio.

Our total loss function is a composite of a data-driven term and a physics-based term, weighted by a dynamic parameter α(t):

L\_total = L\_data + α(t) \* L\_physics

* **Data Loss (L\_data):** This is the Mean Squared Error (MSE) between the GNN's predicted displacement vectors at the mesh nodes and the ground truth displacements from the FEA solver. This term ensures the model remains faithful to the simulation data.
* **Physics Loss (L\_physics):** This term quantifies the extent to which the model's predictions violate the Navier-Cauchy equations. It is calculated as the mean squared residual of the governing PDE over a large set of collocation points sampled randomly throughout the beam's volume at each training step. Crucially, all spatial derivatives required to compute the PDE residual (e.g., ∇u, ∇²u) are calculated analytically using **automatic differentiation**. This is a key advantage of using neural networks, as it allows us to approximate the differential operators with high precision by differentiating the network's output with respect to its input spatial coordinates.

**3.4.2. Treatment of Boundary Conditions**  
In many "classic" PINN applications that solve PDEs from scratch, an explicit boundary condition loss term (L\_bc) is required. However, in our surrogate modeling framework, this is unnecessary. The Dirichlet boundary conditions (i.e., the zero-displacement constraint at the fixed end of the beam) are already present in the ground truth data. By training the model to minimize L\_data, it implicitly learns to satisfy these boundary conditions. The physics loss L\_physics then regularizes the solution *within* the domain, conditioned on these data-enforced boundaries.

**3.5. Curriculum Learning for PINN Stabilization**

A significant challenge in training PINNs is balancing the gradients from the data and physics loss terms. Our initial attempts to train the PINN-enhanced GNNs with a fixed, non-zero weight α from the beginning of training proved to be unstable.

**3.5.1. Observed Instability**  
The naive joint-training approach consistently failed to converge to a meaningful solution. The typical failure mode observed was a decreasing training loss while the validation loss either fluctuated erratically or steadily increased. This behavior indicates that the optimizer was struggling with conflicting or poorly scaled gradients from the L\_data and L\_physics terms. The high-frequency nature of the second-order derivatives in the physics loss can easily dominate the training process in early stages before the model has learned a reasonable approximation of the solution, preventing the model from learning the fundamental input-output mapping.

**3.5.2. The Successful Curriculum Strategy: Fine-Tuning with Annealing**  
To overcome this instability, we developed a robust two-stage curriculum learning strategy, reframing the task from joint training to **pre-training and fine-tuning**:

1. **Stage 1: Data-Driven Pre-training.** First, a GNN model is trained to convergence on the dataset using *only* the data loss (L\_data, i.e., α = 0). This allows the model to learn a strong, stable, and accurate mapping from the input parameters to the displacement field without any interference from the physics loss.
2. **Stage 2: Physics-Informed Fine-tuning.** The weights of the converged, pre-trained model are then loaded. In this second stage, the physics loss term is introduced. The weight α is not fixed but is **annealed**—it is gradually increased from 0 to its final target value over a set number of epochs.

This fine-tuning approach proved to be critical for success. By starting from a model that already provides a very good solution, the physics loss acts as a gentle regularizer, "nudging" the pre-trained solution into a nearby region of the parameter space that better conforms to the Navier-Cauchy equations. This prevents the gradient conflicts observed in the naive approach and leads to a stable decrease in both training and validation loss, ultimately yielding a more accurate and physically plausible final model.

**3.6. Evaluation and Benchmarking**

To provide a comprehensive and rigorous assessment of our models, we evaluated their performance from two critical perspectives: predictive accuracy and computational efficiency. All evaluations were performed on a held-out, unseen test set, ensuring an unbiased measure of each model's generalization capabilities.

**3.6.1. Predictive Accuracy Metrics**

We used a suite of three metrics to quantify the accuracy of the predicted displacement fields against the ground truth FEA results.

* **Mean Absolute Error (MAE):** This metric provides a direct, interpretable measure of the average pointwise error in physical units. It is calculated as the mean of the absolute differences between the predicted displacement vectors (**u***pred*) and the ground truth vectors (**u***true*) over all *N* nodes in a sample:  
  MAE = (1/N) \* Σ\_{i=1 to N} | **u***pred,i* - **u***true,i* |  
  The final MAE reported is the average over all samples in the test set, with units of millimeters (mm).
* **Relative L2 Error (%):** To provide a normalized measure of error that is independent of the absolute magnitude of displacement, we use the Relative L2 Error. This is particularly important for comparing performance across different load cases and signal strengths (e.g., low-force vs. high-force scenarios). It is calculated as the ratio of the L2 norm of the error field to the L2 norm of the ground truth field, expressed as a percentage:  
  Relative L2 Error = ( ||**u***pred* - **u***true*||₂ / ||**u***true*||₂ ) \* 100%  
  A lower percentage indicates a more accurate field-level prediction.
* **R² Score (Coefficient of Determination):** A standard statistical measure, the R² score represents the proportion of the variance in the ground truth data that is predictable from the model's predictions. It provides a valuable assessment of the model's goodness of fit. An R² score of 1.0 indicates a perfect fit, while a score of 0 indicates the model performs no better than a constant baseline predicting the mean of the data.

While we focus our evaluation on these full-field metrics, their high fidelity directly implies accuracy on derived Quantities of Interest (QoI), such as maximum deflection, as these are direct functions of the predicted field.

**3.6.2. Computational Performance Metrics**

For surrogate models to be practical, they must offer a significant speed advantage over the original solver. We use two key metrics to quantify this efficiency.

* **Inference Time (ms):** This is the wall-clock time required for a trained model to perform a single forward pass and generate a prediction for one sample from the test set. The reported time is averaged over the entire test set to ensure a stable measurement. This metric directly quantifies the speedup of the surrogate compared to the minutes or hours required for a single FEA simulation.
* **Model Complexity (Parameters, M):** The number of trainable parameters in a model serves as a direct proxy for its size and memory footprint. Reported in millions (M), this metric is crucial for understanding the trade-off between model accuracy and its computational requirements for both training and deployment, especially in resource-constrained environments.

**3.6.3. Experimental Setup**

All models were trained and evaluated using a consistent experimental setup to ensure fair and reproducible comparisons.

* **Data Split:** The datasets were split into training (80%), validation (10%), and test (10%) sets. The validation set was used for hyperparameter tuning and to monitor for overfitting during training, while the test set was strictly held out and used only for the final performance evaluation reported in our results.
* **Hardware:** All training and inference benchmarks were conducted on a consistent hardware platform, specifically using an **NVIDIA GeForce RTX 4050 with 6 GB of VRAM.**

**4. Experiments and Results**

We conducted a comprehensive set of experiments to evaluate the performance of the proposed architectures across the defined learning tasks. The models were evaluated on the held-out test set using the metrics defined in Section 3.6. The complete quantitative results are presented in Table 2, which serves as the primary reference for the analyses in the following subsections.

**Table 2: Comprehensive Evaluation Results**

*This table is a formatted version of your results image. It will be the central piece of evidence for the entire section.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Task** | **MAE (mm) ↓** | **R-L2 (%) ↓** | **R² Score ↑** | **Inference (ms) ↓** | **Params (M) ↓** |
| **---** | **LOW SIGNAL (Generalist)** | **---** | **---** | **---** | **---** | **---** |
| GCN | Generalist | 0.0002 | 9.7473 | 0.9899 | 1.2365 | 0.0521 |
| GAT | Generalist | 0.0002 | 6.8524 | 0.9950 | 0.9444 | 0.3992 |
| MPNN | Generalist | 0.0001 | 3.8348 | 0.9984 | 0.1441 | 0.2990 |
| **MPNN-PINN** | **Generalist** | **0.0001** | **3.6751** | **0.9986** | **0.5367** | **0.2990** |
| Graph Transformer | Generalist | 0.0001 | 3.8524 | 0.9984 | 0.2903 | 1.5800 |
| U-Net (Low-Res) | Generalist | 0.0000 | 25.4867 | 0.9244 | 1.4520 | 5.6107 |
| U-Net (High-Res) | Generalist | 0.0000 | 26.2694 | 0.9194 | 1.8244 | 5.6107 |
| U-Net Small+Attn | Generalist | 0.0000 | 25.8114 | 0.9219 | 2.4954 | 5.6243 |
| **---** | **LOW SIGNAL (Specialist)** | **---** | **---** | **---** | **---** | **---** |
| GCN | Specialist | 0.0001 | 9.1548 | 0.9880 | 0.7837 | 0.0518 |
| MPNN | Specialist | 0.0001 | 4.0021 | 0.9977 | 0.1726 | 0.2988 |
| GAT | Specialist | 0.0001 | 5.8954 | 0.9950 | 2.3798 | 0.3990 |
| **---** | **HIGH SIGNAL (Specialist)** | **---** | **---** | **---** | **---** | **---** |
| GCN | Specialist | 0.0128 | 8.7901 | 0.9889 | 0.6130 | 0.0518 |
| GAT | Specialist | 0.0089 | 5.4931 | 0.9957 | 2.6013 | 0.3990 |
| MPNN | Specialist | 0.0063 | 4.0354 | 0.9977 | 0.1627 | 0.2988 |
| **MPNN-PINN** | **Specialist** | **0.0055** | **3.5789** | **0.9982** | **0.5751** | **0.2988** |
| **Graph Transformer** | **Specialist** | **0.0042** | **2.6466** | **0.9990** | **0.2237** | **1.5798** |
| U-Net+Attn (High-Res) | Specialist | 0.0001 | 13.0801 | 0.9656 | 1.7742 | 5.6243 |