MESHFILM: FAST PHYSICS SURROGATES ON UNSTRUCTURED MESHES VIA NODE-WISE MODULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural surrogates for physics on unstructured meshes usually exchange information between nodes in every layer, by message passing along mesh edges or through a small set of slice tokens. We study how far a surrogate can go when this exchange happens once. **MeshFiLM** embeds every node independently, pools the embeddings once into a context vector for each node, and decodes the fields with a pointwise MLP whose hidden features are modulated in every layer by feature-wise linear modulation, computed from the node's own embedding and its context. We compare two pooling operators: statistics pooling gives every node the same summary, and routed pooling lets each node read its own mixture of learned region summaries. On five PLAID benchmarks and the surface and volume fields of `AhmedML`, we compare models in one training pipeline using the same node features and hardware. MeshFiLM trains 4.9-8.4$\times$ faster per epoch than the fastest baseline on PLAID and 1.7-2.6$\times$ faster on `AhmedML`, with fewer parameters. At equal epochs, its routed variant has lower mean error than the strongest baseline in four of seven settings and comparable error in a fifth. In four PLAID settings with equal-time runs, MeshFiLM-R has 21-64% lower error than the strongest baseline on three; on the fourth, both MeshFiLM variants remain about 4% higher.

1 INTRODUCTION

Design studies in structural mechanics and aerodynamics run a numerical solver for many shapes and operating conditions. Neural surrogates trained on previous simulations can replace part of these runs (Pfaff et al., 2021; Casenave et al., 2023). Industrial data make this difficult. Samples live on unstructured meshes whose size and connectivity change from one sample to the next, and the targets combine dense fields with a few sample-level quantities such as a mass flow or a maximum stress (Casenave et al., 2025). Training cost also matters because, until generic foundation models for physics become available, a surrogate is retrained for each new product family or design space.

Most mesh surrogates exchange information between nodes in every layer. Graph networks (Sanchez-Gonzalez et al., 2020) such as MeshGraphNets (MGN, Pfaff et al., 2021) update each node from its neighbours along mesh edges, so information travels one hop per layer and each layer costs time proportional to the number of edges. Reaching distant regions of a fine mesh requires many layers or coarsened graphs (Fortunato et al., 2022; Cao et al., 2023), and long message chains compress information from many nodes into fixed size vectors (Alon & Yahav, 2021). Fourier neural operators avoid edges but need a regular grid, a learned deformation to one, or an intermediate latent grid (Li et al., 2021; 2023a;b). Transformer solvers communicate through a small set of tokens. Transolver (Wu et al., 2024) softly assigns nodes to slices, applies attention among the slice tokens and maps the result back to the nodes in every layer. Transolver++ (Luo et al., 2025) adapts the assignment to each point and parallelizes it across GPUs. UPT, LNO and AB-UPT (Alkin et al., 2024; Wang & Wang, 2024; Alkin et al., 2025) process a latent token set and read the outputs at query points with cross-attention. These models scale linearly with the number of nodes, but they either repeat the node-token exchange in every layer or add a latent processor and an attention readout.

Conditional neural fields take another route: a single coordinate network predicts the field at any query point, and a code adapts it to each sample (Lu et al., 2021; Park et al., 2019a; Dupont et al., 2022). CORAL and DINo obtain the code by fitting it to each sample (Serrano et al., 2023; Yin et al., 2023). MARIO (Catalani et al., 2026) conditions a Fourier-feature network with layer-wise

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

shifts generated from a sample level code; for non-parametric shapes, this code comes from a signed distance field fitted in a first training stage. BlendedNet (Sung et al., 2025) regresses design parameters from a point cloud and feeds them to a FiLM network. Enf2enf (Catalani et al., 2025) replaces the single code with localized latents read by distance-windowed cross-attention, again fitted per sample. In these models, either every point of a sample receives the same modulation, or the latent representation needs a separate fitting stage. Point cloud networks also exchange information once: PointNet (Qi et al., 2017) concatenates a max pooled feature of all points to the feature of every point before a pointwise head, and this design predicts flow fields around irregular geometries (Kashefi et al., 2021) and, as the branch of a DeepONet, nonlinear fields in solids under varying loads (Park & Kang, 2026). Modulation that varies across positions or nodes is used in image synthesis (Park et al., 2019b) and in graph networks (Brockschmidt, 2020).

MeshFiLM combines these routes (Figure 1). An MLP embeds every node from its coordinates and local features, a single pooling step builds a context, and a pointwise MLP decodes the fields. In every decoder layer, feature-wise linear modulation (Perez et al., 2018) rescales and shifts the hidden features of node j with parameters computed from its embedding $\mathbf{z}_j$ and a context vector $\mathbf{c}_j$, so that, unlike in PointNet, the pooled information conditions every layer and the modulation differs across nodes. Our hypothesis is that, for the steady problems we consider, a prediction can be made node-specific without re-reading the node set in every layer: the embedding locates the node, the context summarizes the rest of the mesh, and FiLM combines the two. We study two ways to build $\mathbf{c}_j$, which give two variants. MeshFiLM-S uses statistics pooling, the mean, maximum and standard deviation of all node embeddings (Zaheer et al., 2017; Corso et al., 2020), so every node receives the same context. MeshFiLM-R uses routed pooling: it softly assigns nodes to K learned regions, as in Slot Attention and Transolver slices (Locatello et al., 2020; Wu et al., 2024), and each node reads its own mixture of the region summaries. The two variants also differ in their encoder and in where sample-level inputs enter (Section 2.1). Both are trained end to end in one stage, use no mesh edges in the forward pass, and exchange information between nodes only in the pooling step.

We train MeshFiLM, MGN, MGN-Tree, Transolver and Transolver++ in one pipeline, with the same input features and hardware, on five PLAID benchmarks (Casenave et al., 2025) and on the surface and volume fields of `AhmedML` (Ashton et al., 2024). Our contributions are:

- MeshFiLM, a single stage surrogate for unstructured meshes in which nodes exchange information once, in a pooling step, and a pointwise decoder is modulated in every layer by node-wise FiLM, with two pooling operators (Section 2).
- A controlled comparison of accuracy and training cost (Section 3.2). MeshFiLM trains 1.7-8.4$\times$ faster per epoch than the fastest baseline, with fewer parameters. With routed pooling, it matches or improves on the most accurate baseline in five of seven settings at equal epochs, and at equal training time its error is 21-64% lower on `Tensile2d`, `VKI-LS59` and `Rotor37`; it remains 4% behind Transolver++ on `2D_profile`.
- An analysis of the conditioning and of the two operators (Sections 3.3 and 3.4). On `Tensile2d`, feeding the context to the decoder by concatenation instead of FiLM increases the error by 7-28%. Replacing only the routed context by statistics pooling shows that the operator changes the balance between field and scalar errors on `Tensile2d`. On `Rotor37`, the operator does not explain the gap between the two variants, and passing the operating point through the node features lowers the error of MeshFiLM-R by 34%.

We also report MeshFiLM entries on the public PLAID leaderboards, which score predictions on private test sets (Section 3.5).

2 MeshFiLM

Problem: We consider supervised surrogate modeling for steady-state physics simulations defined on variable size meshes. A sample is a mesh with N nodes. Node i has coordinates $\mathbf{x}_i \in \mathbb{R}^{d_x}$ and local features $\mathbf{f}_i \in \mathbb{R}^{d_f}$ computed from the mesh before training, such as boundary indicators or distances to walls. The targets are fields $\mathbf{y}_i \in \mathbb{R}^{d_y}$ at every node and, for some datasets, sample-level scalars $\mathbf{s} \in \mathbb{R}^{d_s}$. The number of nodes and the connectivity change across samples. Field predictions must be equivariant to permutations of the nodes, and scalar predictions invariant to them. MeshFiLM sees the mesh only through the node inputs; no edges enter the forward pass.

Overview: MeshFiLM has three parts (Figure 1). An MLP E encoder maps each node input $\mathbf{a}_i = [\mathbf{x}_i \,\|\, \mathbf{f}_i]$ to a node embedding $\mathbf{z}_i = E(\mathbf{a}_i) \in \mathbb{R}^H$, the same embeddings are used by the context operator and by the decoder. The context operator reads the set $\{\mathbf{z}_i\}_{i=1}^N$ once and returns a context vector $\mathbf{c}_j$ for every node j and a summary $\mathbf{g}$ of the sample. A modulated decoder predicts $\mathbf{y}_j$ from $\mathbf{x}_j$, $\mathbf{z}_j$ and $\mathbf{c}_j$, and a scalar head predicts $\mathbf{s}$ from $\mathbf{g}$. In the forward pass, the context operator is the only operation that combines several nodes: the encoder and the decoder share their weights across nodes but process each node on its own.

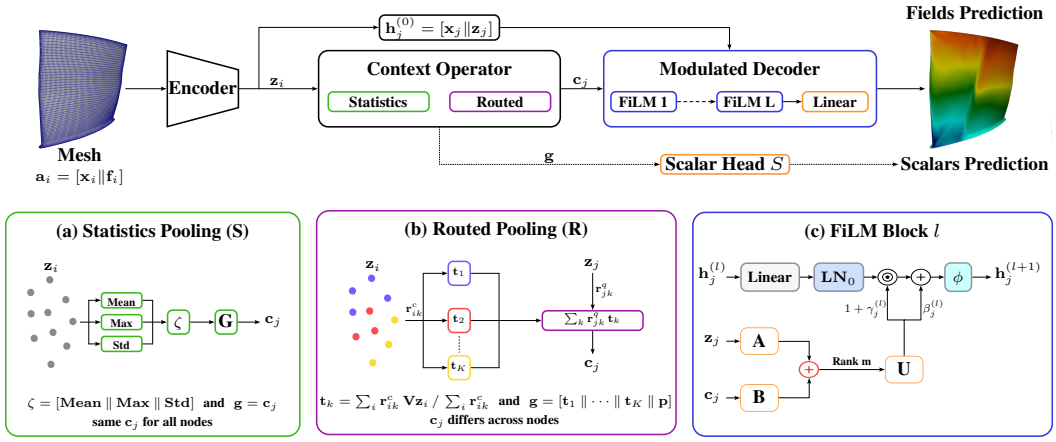

Figure 1: **MeshFiLM.** (a) MeshFiLM-S: statistics pooling gives every node the same context, built from the per-channel mean, maximum and standard deviation of the node embeddings $\mathbf{z}_i$. (b) MeshFiLM-R: routed pooling writes the embeddings into K soft regions with a write router r^{c}, and each node j reads its own mixture of the region summaries $\mathbf{t}_k$ with a read router r^{q}. Indices i and j run over the same nodes unless the context is subsampled. (c) In every decoder block, a rank-m head maps $\mathbf{z}_j$ and $\mathbf{c}_j$ to the FiLM parameters $(\boldsymbol{\gamma}_j, \boldsymbol{\beta}_j)$, its output is zero at initialization. Sample-level inputs $\mathbf{s}_{\text{in}}$ enter MeshFiLM-S through $\mathbf{f}_i$ at every node and MeshFiLM-R once, as $\mathbf{p} = P(\mathbf{s}_{\text{in}})$ appended to the context and to the input of the scalar head (Section 2.1).

2.1 Context Operators

Statistics Pooling (MeshFiLM-S)

$$\boldsymbol{\zeta} = \big[\,\text{mean} \,\|\, \max \,\|\, \text{std}\,\big], \qquad \mathbf{c}_j = G(\boldsymbol{\zeta}) \quad \text{for all } j, \qquad \mathbf{g} = \mathbf{c}_j \tag{1}$$

where the statistics are taken per channel over the N nodes and G is an MLP. Mean, maximum and standard deviation are standard permutation-invariant aggregators. All nodes receive the same context.

Routed pooling (MeshFiLM-R)

$$r_{ik}^{\mathrm{c}} = \mathrm{softmax}_k\big(\mathbf{w}_k^{\mathrm{c}\top}\mathbf{z}_i/\tau_{\mathrm{c}}\big), \qquad \mathbf{t}_k = \frac{\sum_{i=1}^N r_{ik}^{\mathrm{c}}\,\mathbf{V}\mathbf{z}_i}{\sum_{i=1}^N r_{ik}^{\mathrm{c}}}, \quad k = 1,\dots,K \tag{2}$$

$$r_{jk}^{\mathrm{q}} = \mathrm{softmax}_k\big(\mathbf{w}_k^{\mathrm{q}\top}\mathbf{z}_j/\tau_{\mathrm{q}}\big), \qquad \mathbf{c}_j = \Big[\textstyle\sum_{k=1}^K r_{jk}^{\mathrm{q}}\mathbf{t}_k \Big\| \mathbf{p}\Big], \quad \mathbf{g} = [\mathbf{t}_1\|\cdots\|\mathbf{t}_K\|\mathbf{p}] \tag{3}$$

The write router r^{c} softly assigns each node to K regions, and the region summary $\mathbf{t}_k$ is the weighted mean of the projected embeddings $\mathbf{V}\mathbf{z}_i$ of the nodes written to region k. The regions are soft and are defined in embedding space, so a region need not be connected in the mesh. The read router r^{q} lets node j read its own mixture of the K summaries. Equation 2 has the same form as the slices of Transolver (Wu et al., 2024) and the slot update of Slot Attention (Locatello et al., 2020). Pooling by multihead attention (Lee et al., 2019) also summarizes a set with K learned queries, but normalizes over the nodes, whereas Equation 2 normalizes over the regions, so that the regions compete for each

node. MeshFiLM applies this aggregation once, before decoding, with separate routers for writing and reading, and the regions never interact with each other. The context now varies across nodes: the matrix $[\mathbf{c}_1, \ldots, \mathbf{c}_N]$ has rank at most K, while statistics pooling gives rank one.

Sample-level inputs: In some datasets, each sample also has a few scalar inputs $\mathbf{s}_{\text{in}}$, which are the same at every node. MeshFiLM-S appends $\mathbf{s}_{\text{in}}$ to the features $\mathbf{f}_i$, allowing the encoder to combine the operating conditions with each node's position and local features before pooling. MeshFiLM-R encodes the sample-level inputs once, $\mathbf{p} = P(\mathbf{s}_{\text{in}})$, and appends $\mathbf{p}$ to the context and to the input of the scalar head (Equation 3), so that its routers depend on the node coordinates and local features, but not directly on $\mathbf{s}_{\text{in}}$. The two variants therefore differ in three places: the context operator, the encoder (Table 4), and where $\mathbf{s}_{\text{in}}$ enters. Section 3.3 isolates the effect of the context operator by replacing it while keeping the other parts of MeshFiLM-R fixed.

Two properties follow from Equation 2. Each region summary is a ratio of two sums over nodes, so a random subset of the nodes gives a consistent estimate of it, whose variance grows as fewer nodes carry the weight of the region (Appendix K); we use this to encode large meshes from a subset of context nodes (Section 3.4). The mean and standard deviation of Equation 1 share this property, but the maximum does not: the maximum over a random subset underestimates the maximum over the mesh, by an amount that depends on the subset size. Each summary also lies in the convex hull of $\{\mathbf{V}\mathbf{z}_i\}_{i=1}^N$: for any direction $\mathbf{a}$, $\mathbf{a}^\top \mathbf{t}_k \le \max_i \mathbf{a}^\top \mathbf{V}\mathbf{z}_i$, with equality only if the router puts all the weight of region k on the maximizing nodes. Routed pooling therefore does not provide the extremes and the spread of the embeddings explicitly, whereas statistics pooling does, which may matter for sample-level targets defined as maxima, such as the three scalars of `Tensile2d` (Section 3.3).

2.2 Modulated Decoder With Node-wise FiLM

The decoder starts from $\mathbf{h}_j^{(0)} = [\mathbf{x}_j \,\|\, \mathbf{z}_j]$ and applies L blocks, $\ell = 0, \ldots, L-1$:

$$\tilde{\mathbf{h}}_j^{(\ell)} = \mathrm{LN}_0\big(\mathbf{W}_\ell\, \mathbf{h}_j^{(\ell)} + \mathbf{b}_\ell\big) \tag{4}$$

$$\big[\boldsymbol{\gamma}_j^{(\ell)} \,\|\, \boldsymbol{\beta}_j^{(\ell)}\big] = \mathbf{U}_\ell\big(\mathbf{A}_\ell\, \mathbf{z}_j + \mathbf{B}_\ell\, \mathbf{c}_j\big) + \mathbf{d}_\ell \tag{5}$$

$$\mathbf{h}_j^{(\ell+1)} = \phi\big((\mathbf{1} + \boldsymbol{\gamma}_j^{(\ell)}) \odot \tilde{\mathbf{h}}_j^{(\ell)} + \boldsymbol{\beta}_j^{(\ell)}\big) \tag{6}$$

and a linear layer outputs $\hat{\mathbf{y}}_j = \mathbf{W}_{\text{out}}\mathbf{h}_j^{(L)} + \mathbf{b}_{\text{out}}$. Here LN_0 is layer normalization (Ba et al., 2016) without learned scale and shift, so the FiLM parameters are the only feature-wise affine transform in a block, and ϕ is the SiLU activation (Elfwing et al., 2018). $\mathbf{W}_0 \in \mathbb{R}^{H\times(d_x+H)}$, $\mathbf{W}_\ell \in \mathbb{R}^{H\times H}$ for $\ell \ge 1$, $\mathbf{A}_\ell \in \mathbb{R}^{m\times H}$, $\mathbf{B}_\ell \in \mathbb{R}^{m\times d_c}$ and $\mathbf{U}_\ell \in \mathbb{R}^{2H\times m}$ are learned matrices, $\mathbf{b}_\ell$ and $\mathbf{d}_\ell$ learned biases, and d_c is the width of the context. The head of Equation 5 projects the node embedding and the context to width m, adds them, and maps the sum to the $2H$ FiLM parameters. Both variants use the same head of rank $m = 32$, a factorization in the spirit of LoRA (Hu et al., 2022); only the input width of $\mathbf{B}_\ell$ differs (Section 2.1). We initialize $\mathbf{U}_\ell$ and $\mathbf{d}_\ell$ to zero, so that $\boldsymbol{\gamma} = \boldsymbol{\beta} = \mathbf{0}$ at initialization and the decoder starts as an unmodulated normalized MLP, as in the AdaLN-Zero blocks of DiT (Peebles & Xie, 2023). Because $\mathbf{A}_\ell\mathbf{z}_j$ depends on the node, the modulation can differ across nodes even when all nodes share the same context.

2.3 Scalar Outputs, Training and Cost

Sample-level scalars are predicted by an MLP from the summary of the context operator, $\hat{\mathbf{s}} = S(\mathbf{g})$, which is invariant to node permutations. We minimize $\mathcal{L} = \mathcal{L}_{\text{field}} + \lambda_s \mathcal{L}_{\text{scalar}}$, with mean squared errors on standardized targets. Dataset-specific auxiliary losses, such as the gradient term used on `Tensile2d`, are described in Appendix I. The controlled comparison of Section 3.2 uses the field loss only.

Cost: MeshFiLM spends $\mathcal{O}(NH)$ once in statistics pooling, or $\mathcal{O}(NKH)$ once in routed pooling, and $\mathcal{O}(N(H^2 + mH))$ per decoder block, where every operation acts on one node at a time and can be batched or split into chunks of query nodes. For comparison, with E mesh edges, an MGN

layer updates all edges and nodes in $\mathcal{O}((N+E)H^2)$, and a Transolver layer with M slices costs $\mathcal{O}(NMH + M^2H + NH^2)$; both exchange information between nodes in every layer. For fixed widths and $E = \mathcal{O}(N)$, all three costs are linear in N. Section 3.2 reports measured training times under one pipeline, and Appendix B details the count.

3 EXPERIMENTS

3.1 SETUP

Datasets: We use the five PLAID benchmarks (Casenave et al., 2025) and `AhmedML` (Ashton et al., 2024) (Table 6 in Appendix C). The PLAID benchmarks cover nonlinear solid mechanics (`Tensile2d`, `2D_MultiScHypEl`) and compressible RANS flows (`VKI-LS59`, `Rotor37`, `2D_profile`), with 5.7k to 37k nodes per mesh on average and geometry that varies across samples; in `2D_MultiScHypEl`, the number of holes also varies. `AhmedML` contains 3D incompressible flows around 500 variants of the Ahmed body. We predict the surface fields and the volume fields as two separate settings, giving seven settings in total.

Subsampled meshes on `AhmedML`: An `AhmedML` mesh has 1.1M surface or 26.8M volume nodes on average, too many for one forward pass. Every model is trained on random subsets of 131,072 nodes per mesh and evaluated on all nodes of every test mesh, split at random into chunks of 131,072 nodes that are predicted independently (Appendix E). A chunk is a uniform sample of the whole mesh, 12% of the surface nodes or 0.5% of the volume nodes, and for MeshFiLM, also its own context set unless stated otherwise.

Baselines: MGN (Pfaff et al., 2021) and MGN-Tree (Ripken et al., 2023) pass messages along mesh edges, Transolver (Wu et al., 2024) and Transolver++ (Luo et al., 2025) communicate through slice tokens. All models are implemented in one code base and share the data loading, the optimizer, the precision and the hardware. Within a benchmark, every model consumes exactly the same node inputs $\mathbf{a}_i$, produced by the same feature construction code, nothing in Table 1 depends on a model-specific input pipeline. Hyperparameters of every model are selected on validation (Appendix E).

Two protocols: In the controlled protocol, every model is trained for the same number of epochs (*equal epochs*, Appendix E), and MeshFiLM is also trained for the wall-clock time that Transolver++ needs for this budget (*equal time*). We report the field RRMSE of PLAID (Appendix D) averaged over the output fields (*macro RRMSE*), as mean $\pm$ standard deviation over three seeds. In the public protocol, we submitted single runs with per-benchmark settings (Table 27) to the PLAID leaderboards, which score fields and scalars together on private test sets (Table 3, Appendix H).

3.2 ACCURACY AND COST UNDER ONE PIPELINE

Table 1 and Figure 2 summarize the controlled comparison. MeshFiLM is 4.9-8.4$\times$ faster per epoch than the fastest baseline on the PLAID benchmarks, 1.7-2.6$\times$ faster on `AhmedML`, and 8-19$\times$ faster than MGN. It also has fewer parameters: 0.44-0.65M against 0.73-0.92M for the baselines on PLAID, and 0.45-0.63M against 2.8-3.1M on `AhmedML`.

Equal epochs: MeshFiLM-R matches or improves on the strongest baseline in five of seven settings: `Tensile2d` (-27%), `2D_MultiScHypEl` (-3.5%), `AhmedML` surface (-6%, error maps in Figure 3) and volume (-7%), and `VKI-LS59`, where the difference (-2.5%) is within one standard deviation.

MeshFiLM-S has the lowest error on `Rotor37`, 59% below Transolver. MeshFiLM-R is 5% behind Transolver (7.15 against 6.82×10^{-3}) and 11% ahead of MGN, while it trains 5.1$\times$ faster per epoch than Transolver++; given the same training time, its error is 28% below that of Transolver (4.88×10^{-3}). MeshFiLM-S also has the lowest mean on `VKI-LS59`. It is 7% worse than Transolver on `2D_MultiScHypEl`, and its error on the `AhmedML` volume is 2.4 times that of Transolver++. Both variants are behind Transolver and Transolver++ on `2D_profile` ($+6\%$ and $+9\%$), yet train 6.4-8.4$\times$ faster per epoch.

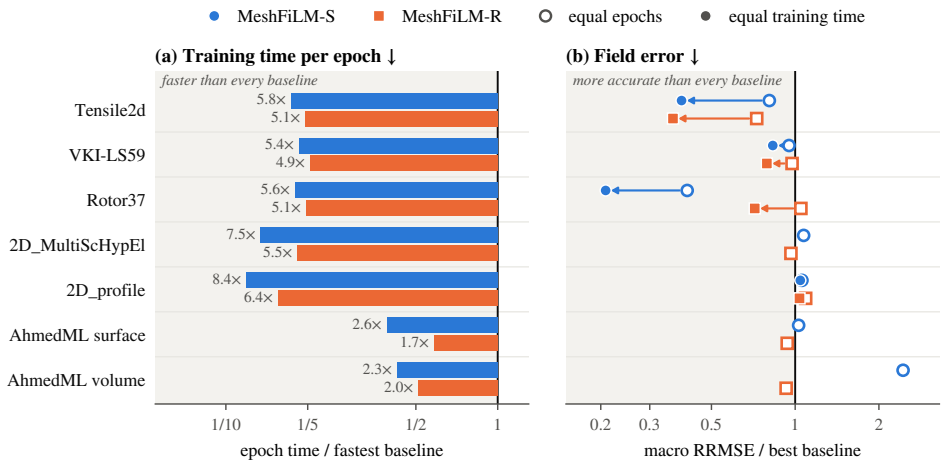

Figure 2: Speed and accuracy of MeshFiLM. Both panels divide a MeshFiLM value by the corresponding baseline value on the same setting, on a log scale; values left of 1 (shaded) are better than every baseline. (a) Seconds per training epoch divided by those of the fastest baseline, Transolver (Transolver++ on `VKI-LS59`), labels give the speed-up. (b) Macro RRMSE divided by that of the most accurate baseline at equal epochs: Transolver++, except Transolver on `Rotor37` and `2D_MultiScHypEl`. Hollow markers: equal number of epochs. Filled markers and arrows: MeshFiLM trained for the wall-clock time of Transolver++ in the four PLAID settings with equal-time runs. Standard deviations over three seeds are in Table 1.

Table 1: Controlled comparison. Macro RRMSE over the output fields of models trained on fields only ($\times 10^3$, lower is better), mean $\pm$ standard deviation over three seeds. *Equal epochs*: every model is trained for the same number of epochs. *Equal time*: on four PLAID settings, MeshFiLM is trained for the wall-clock time of the corresponding Transolver++ equal epoch run. The lower blocks report seconds per epoch and parameter counts. Best in bold and second best underlined, within the equal-epoch block. In the remaining blocks, bold highlights the lowest value within each column. –: not run. Per-field results and parameter counts are in Appendix F.

						AhmedML	
Model	`Tensile2d`	`VKI-LS59`	`Rotor37`	`2D_MultiScHypEl`	`2D_profile`	surface	volume
Equal epochs							
MGN	3.92±0.09	96.90±0.31	8.07±0.08	27.16±0.90	47.60±1.04	–	–
MGN-Tree	3.92±0.26	96.65±0.41	–	26.07±0.43	41.06±1.94	–	–
Transolver	1.96±0.13	24.19±2.21	6.82±0.11	26.02±0.18	25.83±1.15	11.85±0.42	16.76±0.47
Transolver++	1.87±0.03	23.21±1.28	6.88±0.10	26.48±0.19	**25.80±0.52**	10.66±0.19	15.10±0.37
MeshFiLM-S	1.51±0.06	**22.06±0.89**	**2.79±0.12**	27.91±0.08	27.29±0.36	10.97±0.08	36.89±0.97
MeshFiLM-R	**1.36±0.04**	22.62±0.46	7.15±0.17	**25.12±0.03**	28.14±0.22	**9.98±0.43**	**14.01±0.47**
Equal time							
MeshFiLM-S	0.73±0.02	19.28±0.59	**1.42±0.04**	–	26.92±0.10	–	–
MeshFiLM-R	**0.68±0.03**	**18.36±0.32**	4.88±0.07	–	**26.74±0.43**	–	–
Seconds per epoch							
MGN	32.9	107.8	129.1	35.4	67.7	–	–
MGN-Tree	41.4	151.5	–	43.0	86.9	–	–
Transolver	14.4	64.1	77.0	15.5	29.7	50.1	50.8
Transolver++	14.6	64.1	77.7	15.8	29.7	57.9	57.9
MeshFiLM-S	**2.5**	**11.9**	**13.8**	**2.1**	**3.5**	**19.6**	**21.6**
MeshFiLM-R	2.8	13.1	15.2	2.8	4.6	29.2	25.9
Parameters (M)							
MGN	0.921045	0.921522	0.922243	0.922007	0.920164	–	–
MGN-Tree	0.921045	0.921522	–	0.922007	0.920164	–	–
Transolver	0.792149	0.794066	0.796243	0.794967	0.789460	3.05	3.05
Transolver++	0.728281	0.730198	0.732375	0.731099	0.725592	2.79	2.79
MeshFiLM-S	0.645507	0.647058	0.649377	0.648601	0.642670	0.63	0.63
MeshFiLM-R	**0.467253**	**0.469298**	**0.471603**	**0.470711**	**0.436020**	**0.45**	**0.45**

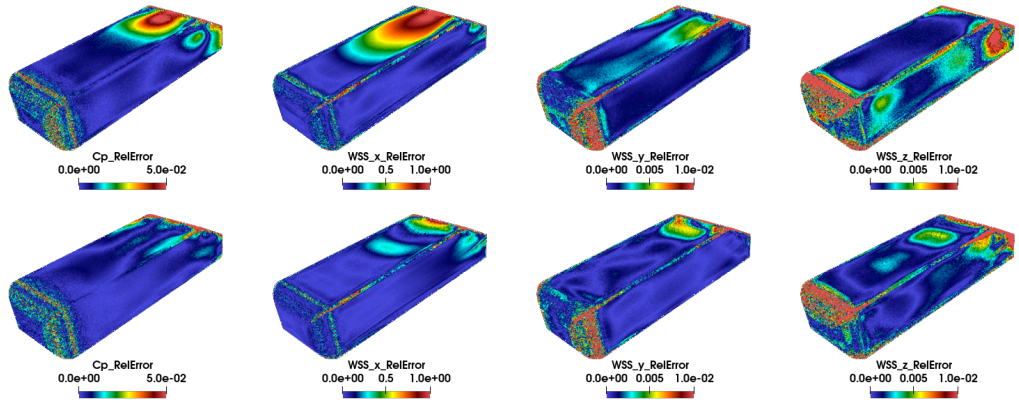

Figure 3: Error of Transolver++ (top) and MeshFiLM-R (bottom) on one `AhmedML` sample, for the pressure coefficient C_p and the three wall-shear stress components.

Equal time: In the time Transolver++ needs for its budget, MeshFiLM runs 4.9-8.4× more epochs. Relative to equal epochs, its error then falls by 50-52% on `Tensile2d` and by 13-49% on `VKI-LS59` and `Rotor37`. On these three datasets, the error is 21-64% below the strongest baseline with MeshFiLM-R and 17-79% below with MeshFiLM-S. On `2D_profile`, the additional epochs reduce the error by only 1.4% (MeshFiLM-S) and 5% (MeshFiLM-R), and both variants remain about 4% behind Transolver++. The tested equal-time budget therefore does not close the gap on this dataset (Section 4).

3.3 Statistics or routed pooling?

The two variants share the decoder and the FiLM head, and differ in the context operator, the encoder and where $\mathbf{s}_{\text{in}}$ enters (Section 2.1). On five of the seven settings their errors differ by at most 11%; MeshFiLM-R is ahead on three of them at equal epochs and on all three that have equal-time runs. The two large gaps go in opposite directions: MeshFiLM-S has 2.6 times the error of MeshFiLM-R on the `AhmedML` volume, and MeshFiLM-R 2.6 times that of MeshFiLM-S on `Rotor37`. To separate the operator from the rest, we keep MeshFiLM-R and replace only its routed context by statistics pooling (Tables 25 and 26).

`Tensile2d`: With fields and scalars trained jointly, this replacement raises the field error by 7% and lowers the scalar error by 22%, to within 6% of MeshFiLM-S on both. The three scalars of `Tensile2d` are maxima; statistics pooling exposes maxima of the node embeddings, though not of the targets, while a region summary is a weighted mean. Giving only the scalar head the statistics does not lower the scalar error.

`Rotor37`: The replacement raises the field error from 4.89 to 6.19×10^{-3}, so statistics pooling does not explain the advantage of MeshFiLM-S on this dataset. The path of the operating point (P, Ω) explains part of it: passing $\mathbf{s}_{\text{in}}$ through the node features $\mathbf{f}_i$, as in MeshFiLM-S, instead of through $\mathbf{p}$, lowers the error of MeshFiLM-R by 34% at equal epochs (one seed), from 7.10 to 4.69×10^{-3} (Table 36), below that of Transolver (6.82×10^{-3}). The `Rotor37` geometries vary little between samples (Appendix K), so the operating point drives much of the variation of the fields. Through $\mathbf{f}_i$, the encoder combines it with the position of every node, and the result enters the decoder input; through $\mathbf{p}$, it reaches the decoder only through the FiLM head, where $\boldsymbol{\gamma}_j$ and $\boldsymbol{\beta}_j$ depend on it affinely through a rank-32 map (Equation 5). With this input path, MeshFiLM-R still has 1.7 times the error of MeshFiLM-S (2.79×10^{-3}).

`AhmedML` volume: The gap lies in the velocity (U_x: 64.3 against 16.7×10^{-3}), while the pressure errors are close (13.0 against 11.0×10^{-3}; Appendix F). Appendix K describes what the routers of MeshFiLM-R learn on four datasets.

3.4 ABLATIONS

Table 2 isolates parts of the conditioning mechanism. Removing the context entirely, so that the decoder becomes a pointwise MLP that sees neither the rest of the mesh nor $\mathbf{s}_{\text{in}}$, multiplies the error by 60 on `Tensile2d`. Given the context, how it enters the decoder still matters. Concatenating it to the decoder input costs 28%, concatenating it to every block costs 7%. Replacing the residual scale $(\mathbf{1}+\boldsymbol{\gamma})$ with zero initialization by a plain $\boldsymbol{\gamma}$ with default initialization increases the `Rotor37` error by 37%. Combining the node embedding and the context by a product instead of a sum increases the error by 32-46%. Replacing the rank-32 sum by a full linear map of the concatenation $[\mathbf{z}_j \| \mathbf{c}_j]$ is 26-43% worse on `Tensile2d` and `Rotor37` and 5% worse on `VKI-LS59`, in single seed runs. The full map contains every rank-32 sum as a special case and has 7-8 times more head parameters, so the gap is not a lack of expressiveness; the low-rank sum appears easier to optimize or to generalize better. With routed pooling, $K = 8$ and $K = 16$ give the same error on the `AhmedML` surface, and $K = 32$ is 9% worse.

Table 2: Ablations of the conditioning mechanism. Change in RRMSE relative to the default model trained with the same budget (negative is better). Raw values in Appendix I.

Component	Default → variant	Dataset, budget, task	Δ RRMSE
Decoder conditioning	FiLM → no context (no $\mathbf{s}_{\text{in}}$)	`Tensile2d`, 2000 epochs, fields	+5905%
	FiLM → $\mathbf{c}_j$ concatenated to the decoder input	`Tensile2d`, 2000 epochs, fields	+28%
	FiLM → $\mathbf{c}_j$ concatenated to every block	`Tensile2d`, 2000 epochs, fields	+7%
FiLM head	$(\mathbf{1}+\boldsymbol{\gamma})$ with zero init → $\boldsymbol{\gamma}$ with default init	`Rotor37`, 3000 epochs, fields+scalars	+37%
	sum → product, $\mathbf{A}\mathbf{z}_j \odot \mathbf{B}\mathbf{c}_j$	`Tensile2d` / `VKI-LS59` / `Rotor37`, 3000 epochs, fields+scalars	+46% / +32% / +40%
	rank-32 sum → full linear map of $[\mathbf{z}_j \,\|\, \mathbf{c}_j]$	same as above	+43% / +5% / +26%
Routed pooling	$K = 8 \to 16\,/\,32$	`AhmedML` surface, fields	+1% / +9%
Sample-level input	$\mathbf{p} \to \mathbf{s}_{\text{in}}$ in $\mathbf{f}_i$	`Rotor37`, 1000 epochs, fields (R)	−34%

Context subsampling: By default, the context of MeshFiLM on `AhmedML` is the chunk being decoded (Section 3.1). Because the region summaries are ratios of sums (Section 2.1), MeshFiLM-R can instead be trained with context and query nodes drawn as independent random subsets, which decouples how many nodes are summarized from how many are decoded. On the `AhmedML` surface, a model trained this way reaches the same error with 32,768 context nodes at inference as with 131,072 (10.78 against 10.77×10^{-3}), and 8,192 context nodes increase the error by 18% (Figure 5 in Appendix L). Compared with the default, it raises inference throughput by 35%, lengthens a training epoch from 26.2 to 45.0 s, and increases the error by 11%. The trade-off is explicit and is set at training time.

Table 3: Public PLAID leaderboards, snapshot of 20 September 2026. `total_error` (lower is better) averages the RRMSE of all scored fields and scalars. We list the entries whose method has a public description, and the best entry for reference. The letter after each MeshFiLM score is its context operator (S: statistics, R: routed); Table 27 lists all settings. These entries are not comparable with Table 1.

Benchmark	MGN	FNO	MMGP	MARIO	MiSe-GNN	MeshFiLM	Best entry
`Tensile2d`	0.0080	0.0123	0.0026	0.0038	–	**0.0015** (S)	0.0007
`VKI-LS59`	0.0497	0.0215	0.0312^{+}	0.0124	–	**0.0114** (R)	0.0093
`Rotor37`	0.0074	0.0313	0.0014	0.0017	–	**0.0011** (S)	0.0004
`2D_MultiScHypEl`	0.0437*	0.0302	–	0.0573	**0.0209**	0.0217 (R)	0.0098
`2D_profile`	0.0593*	0.0972	0.0365	**0.0213**	0.0285	0.0250 (R)	0.0134

$^{+}$ entry named MMGP+ on this leaderboard. * not on the leaderboard; from Casenave et al. (2025).

3.5 PUBLIC PLAID LEADERBOARDS

MeshFiLM places 4th of 13 on `Tensile2d`, 4th of 12 on `VKI-LS59`, 4th of 10 on `Rotor37`, 5th of 11 on `2D_MultiScHypEl` and 7th of 14 on `2D_profile` (Table 3, Appendix H). On the first three, the three entries ranked above it are industrial entries whose training setups are not public. On `Tensile2d`, MeshFiLM matches the field errors of the entries ranked second and third and is behind them on `max_von_mises` and `max_U2_top` (Section 3.3), and on `2D_MultiScHypEl` the graph network MiSe-GNN (Nguyen et al., 2026) is more accurate on the four stress components (Section 4).

4 LIMITATIONS

Transonic flow: On `2D_profile`, MeshFiLM is 6-9% behind Transolver++ at equal epochs and still about 4% behind at equal training time. One hypothesis is that the position of a shock depends on the whole flow field, which models that re-read the node set in every layer can refine, whereas MeshFiLM must deliver it in a single pooling step; a variant with two pooling steps would test it.

Subsampled evaluation and scope: On `AhmedML` every model predicts independent random chunks of each mesh (Section 3.1), so no model reads a full 26.8M-node volume at once; the results compare architectures under this protocol, not their use of the full mesh as context. Subsampling also affects the two operators differently: region summaries are ratios of sums, which a random chunk estimates well, whereas the maximum over a chunk of 0.5% of the volume nodes underestimates the maximum over the mesh and varies between chunks of the same sample.

Local stress concentrations: On `2D_MultiScHypEl`, the graph networks are the most accurate models on the four stress components, which concentrate around the holes: MGN-Tree is 17-27% below MeshFiLM-R on them (Table 12). An edge-free model reads the local geometry only through the node features. The gap is confined to these fields and to edge-based models: MeshFiLM-R is ahead of Transolver and Transolver++ on every stress component, has 2.4 times lower displacement errors than MGN-Tree, the lowest macro RRMSE on this dataset (3.6% below MGN-Tree), and trains 15 times faster per epoch with half the parameters.

Scope: We evaluate interpolation within the geometric and operating ranges of each benchmark, on steady problems; changes of mesh density, extrapolation and time-dependent problems are not tested. The node features are computed from the mesh before training and are the same for every model, so the single exchange refers to the learned forward pass, not to the computation of these features.

5 CONCLUSION

On the benchmarks we study, a surrogate in which nodes exchange information once can match or improve on models that exchange it in every layer, at a fraction of the training cost. MeshFiLM pools the node embeddings once and makes its prediction node-specific in the decoder, with FiLM parameters computed from each node's embedding and its context. Under one pipeline, it trains 1.7-8.4$\times$ faster per epoch than the fastest baseline with fewer parameters; with routed pooling, it is at least as accurate as the most accurate baseline in five of seven settings at equal epochs, and at equal training time its error is 21-64% lower on three of four PLAID benchmarks. On the transonic `2D_profile`, it is 6-9% behind Transolver++ at equal epochs and still about 4% behind at equal training time.

The two pooling operators are within 11% of each other on five of seven settings. Replacing only the routed context of MeshFiLM-R by statistics pooling shows that the operator changes the balance between field and scalar errors on `Tensile2d`. On `Rotor37`, the advantage of MeshFiLM-S comes in part from where the operating point enters: passed through the node features, it lowers the error of MeshFiLM-R by a third, which suggests that sample-level inputs that drive most of the variation of the fields should enter the encoder.

AI USE STATEMENT

LLM was used to polish the writing of this paper and to help find related work. The authors reviewed the suggestions against the underlying methods and results and take responsibility for the final manuscript. Generative AI was not used to produce the results reported in this paper.

ETHICS STATEMENT

This work uses public simulation datasets and involves no human subjects or personal data. Surrogate predictions used in engineering design should be checked against a numerical solver before safety relevant decisions.

REPRODUCIBILITY STATEMENT

Section 2, Table 4 and Algorithm 1 specify both variants; Appendix E gives the training protocol, the configurations of all models and the shared settings (Tables 7 and 8); Appendix C the datasets, splits and outputs; Appendix D the metrics. All datasets are public. We will release the code, the feature construction and the configuration of every run at publication.

REFERENCES

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Benedikt Alkin, Maurits Bleeker, Richard Kurle, Tobias Kronlachner, Reinhard Sonnleitner, Matthias Dorfer, and Johannes Brandstetter. AB-UPT: Scaling neural CFD surrogates for high-fidelity automotive aerodynamics simulations via anchored-branched universal physics transformers. *Transactions on Machine Learning Research (TMLR)*, 2025.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations (ICLR)*, 2021.

Neil Ashton, Danielle C. Maddix, Samuel Gundry, and Parisa M. Shabestari. AhmedML: High-fidelity computational fluid dynamics dataset for incompressible, low-speed bluff body aerodynamics. *arXiv preprint arXiv:2407.20801*, 2024.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Marc Brockschmidt. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, 2020.

Yadi Cao, Menglei Chai, Minchen Li, and Chenfanfu Jiang. Efficient learning of mesh-based physical simulation with bi-stride multi-scale graph neural network. In *International Conference on Machine Learning (ICML)*, 2023.

Fabien Casenave, Brian Staber, and Xavier Roynard. MMGP: a mesh morphing gaussian process-based machine learning method for regression of physical problems under nonparametrized geometrical variability. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Fabien Casenave, Xavier Roynard, Brian Staber, Alexandre Devaux-Rivière, William Piat, Michele Alessandro Bucci, Nissrine Akkari, Abbas Kabalan, Xuan Minh Vuong Nguyen, Luca Saverio, Raphaël Carpintero Perez, Anthony Kalaydjian, Samy Fouché, Thierry Gonon, Ghassan Najjar, Thomas Daniel, Emmanuel Menier, Matthieu Nastorg, Giovanni Catalani, and Christian Rey. PLAID: A unified data model for machine learning on heterogeneous physics simulations. *arXiv preprint arXiv:2505.02974*, 2025.

Giovanni Catalani, Michael Bauerheim, Frédéric Tost, Xavier Bertrand, and Joseph Morlier. Geometry aware inference of steady state PDEs using equivariant neural fields representations. *arXiv preprint arXiv:2504.18591*, 2025.

Giovanni Catalani, Jean Fesquet, Xavier Bertrand, Frédéric Tost, Michael Bauerheim, and Joseph Morlier. Towards scalable surrogate models based on neural fields for large scale aerodynamic simulations. *Computers & Fluids*, 306:106929, 2026. doi: 10.1016/j.compfluid.2025.106929.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Wojciech M. Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Świrszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning (ICML)*, 2022.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. MultiScale MeshGraphNets. *arXiv preprint arXiv:2210.00612*, 2022. ICML 2022 Workshop on AI for Science.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Ali Kashefi, Davis Rempe, and Leonidas J. Guibas. A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *Physics of Fluids*, 33(2):027104, 2021.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research*, 24, 2023a.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Animashree Anandkumar. Geometry-informed neural operator for large-scale 3D PDEs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218–229, 2021.

Huakun Luo, Haixu Wu, Hang Zhou, Lanxiang Xing, Yichen Di, Jianmin Wang, and Mingsheng Long. Transolver++: An accurate neural solver for PDEs on million-scale geometries. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 41432–41449, 2025.

Xuan Minh Vuong Nguyen, Nissrine Akkari, Fabien Casenave, Jonathan Viquerat, and Elie Hachem. Error-driven graph augmentation for mesh-based PDE surrogates. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Jangseop Park and Namwoo Kang. Point-DeepONet: Predicting nonlinear fields on non-parametric geometries under variable load conditions. *Neural Networks*, 198:108560, June 2026. ISSN 0893-6080. doi: 10.1016/j.neunet.2026.108560. URL `http://dx.doi.org/10.1016/j.neunet.2026.108560`.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019a.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019b.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations (ICLR)*, 2021.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Winfried Ripken, Lisa Coiffard, Felix Pieper, and Sebastian Dziadzio. Multiscale neural operators for solving time-independent PDEs. In *NeurIPS 2023 Workshop on the Symbiosis of Deep Learning and Differential Equations (DLDE)*, 2023. arXiv:2311.05964.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning (ICML)*, 2020.

Louis Serrano, Lise Le Boudec, Armand Kassaï Koupaï, Thomas X. Wang, Yuan Yin, Jean-Noël Vittaut, and Patrick Gallinari. Operator learning with neural fields: Tackling PDEs on general geometries. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Nicholas Sung, Steven Spreizer, Mohamed Elrefaie, Kaira Samuel, Matthew C. Jones, and Faez Ahmed. BlendedNet: A blended wing body aircraft dataset and surrogate model for aerodynamic predictions. In *Volume 3B: 51st Design Automation Conference (DAC)*, IDETC-CIE2025. American Society of Mechanical Engineers, August 2025. doi: 10.1115/detc2025-168977. URL http://dx.doi.org/10.1115/DETC2025-168977.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse PDE problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *International Conference on Machine Learning (ICML)*, 2024.

Yuan Yin, Matthieu Kirchmeyer, Jean-Yves Franceschi, Alain Rakotomamonjy, and Patrick Gallinari. Continuous PDE dynamics forecasting with implicit neural representations. In *International Conference on Learning Representations (ICLR)*, 2023.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

A ARCHITECTURE DETAILS

Table 4: Configurations of the two variants in the controlled runs. Parameter counts are for the fields-only models.

	MeshFiLM-S	MeshFiLM-R
Context operator	Statistics pooling (Eq. 1)	Routed pooling (Eqs. 2,3)
Encoder	Layer norm of $\mathbf{a}_i$, then 2 layers (Linear, SiLU, LN)	3 layers (Linear, LN, SiLU)
G / Value map $\mathbf{V}$	G: $3H \to H \to H$	$\mathbf{V}$: $H \to C$, $C = 32$
Regions K, τ_c, τ_q	–	$K = 8$, $\tau_c = \tau_q = 1$
Sample-level inputs $\mathbf{s}_{in}$	repeated into $\mathbf{f}_i$ at every node	encoded once, $\mathbf{p} = P(\mathbf{s}_{in}) \in \mathbb{R}^{128}$, P a 2-layer MLP
Decoder: blocks L, width H	$L = 3$, $H = 256$	$L = 3$, $H = 256$
FiLM head (Eq. 5)	rank $m = 32$; $\mathbf{B}$: $H \to m$	rank $m = 32$; $\mathbf{B}$: $(C+128) \to m$
Scalar head S	input $\mathbf{g}$ (H), 1 hidden layer of width $H/2$	input $[\mathbf{t}_1 \| \cdots \| \mathbf{t}_K \| \mathbf{p}]$, 2 hidden layers of width H
Loss weight λ_s	0.014	0.014
Parameters: `Tensile2d`, `VKI-LS59`, `Rotor37`	645,507; 647,058; 649,377	467,253; 469,298; 471,603
Parameters: `2D_MultiScHypEl`, `2D_profile`	648,601; 642,670	470,711; 436,020
Parameters: `AhmedML`	0.63M	0.45M

Algorithm 1 Forward pass of MeshFiLM-R for one sample

Require: Node inputs $\{\mathbf{a}_i\}_{i=1}^N$; sample parameters $\mathbf{s}$; context nodes $\mathcal{C} \subseteq \{1, \dots, N\}$ (all nodes unless subsampled)

1: $\mathbf{z}_i \leftarrow E(\mathbf{a}_i)$ for all i; $\quad \mathbf{p} \leftarrow P(\mathbf{s})$
2: $r^c_{ik} \leftarrow \mathrm{softmax}_k(\mathbf{w}_k^{c\top} \mathbf{z}_i / \tau_c)$ for $i \in \mathcal{C}$
3: $\mathbf{t}_k \leftarrow \sum_{i \in \mathcal{C}} r^c_{ik} \mathbf{V}\mathbf{z}_i \,/\, \sum_{i \in \mathcal{C}} r^c_{ik}$ for $k = 1, \dots, K$ ▷ the only cross-node step
4: **for** each query node j, in chunks **do**
5: $\quad \mathbf{c}_j \leftarrow \big[\sum_k \mathrm{softmax}_k(\mathbf{w}_k^{q\top} \mathbf{z}_j / \tau_q)\, \mathbf{t}_k \,\|\, \mathbf{p}\big]$; $\quad \mathbf{h} \leftarrow [\mathbf{x}_j \,\|\, \mathbf{z}_j]$
6: $\quad$ **for** $\ell = 0, \dots, L-1$ **do**
7: $\quad\quad \tilde{\mathbf{h}} \leftarrow \mathrm{LN}_0(\mathbf{W}_\ell \mathbf{h} + \mathbf{b}_\ell)$; $\quad [\boldsymbol{\gamma} \,\|\, \boldsymbol{\beta}] \leftarrow \mathbf{U}_\ell(\mathbf{A}_\ell \mathbf{z}_j + \mathbf{B}_\ell \mathbf{c}_j) + \mathbf{d}_\ell$
8: $\quad\quad \mathbf{h} \leftarrow \phi\big((\mathbf{1} + \boldsymbol{\gamma}) \odot \tilde{\mathbf{h}} + \boldsymbol{\beta}\big)$
9: $\quad$ **end for**
10: $\quad \hat{\mathbf{y}}_j \leftarrow \mathbf{W}_{\mathrm{out}} \mathbf{h} + \mathbf{b}_{\mathrm{out}}$
11: **end for**
12: $\hat{\mathbf{s}} \leftarrow S([\mathbf{t}_1 \| \cdots \| \mathbf{t}_K \,\|\, \mathbf{p}])$

The two variants share the decoder ($L = 3$, $H = 256$) and the rank-32 FiLM head. They differ in the encoder, the context operator and the path of the sample-level inputs $\mathbf{s}_{\text{in}}$ (Section 2.1). MeshFiLM-S appends $\mathbf{s}_{\text{in}}$ to $\mathbf{f}_i$, so every node embedding, and therefore every statistic, depends on it. MeshFiLM-R encodes $\mathbf{s}_{\text{in}}$ once into $\mathbf{p}$ and appends $\mathbf{p}$ to the context and to the input of the scalar head; its encoder, routers and region summaries see only the node inputs. MeshFiLM-R has 27-32% fewer parameters than MeshFiLM-S on PLAID and 29% fewer on `AhmedML`. Line 3 is the only step that reads several nodes. For a fixed context set $\mathcal{C}$, the prediction of a node therefore does not depend on how the query nodes are split into chunks.

B COST OF THE EXCHANGE BETWEEN NODES

Table 5 counts the operations that mix information across nodes. N is the number of nodes, E the number of mesh edges, M the number of Transolver slices, K the number of MeshFiLM regions,

Table 5: Operations that exchange information between nodes, per forward pass.

Model	Cross-node operation	Times	Cost each
MGN	message passing along mesh edges	L	$\mathcal{O}(EH^2)$
Transolver, Transolver++	slicing, attention among slices, de-slicing	L	$\mathcal{O}(NMH + M^2H)$
MeshFiLM-S	mean, maximum, standard deviation over nodes	1	$\mathcal{O}(NH)$
MeshFiLM-R	routing to K regions and reading their summaries	1	$\mathcal{O}(NKH)$

H the width and L the number of layers. All other operations of the three model families act on one node (or one edge) at a time and cost $\mathcal{O}(NH^2)$ per layer. For the configurations of Table 4, the cross-node step of either variant costs less per node than one decoder block, so the forward cost of MeshFiLM is dominated by node-wise MLPs, which can be batched over nodes and chunks. The measured times per epoch are in Table 1.

C DATASETS

Table 6: PLAID benchmarks use the official splits and the outputs scored by their leaderboards. $\bar{N}$: mean number of nodes per sample.

Dataset	Physics	Train/Test	$\bar{N}$	Inputs	Outputs
`Tensile2d`	2D solid, nonlinear	500/200	9,428	6 scalars	5 fields, 3 scalars
`2D_MultiScHypEl`	2D solid, hyperelastic	764/376	5,692	3 scalars	7 fields, 1 scalar
`VKI-LS59`	2D RANS, blade	671/168	36,421	2 scalars	2 fields, 6 scalars
`Rotor37`	3D RANS, compressor	1000/200	29,773	2 scalars	3 fields, 3 scalars
`2D_profile`	2D RANS, transonic	300/100	37,042	–	4 fields
`AhmedML` surface	3D incompressible	337/125 (37 Val)	1,099,737	8 scalars	4 fields
`AhmedML` volume	3D incompressible	337/125 (37 Val)	26,825,476	8 scalars	4 fields

PLAID benchmarks: We use the benchmark versions of the PLAID datasets (Casenave et al., 2025) and score only the outputs used by the leaderboards.

- `Tensile2d`: 2D quasi-static solid mechanics with small deformations and a nonlinear constitutive law. Inputs: a pressure and five material parameters. Fields: $U1$, $U2$, $sig11$, $sig22$, $sig12$. Scalars: `max_von_mises`, `max_U2_top`, `max_sig22_top`.
- `2D_MultiScHypEl`: 2D quasi-static finite-strain hyperelasticity of porous microstructures, with a varying number of circular holes. Inputs: macroscopic deformation components $C11$, $C12$, $C22$. Fields: $u1$, $u2$, $P11$, $P12$, $P22$, $P21$, psi. Scalar: `effective_energy`.
- `VKI-LS59`: 2D compressible steady RANS flow around a blade geometry. Inputs: `angle_in`, `mach_out`. Fields: nut, $mach$ number. Scalars: `Q`, `power`, `Pr`, `Tr`, `eth_is`, `angle_out`.
- `Rotor37`: 3D compressible steady RANS flow around a compressor blade. Inputs: pressure P and rotation speed Ω. Fields: $Density$, $Pressure$, $Temperature$. Scalars: `Massflow`, `Compression ratio`, `Efficiency`.
- `2D_profile`: 2D compressible steady RANS flow around airfoil-like profiles in the transonic regime. No input scalars. Fields: $Mach$, $Pressure$, V_x, V_y.

The per-node features $\mathbf{f}_i$ are produced from the mesh by dataset-specific geometric encoders, boundary indicators, distances and anchor-based coordinates. Within the controlled protocol they are identical for every model, and the construction is released with the code. All samples of `Rotor37` have the same number of nodes (29,773), as do all samples of `VKI-LS59` (36,421), the geometry varies through the node coordinates.

AhmedML: 500 geometric variants of the Ahmed body with time averaged flow fields (Ashton et al., 2024). Surface fields: pressure coefficient C_p and the three components of the wall shear stress. Volume fields: the three velocity components and pressure.

D METRICS

For a test set of n samples, where sample i has N^i nodes, PLAID defines the RRMSE of a field f and of a scalar s as:

$$\mathrm{RRMSE}_f = \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\frac{1}{N^i} \left\| \mathbf{f}^i_{\mathrm{ref}} - \mathbf{f}^i_{\mathrm{pred}} \right\|_2^2}{\left\| \mathbf{f}^i_{\mathrm{ref}} \right\|_\infty^2} \right)^{1/2}, \qquad \mathrm{RRMSE}_s = \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\left(s^i_{\mathrm{ref}} - s^i_{\mathrm{pred}} \right)^2}{\left(s^i_{\mathrm{ref}} \right)^2} \right)^{1/2} \tag{7}$$

RRMSE normalizes the error of each sample by the largest magnitude of the reference field in that sample, so each sample weighs the same whatever its scale. The ranking of two models can therefore differ between RRMSE and the unnormalized RMSE, as between MeshFiLM-R and Transolver++ on the `AhmedML` volume (Tables 20 and 22). On `2D_profile`, which has no scalars, our macro RRMSE is the leaderboard `total_error`. For RMSE and MAE on `AhmedML`, the macro column averages fields of different magnitude and, on the surface, is dominated by C_p.

The leaderboard score `total_error` is the mean of the RRMSE over all scored fields and scalars. Our macro RRMSE is the mean over the scored fields only. On `AhmedML` we use the same field RRMSE and also report the relative L_2 error, RMSE and MAE.

E TRAINING AND EVALUATION PROTOCOL

Budgets and timing: In the controlled protocol, every model is trained on each dataset for the number of epochs E of Table 8 with the same optimizer, learning rate schedule, batch size and precision, on one GPU of the same type. The equal time runs of MeshFiLM use the wall-clock time that Transolver++ needs for E epochs on the same dataset. Seconds per epoch are measured on the training loop only. One A30 GPU used for `AhmedML`, and one A100 GPU used for PLAID datasets.

Loss and evaluation: All models minimize the mean squared error on standardized fields, with no auxiliary term. We keep the checkpoint with the lowest validation error and report its error on the test split of Table 6.

Hyperparameters and baselines: Table 7 lists the configurations. They were selected on the validation split.

Table 7: Model configurations in the controlled protocol. MeshFiLM: Table 4.

Model	Setting	Layers	Width	Parameters	Other
MGN	PLAID	15MP	80	0.92M	MP=message passing steps
MGN-Tree	PLAID	15MP	80	0.92M	tree levels=10
Transolver	PLAID	4 blocks	128	0.79M	heads=4, slices=16, MLP ratio=4
Transolver	AhmedML	4 blocks	256	3.05M	heads=8, slices=32, MLP ratio=4
Transolver++	PLAID	4 blocks	128	0.73M	heads=4, slices=16, MLP ratio=4
Transolver++	AhmedML	4 blocks	256	2.79M	heads=8, slices=32, MLP ratio=4
MeshFiLM-S / -R	all	3 blocks	256	0.64-0.65M / 0.44-0.47M	Table 4

Subsampling and timing on AhmedML: Training uses random subsets of 131,072 nodes per mesh, redrawn at every iteration, identically for all models. At test time all nodes of a mesh are shuffled and split into chunks of 131,072 nodes, each chunk is predicted independently and, for MeshFiLM, serves as its own context unless stated otherwise. For every model in Tables 14 and 19, throughput multiplied by forward time equals 99.5-99.9% of the $125 \times \bar{N}$ test nodes, so the errors

Table 8: Training settings shared by all models in the controlled protocol.

Optimizer	AdamW, learning rate 0.001, weight decay 0.0001
Schedule	Cosine Annealing
Epochs E (equal epochs)	`Tensile2d` 1000, `VKI-LS59` 1000, `Rotor37` 1000, `2D_MultiScHypEl` 1000, `2D_profile` 1000, `AhmedML` 1000
Precision / hardware	fp32 / One A100 (PLAID), One A30 (`AhmedML`)
Seeds	42, 43, 44

cover the full test meshes. Forward times are summed over the whole 125-sample test set, not per sample. Per sample this is 0.21 s (MeshFiLM-S) to 0.46 s (Transolver++) on the surface and 4.5 s to 10.7 s on the volume.

F Full controlled results

All tables in this section follow the controlled protocol of Section 3.1: RRMSE (lower is better), mean $\pm$ standard deviation over three seeds (42, 43, 44).

Table 9: `Tensile2d`, per-field RRMSE.

Model	U1	U2	sig11	sig22	sig12	Macro RRMSE	Epoch Time (s)	#Params
MGN	0.00348 ± 0.00017	0.00498 ± 0.00007	0.00567 ± 0.00023	0.00247 ± 0.00010	0.00299 ± 0.00009	0.00392 ± 0.00009	32.87	921045
MGN-Tree	0.00342 ± 0.00027	0.00486 ± 0.00033	0.00568 ± 0.00039	0.00254 ± 0.00014	0.00311 ± 0.00023	0.00392 ± 0.00026	41.43	921045
Transolver	0.00159 ± 0.00007	0.00219 ± 0.00010	0.00291 ± 0.00029	0.00128 ± 0.00009	0.00183 ± 0.00020	0.00196 ± 0.00013	14.39	792149
Transolver++	0.00146 ± 0.00006	0.00202 ± 0.00006	0.00298 ± 0.00009	0.00129 ± 0.00005	0.00162 ± 0.00003	0.00187 ± 0.00003	14.55	728281
MeshFiLM-S (equal epochs)	0.00117 ± 0.00007	0.00169 ± 0.00007	0.00223 ± 0.00015	0.00110 ± 0.00010	0.00134 ± 0.00007	0.00151 ± 0.00006	**2.50**	645507
MeshFiLM-S (equal time)	0.00039 ± 0.00003	0.00053 ± 0.00003	0.00128 ± 0.00003	0.00060 ± 0.00001	$\mathbf{0.00088 \pm 0.00002}$	0.00073 ± 0.00002	**2.50**	645507
MeshFiLM-R (equal epochs)	0.00111 ± 0.00011	0.00146 ± 0.00002	0.00194 ± 0.00011	0.00100 ± 0.00004	0.00128 ± 0.00006	0.00136 ± 0.00004	2.82	**467253**
MeshFiLM-R (equal time)	$\mathbf{0.00030 \pm 0.00003}$	$\mathbf{0.00043 \pm 0.00003}$	$\mathbf{0.00121 \pm 0.00004}$	$\mathbf{0.00058 \pm 0.00002}$	0.00088 ± 0.00004	$\mathbf{0.00068 \pm 0.00003}$	2.82	**467253**

At equal epochs, both variants have a lower error than every baseline on all five fields, and MeshFiLM-R is the most accurate on each of them. The largest difference is on $sig11$, 33% below Transolver (1.94 against 2.91×10^{-3}). With the equal-time budget, the displacement errors fall by a factor of 3.0-3.7 and the stress errors by 1.5-1.8.

Table 10: `VKI-LS59`, per-field RRMSE.

Model	mach	nut	Macro RRMSE	Epoch Time (s)	#Params
MGN	0.04208 ± 0.00088	0.15173 ± 0.00088	0.09690 ± 0.00031	107.83	921522
MGN-Tree	0.04238 ± 0.00033	0.15091 ± 0.00114	0.09665 ± 0.00041	151.53	921522
Transolver	0.01391 ± 0.00128	0.03447 ± 0.00314	0.02419 ± 0.00221	64.10	794066
Transolver++	0.01266 ± 0.00034	0.03376 ± 0.00222	0.02321 ± 0.00128	64.06	730198
MeshFiLM-S (equal epochs)	0.01191 ± 0.00048	0.03220 ± 0.00135	0.02206 ± 0.00089	**11.92**	647058
MeshFiLM-S (equal time)	$\mathbf{0.01000 \pm 0.00035}$	0.02857 ± 0.00085	0.01928 ± 0.00059	**11.92**	647058
MeshFiLM-R (equal epochs)	0.01278 ± 0.00031	0.03247 ± 0.00061	0.02262 ± 0.00046	13.09	**469298**
MeshFiLM-R (equal time)	0.01026 ± 0.00024	$\mathbf{0.02646 \pm 0.00041}$	$\mathbf{0.01836 \pm 0.00032}$	13.09	**469298**

At equal epochs, MeshFiLM-S has the lowest error on both fields. MeshFiLM-R is 1% behind Transolver++ on the $mach$ number and 4% ahead on nut. The spread over seeds of the baselines is large on this dataset, up to 3.1×10^{-3} on nut for Transolver. With the equal-time budget, MeshFiLM-S has the lowest $mach$ error and MeshFiLM-R the lowest nut error.

Table 11: `Rotor37`, per-field RRMSE.

Model	Density	Pressure	Temperature	Macro RRMSE	Epoch Time (s)	#Params
MGN	0.01114 ± 0.00010	0.01100 ± 0.00010	0.00208 ± 0.00004	0.00807 ± 0.00008	129.06	922243
Transolver	0.00932 ± 0.00014	0.00917 ± 0.00014	0.00197 ± 0.00005	0.00682 ± 0.00011	77.02	796243
Transolver++	0.00941 ± 0.00013	0.00925 ± 0.00014	0.00199 ± 0.00004	0.00688 ± 0.00010	77.74	732375
MeshFiLM-S (equal epochs)	0.00375 ± 0.00017	0.00366 ± 0.00018	0.00095 ± 0.00004	0.00279 ± 0.00012	**13.83**	649377
MeshFiLM-S (equal time)	$\mathbf{0.00187 \pm 0.00005}$	$\mathbf{0.00183 \pm 0.00005}$	$\mathbf{0.00056 \pm 0.00001}$	$\mathbf{0.00142 \pm 0.00004}$	**13.83**	649377
MeshFiLM-R (equal epochs)	0.00980 ± 0.00026	0.00966 ± 0.00026	0.00201 ± 0.00001	0.00715 ± 0.00017	15.19	**471603**
MeshFiLM-R (equal time)	0.00668 ± 0.00009	0.00658 ± 0.00007	0.00140 ± 0.00005	0.00488 ± 0.00007	15.19	**471603**

MeshFiLM-S has the lowest error on all three fields, 60% below Transolver on $Density$ and $Pressure$ and 52% on $Temperature$. MeshFiLM-R is within 2-5% of Transolver on every field

and has 2.1-2.6 times the error of MeshFiLM-S on each. Section 3.3 shows that this gap does not come from the operator (Table 26), and that passing the operating point through the node features lowers the error of MeshFiLM-R by 34% (Table 36).

Table 12: `2D_MultiScHypEl`, per-field RRMSE.

Model	u1	u2	P11	P12	P22	P21	psi	Macro RRMSE	Epoch Time (s)	#Params
MGN	0.02307 ± 0.00006	0.02333 ± 0.00041	0.02084 ± 0.00128	0.03341 ± 0.00169	0.02104 ± 0.00125	0.03339 ± 0.00158	0.03505 ± 0.00008	0.02716 ± 0.00090	35.44	922007
MGN-Tree	0.02294 ± 0.00050	0.02296 ± 0.00014	$\mathbf{0.01925 \pm 0.00053}$	$\mathbf{0.03155 \pm 0.00083}$	$\mathbf{0.01938 \pm 0.00055}$	$\mathbf{0.03149 \pm 0.00079}$	0.03496 ± 0.00014	0.02607 ± 0.00043	42.98	922007
Transolver	0.01178 ± 0.00045	0.01221 ± 0.00081	0.02668 ± 0.00035	0.03867 ± 0.00025	0.02687 ± 0.00031	0.03829 ± 0.00051	$\mathbf{0.02764 \pm 0.00034}$	0.02602 ± 0.00018	15.50	794967
Transolver++	0.01164 ± 0.00046	0.01229 ± 0.00053	0.02720 ± 0.00027	0.03950 ± 0.00031	0.02734 ± 0.00020	0.03934 ± 0.00030	0.02806 ± 0.00014	0.02648 ± 0.00019	15.80	731099
MeshFiLM-S (equal epochs)	0.01403 ± 0.00042	0.01448 ± 0.00034	0.02783 ± 0.00017	0.04107 ± 0.00037	0.02814 ± 0.00025	0.04092 ± 0.00020	0.02888 ± 0.00022	0.02791 ± 0.00008	**2.08**	648601
MeshFiLM-R (equal epochs)	$\mathbf{0.00955 \pm 0.00044}$	$\mathbf{0.00973 \pm 0.00017}$	0.02622 ± 0.00015	0.03812 ± 0.00004	0.02648 ± 0.00009	0.03791 ± 0.00003	0.02781 ± 0.00004	$\mathbf{0.02512 \pm 0.00003}$	2.84	**470711**

The macro error hides two groups of fields. On the displacements, MeshFiLM-R has 2.4 times lower error than MGN-Tree, and Transolver, Transolver++ and MeshFiLM-S 1.6-2.0 times lower. On the four stress components the order reverses: MGN-Tree has the lowest error, 17-27% below MeshFiLM-R, and MGN is second. The stresses concentrate around the holes, where message passing along mesh edges reads the local geometry directly. Transolver and MeshFiLM-R are within 1% on psi.

Table 13: `2D_profile`, per-field RRMSE.

Model	Mach	Pressure	Velocity-x	Velocity-y	Macro RRMSE	Epoch Time (s)	#Params
MGN	0.05193 ± 0.00017	0.03618 ± 0.00154	0.05980 ± 0.00019	0.04247 ± 0.00265	0.04760 ± 0.00104	67.69	920164
MGN-Tree	0.04527 ± 0.00302	0.02938 ± 0.00075	0.05381 ± 0.00349	0.03579 ± 0.00297	0.04106 ± 0.00194	86.94	920164
Transolver	0.02945 ± 0.00154	$\mathbf{0.01750 \pm 0.00068}$	0.03612 ± 0.00165	0.02025 ± 0.00106	0.02583 ± 0.00115	29.65	789460
Transolver++	0.02910 ± 0.00118	0.01754 ± 0.00027	0.03604 ± 0.00106	0.02053 ± 0.00029	$\mathbf{0.02580 \pm 0.00052}$	29.66	725592
MeshFiLM-S (equal epochs)	0.02928 ± 0.00056	0.02023 ± 0.00011	0.03536 ± 0.00072	0.02428 ± 0.00014	0.02729 ± 0.00036	**3.53**	642670
MeshFiLM-S (equal time)	$\mathbf{0.02895 \pm 0.00031}$	0.01992 ± 0.00029	$\mathbf{0.03517 \pm 0.00034}$	0.02365 ± 0.00049	0.02692 ± 0.00010	**3.53**	642670
MeshFiLM-R (equal epochs)	0.03197 ± 0.00045	0.01900 ± 0.00024	0.04005 ± 0.00085	0.02152 ± 0.00030	0.02814 ± 0.00022	4.62	**436020**
MeshFiLM-R (equal time)	0.03084 ± 0.00038	0.01803 ± 0.00022	0.03816 ± 0.00043	$\mathbf{0.01992 \pm 0.00085}$	0.02674 ± 0.00043	4.62	**436020**

No model is the most accurate on every field. At equal epochs, Transolver++ is best on the $Mach$ number, Transolver on the $Pressure$ and on V_y, and MeshFiLM-S on V_x. The two variants have complementary errors: MeshFiLM-S is more accurate on the $Mach$ number and V_x, MeshFiLM-R on the $Pressure$ and V_y. With the equal-time budget, MeshFiLM-S becomes the most accurate on the $Mach$ number and MeshFiLM-R on V_y, but the $Pressure$ error of MeshFiLM-R stays 3% above Transolver.

Table 14: `AhmedML` surface. Forward time is summed over the 125-sample test set.

Model	Error Metrics				Efficiency			
	RRMSE	L2	RMSE	MAE	Throughput (#nodes/s)	Latency (s)	#Params (M)	Time/Ep (s)
Transolver	0.01185 ± 0.00042	0.06190 ± 0.00226	0.00572 ± 0.00031	0.00278 ± 0.00022	3,007,847	45.47	3.05	50.08
Transolver++	0.01066 ± 0.00019	0.05585 ± 0.00031	0.00491 ± 0.00005	0.00214 ± 0.00003	2,389,651	57.23	2.79	57.91
MeshFiLM-S	0.01097 ± 0.00008	0.05848 ± 0.00057	0.00508 ± 0.00004	0.00223 ± 0.00002	**5,220,931**	**26.27**	0.63	**19.56**
MeshFiLM-R	$\mathbf{0.00998 \pm 0.00043}$	$\mathbf{0.05574 \pm 0.00112}$	$\mathbf{0.00471 \pm 0.00010}$	$\mathbf{0.00200 \pm 0.00005}$	4,534,827	30.15	**0.45**	29.23

Table 15: `AhmedML` surface RRMSE.

Model	Cp	WSS_x	WSS_y	WSS_z	Macro
Transolver	0.01771 ± 0.00072	$\mathbf{0.00913 \pm 0.00012}$	0.00840 ± 0.00048	0.01217 ± 0.00039	0.01185 ± 0.00042
Transolver++	0.01517 ± 0.00077	0.00918 ± 0.00008	0.00732 ± 0.00008	0.01095 ± 0.00010	0.01066 ± 0.00019
MeshFiLM-S	0.01512 ± 0.00025	0.00978 ± 0.00020	0.00781 ± 0.00007	0.01115 ± 0.00015	0.01097 ± 0.00008
MeshFiLM-R	$\mathbf{0.01287 \pm 0.00114}$	0.00920 ± 0.00018	$\mathbf{0.00720 \pm 0.00022}$	$\mathbf{0.01064 \pm 0.00037}$	$\mathbf{0.00998 \pm 0.00043}$

Table 16: `AhmedML` surface Relative L2.

Model	Cp	WSS_x	WSS_y	WSS_z	Macro
Transolver	0.05324 ± 0.00290	0.05783 ± 0.00081	0.06756 ± 0.00278	0.06899 ± 0.00255	0.06190 ± 0.00226
Transolver++	0.04566 ± 0.00045	$\mathbf{0.05736 \pm 0.00042}$	$\mathbf{0.05932 \pm 0.00018}$	$\mathbf{0.06108 \pm 0.00056}$	0.05585 ± 0.00031
MeshFiLM-S	0.04712 ± 0.00037	0.06030 ± 0.00122	0.06281 ± 0.00066	0.06371 ± 0.00057	0.05848 ± 0.00057
MeshFiLM-R	$\mathbf{0.04380 \pm 0.00104}$	0.05774 ± 0.00096	0.05954 ± 0.00138	0.06187 ± 0.00165	$\mathbf{0.05574 \pm 0.00112}$

Table 17: AhmedML surface RMSE.

Model	Cp	WSS_x	WSS_y	WSS_z	Macro
Transolver	0.02267 ± 0.00124	$\mathbf{0.00011 \pm 0.00000}$	$\mathbf{0.00004 \pm 0.00000}$	0.00005 ± 0.00000	0.00572 ± 0.00031
Transolver++	0.01946 ± 0.00020	$\mathbf{0.00011 \pm 0.00000}$	$\mathbf{0.00004 \pm 0.00000}$	$\mathbf{0.00004 \pm 0.00000}$	0.00491 ± 0.00005
MeshFiLM-S	0.02010 ± 0.00018	0.00012 ± 0.00000	$\mathbf{0.00004 \pm 0.00000}$	0.00005 ± 0.00000	0.00508 ± 0.00004
MeshFiLM-R	$\mathbf{0.01865 \pm 0.00039}$	$\mathbf{0.00011 \pm 0.00000}$	$\mathbf{0.00004 \pm 0.00000}$	$\mathbf{0.00004 \pm 0.00000}$	$\mathbf{0.00471 \pm 0.00010}$

Table 18: AhmedML surface MAE.

Model	Cp	WSS_x	WSS_y	WSS_z	Macro
Transolver	0.01105 ± 0.00088	$\mathbf{0.00005 \pm 0.00000}$	0.00002 ± 0.00000	0.00002 ± 0.00000	0.00278 ± 0.00022
Transolver++	0.00850 ± 0.00014	$\mathbf{0.00005 \pm 0.00000}$	$\mathbf{0.00001 \pm 0.00000}$	$\mathbf{0.00001 \pm 0.00000}$	0.00214 ± 0.00003
MeshFiLM-S	0.00884 ± 0.00007	$\mathbf{0.00005 \pm 0.00000}$	$\mathbf{0.00001 \pm 0.00000}$	0.00002 ± 0.00000	0.00223 ± 0.00002
MeshFiLM-R	$\mathbf{0.00792 \pm 0.00019}$	0.00005 ± 0.00001	$\mathbf{0.00001 \pm 0.00000}$	$\mathbf{0.00001 \pm 0.00001}$	$\mathbf{0.00200 \pm 0.00005}$

MeshFiLM-R has the lowest RRMSE on C_p, WSS_y and WSS_z; on WSS_x, Transolver, Transolver++ and MeshFiLM-R are within 1%. The largest difference is on C_p, 15% below Transolver++. MeshFiLM-S processes 5.2M nodes per second at inference, 1.7 times Transolver and 2.2 times Transolver++, and MeshFiLM-R 4.5M. The absolute wall shear stress errors (Tables 17 and 18) are below the precision of the tables; the relative metrics compare them.

Table 19: AhmedML volume. Forward time is summed over the 125-sample test set.

Model	Error Metrics				Efficiency			
	RRMSE	L2	RMSE	MAE	Throughput (#nodes/s)	Latency (s)	#Params (M)	Time/Ep (s)
Transolver	0.01676 ± 0.00047	0.06640 ± 0.00315	0.01339 ± 0.00039	0.00564 ± 0.00026	3,169,802	1056.20	3.05	50.78
Transolver++	0.01510 ± 0.00037	0.05914 ± 0.00160	$\mathbf{0.01182 \pm 0.00042}$	0.00492 ± 0.00023	2,513,402	1332.06	2.79	57.92
MeshFiLM-S	0.03689 ± 0.00097	0.16648 ± 0.00219	0.04087 ± 0.00050	0.01356 ± 0.00023	**5,894,874**	**568.03**	0.63	**21.64**
MeshFiLM-R	$\mathbf{0.01401 \pm 0.00047}$	$\mathbf{0.05838 \pm 0.00210}$	0.01223 ± 0.00032	$\mathbf{0.00476 \pm 0.00030}$	4,824,560	693.94	**0.45**	25.92

Table 20: AhmedML volume RRMSE.

Model	U_x	U_y	U_z	p	Macro
Transolver	0.01850 ± 0.00009	0.01535 ± 0.00065	0.01947 ± 0.00056	0.01370 ± 0.00076	0.01676 ± 0.00047
Transolver++	0.01675 ± 0.00030	0.01374 ± 0.00039	0.01730 ± 0.00035	0.01262 ± 0.00050	0.01510 ± 0.00037
MeshFiLM-S	0.06432 ± 0.00187	0.03401 ± 0.00090	0.03622 ± 0.00089	0.01304 ± 0.00040	0.03689 ± 0.00097
MeshFiLM-R	$\mathbf{0.01668 \pm 0.00043}$	$\mathbf{0.01250 \pm 0.00025}$	$\mathbf{0.01580 \pm 0.00051}$	$\mathbf{0.01104 \pm 0.00074}$	$\mathbf{0.01401 \pm 0.00047}$

Table 21: AhmedML volume Relative L2.

Model	U_x	U_y	U_z	p	Macro
Transolver	0.02467 ± 0.00012	0.08442 ± 0.00374	0.09943 ± 0.00511	0.05708 ± 0.00361	0.06640 ± 0.00315
Transolver++	0.02367 ± 0.00063	$\mathbf{0.07567 \pm 0.00122}$	0.08738 ± 0.00188	0.04985 ± 0.00283	0.05914 ± 0.00160
MeshFiLM-S	0.10046 ± 0.00111	0.24536 ± 0.00323	0.25470 ± 0.00355	0.06541 ± 0.00123	0.16648 ± 0.00219
MeshFiLM-R	$\mathbf{0.02206 \pm 0.00081}$	0.07603 ± 0.00243	$\mathbf{0.08597 \pm 0.00246}$	$\mathbf{0.04946 \pm 0.00288}$	$\mathbf{0.05838 \pm 0.00210}$

Table 22: AhmedML volume RMSE.

Model	U_x	U_y	U_z	p	Macro
Transolver	0.02252 ± 0.00011	0.01031 ± 0.00044	0.01291 ± 0.00057	0.00782 ± 0.00046	0.01339 ± 0.00039
Transolver++	$\mathbf{0.02013 \pm 0.00074}$	0.00924 ± 0.00033	$\mathbf{0.01116 \pm 0.00032}$	$\mathbf{0.00675 \pm 0.00040}$	$\mathbf{0.01182 \pm 0.00042}$
MeshFiLM-S	0.09155 ± 0.00101	0.02977 ± 0.00039	0.03311 ± 0.00046	0.00903 ± 0.00017	0.04087 ± 0.00050
MeshFiLM-R	0.02160 ± 0.00057	$\mathbf{0.00920 \pm 0.00012}$	0.01134 ± 0.00025	0.00680 ± 0.00039	0.01223 ± 0.00032

On the volume, MeshFiLM-R has the lowest RRMSE on all four fields. Against Transolver++, the ranking depends on the normalization: in RMSE, which is not normalized per sample, Transolver++ is ahead on U_x, U_z and p, by at most 7%. MeshFiLM-S has the largest velocity error of all models, with a relative L_2 error of 25% on U_y and U_z against 8-10% for the others, while its pressure error lies between those of Transolver and Transolver++ (Section 3.3). A forward pass over one test

Table 23: `AhmedML` volume MAE.

Model	U_x	U_y	U_z	p	Macro
Transolver	0.00931 ± 0.00019	0.00436 ± 0.00021	0.00513 ± 0.00036	0.00376 ± 0.00028	0.00564 ± 0.00026
Transolver++	0.00874 ± 0.00042	$\mathbf{0.00374 \pm 0.00015}$	$\mathbf{0.00412 \pm 0.00015}$	$\mathbf{0.00306 \pm 0.00022}$	0.00492 ± 0.00023
MeshFiLM-S	0.03313 ± 0.00056	0.00802 ± 0.00013	0.00877 ± 0.00018	0.00431 ± 0.00008	0.01356 ± 0.00023
MeshFiLM-R	$\mathbf{0.00790 \pm 0.00055}$	0.00384 ± 0.00023	0.00413 ± 0.00021	0.00318 ± 0.00025	$\mathbf{0.00476 \pm 0.00030}$

sample takes 4.5 s with MeshFiLM-S, 5.6 s with MeshFiLM-R, 8.4 s with Transolver and 10.7 s with Transolver++.

G SCALAR OUTPUTS

The controlled comparison in Section 3.2 covers fields only. Table 24 compares the two variants on `Tensile2d` when the fields are trained alone and when fields and scalars are trained jointly. With fields only, MeshFiLM-R has the lower error. With joint training, it keeps the lower field error but has a higher error on each scalar, by 44% on `max_von_mises` and 41% on `max_U2_top`, and MeshFiLM-S has the lower mean over the eight outputs. All three `Tensile2d` scalars are maxima, which routed pooling cannot read off directly (Section 2.1).

Table 24: `Tensile2d`: fields only versus joint training (RRMSE). Fields only: equal-time runs of Table 1, three seeds. Joint: fields and scalars trained together for 5000 epochs, one seed.

Variant	Task	Fields (macro)	`max_von_mises`	`max_U2_top`	`max_sig22_top`	All outputs
MeshFiLM-S	fields only	0.00073	–	–	–	0.00073
MeshFiLM-R	fields only	**0.00068**	–	–	–	**0.00068**
MeshFiLM-S	joint	0.00073	**0.00493**	**0.00216**	**0.00171**	**0.00156**
MeshFiLM-R	joint	**0.00067**	0.00709	0.00304	0.00181	0.00191

Table 25: `Tensile2d`: changing only the operator. All rows use the encoder and the input path of MeshFiLM-R and train fields and scalars jointly for 5000 epochs, one seed. Regions: $[\mathbf{t}_1 \| \cdots \| \mathbf{t}_K]$; statistics: $G([\text{mean} \,\|\, \max \,\|\, \text{std}])$ of the node embeddings. RRMSE, macro over the 5 fields, the 3 scalars, and all 8 outputs.

Context of the FiLM head	Input of the scalar head	Fields	Scalars	All
regions $\| \mathbf{p}$	regions $\| \mathbf{p}$ (MeshFiLM-R)	0.00067	0.00398	0.00191
regions $\| \mathbf{p}$	statistics $\| \mathbf{p}$	0.00067	0.00411	0.00196
regions $\| \mathbf{p}$	regions $\|$ statistics $\| \mathbf{p}$	**0.00066**	0.00398	0.00191
statistics $\| \mathbf{p}$	statistics $\| \mathbf{p}$	0.00072	0.00312	0.00162
MeshFiLM-S (Table 24)		0.00073	**0.00294**	**0.00156**

Table 26: `Rotor37`: changing only the operator. Train fields for 5000 epochs, one seed.

Context of the FiLM head	Fields
regions $\| \mathbf{p}$	**0.00489**
statistics $\| \mathbf{p}$	0.00619

Table 25 separates the operator from the other two differences between the variants. Changing only the context of the FiLM head from routed to statistics raises the field error by 7% and lowers the scalar error by 22%, which brings the model within 6% of MeshFiLM-S on both. Changing only the input of the scalar head has no effect beyond single seed noise: the statistics of the embeddings, maximum included, do not lower the scalar error while the decoder reads routed context.

H PUBLIC LEADERBOARD ENTRIES

Snapshot and selection: Table 3 and the tables below were read from the public leaderboards on *20/09/2026*. Table 3 lists entries whose method has a public description, plus the best score for reference.

Table 27: **Settings of the MeshFiLM leaderboard entries.** The context operator and all settings were chosen per benchmark on the validation split.

Benchmark	Operator	Task	Epochs	Losses (λ_s; auxiliary)	Optimizer	Params
`Tensile2d`	statistics	fields+scalars	5000	$\lambda_s = 0.014; \lambda_\nabla(U1, U2) = 0.01$	AdamW	678790
`VKI-LS59`	routed	fields+scalars	5000	$\lambda_s = 0.014; \lambda_\nabla(mach) = 0.005$	AdamW	636216
`Rotor37`	statistics	fields+scalars	5000	$\lambda_s = 0.014$	AdamW	682660
`2D_profile`	routed	fields	5000	–	AdamW	436020
`2D_MultiScHypEl`	routed	fields+scalars	1000	$\lambda_s = 0.014; \lambda_\nabla(u1, u2) = 0.1$	AdamW	636344

Table 28: `Tensile2d` PLAID benchmarks.

Rank	Id	Total error	U1	U2	sig11	sig22	sig12	max von mises	max U2 top	max sig22 top
1	Stochos DIM-GP	**0.0007**	**0.0004**	0.0005	0.0013	**0.0006**	0.0012	0.0004	0.001	**0.0004**
2	NPCo	0.0013	**0.0002**	0.0003	0.0013	**0.0006**	**0.0009**	0.0044	**0.0006**	0.0016
3	PhysicsX Transolver	0.0013	**0.0002**	**0.0002**	**0.0012**	**0.0006**	**0.0009**	0.0046	0.0008	0.0017
4	MeshFiLM	0.0015	**0.0002**	0.0003	**0.0012**	**0.0006**	**0.0009**	0.005	0.0022	0.0015
5	MMGP	0.0026	0.0015	0.0009	0.0031	0.0013	0.0021	0.005	0.0053	0.0017
6	CRT	0.0034	0.0013	0.0015	0.0057	0.0022	0.0039	0.0079	0.0024	0.0026
7	MARIO	0.0038	0.0023	0.003	0.004	0.0017	0.0023	0.0088	0.0063	0.0023
8	MMVT	0.0054	0.0043	0.0051	0.0089	0.0037	0.0051	0.0068	0.0073	0.0019
9	Augur	0.0066	0.0037	0.0048	0.0081	0.0035	0.005	0.0101	0.014	0.0034
10	MGN	0.008	0.0034	0.0043	0.0047	0.0013	0.0016	0.0169	0.0292	0.0022
11	Vi-Transformer	0.0116	0.0086	0.0091	0.0184	0.0102	0.0146	0.009	0.0203	0.0021
12	FNO	0.0123	0.0174	0.011	0.025	0.0057	0.0135	0.0085	0.0152	0.0021
13	JBone	0.2011	0.3054	0.3891	0.4022	0.2354	0.268	0.0052	0.0017	0.0017

Table 29: `VKI-LS59` PLAID benchmarks.

Rank	Id	Total error	nut	mach	Q	power	Pr	Tr	eth_is	angle_out
1	Stochos DIM-GP	**0.0093**	0.0216	0.0093	0.0025	0.0062	**0.0015**	0	**0.0306**	0.0022
2	PXTransolver	0.0101	**0.0205**	**0.0084**	0.0039	**0.0044**	**0.0015**	0	0.0402	0.0023
3	NPCo	0.0112	0.0243	0.0094	0.0025	0.0067	0.0018	0	0.0433	**0.0019**
4	MeshFiLM	0.0114	0.0239	0.0094	0.0044	0.0053	0.0018	0	0.0437	0.0027
5	MARIO	0.0124	0.0259	0.0112	0.0052	0.0077	0.0018	0	0.0453	0.0023
6	CRT	0.0137	0.0337	0.0185	**0.0015**	0.0053	0.0018	0	0.0466	0.0023
7	SAIR	0.0138	0.0278	0.0122	**0.0015**	0.0049	0.0025	0	0.0586	0.0026
8	Vi-Transformer	0.0193	0.0498	0.0232	0.0052	0.0083	0.0024	0	0.0621	0.0031
9	FNO	0.0215	0.0846	0.018	0.0047	0.0062	0.0019	0	0.0539	0.0027
10	Augur	0.0227	0.0424	0.0221	0.012	0.0113	0.0027	0	0.0863	0.0045
11	MMGP+	0.0312	0.0822	0.0309	0.0023	0.0057	0.0026	0	0.1224	0.0033
12	MGN	0.0497	0.0771	0.0156	0.0716	0.0403	0.0064	0.0001	0.1625	0.0241

The entries use one run per benchmark, the settings of Table 27 and the private test sets, so they are not comparable with Table 1. On `Tensile2d`, MeshFiLM matches the entries ranked second or third on all five fields at the four decimals of the leaderboard, and its total error is higher because of the scalars, mainly `max_U2_top` (0.0022 against 0.0006-0.0008), as in Section 3.3. On `VKI-LS59`, it has the third-lowest *nut* error. On `Rotor37`, its errors are 2 to 2.7 times those of the first three entries on every output. On `2D_MultiScHypEl`, it has the fourth lowest displacement errors but is behind MiSe-GNN (Nguyen et al., 2026), a graph network, on all four stress components, the pattern of Table 12.

Table 30: `Rotor37` PLAID benchmarks.

Rank	Id	Total error	Density	Pressure	Temperature	Massflow	Compression_ratio	Efficiency
1	NPCo	**0.0004**	**0.0007**	**0.0007**	**0.0003**	**0.0003**	**0.0003**	**0.0003**
2	PXTransolver	0.0005	0.0008	0.0008	**0.0003**	**0.0003**	**0.0003**	**0.0003**
3	Stochos DIM-GP	0.0005	0.0008	0.0008	**0.0003**	**0.0003**	**0.0003**	**0.0003**
4	MeshFiLM	0.0011	0.0019	0.0019	0.0006	0.0007	0.0007	0.0006
5	MMGP	0.0014	0.0031	0.003	0.0008	0.0005	0.0005	0.0005
6	MARIO	0.0017	0.0035	0.0034	0.001	0.0008	0.0007	0.0006
7	Vi-Transformer	0.0029	0.0063	0.0062	0.0019	0.001	0.0011	0.0007
8	Augur	0.0033	0.0055	0.0053	0.0012	0.0028	0.0028	0.0019
9	MGN	0.0074	0.0114	0.0114	0.0024	0.0061	0.006	0.0071
10	FNO	0.0313	0.084	0.0836	0.0086	0.0046	0.0042	0.0031

Table 31: `2D_MultiScHypEl` PLAID benchmarks.

Rank	Id	Total error	u1	u2	P11	P12	P22	P21	psi	effective_energy
1	NPCo	**0.0098**	**0.002**	**0.002**	**0.0093**	**0.0139**	**0.0092**	0.0137	**0.0257**	0.0021
2	Stochos DIM-GP	0.0101	0.003	0.0031	**0.0093**	**0.0139**	0.0095	**0.0136**	0.0263	**0.002**
3	PXTransolver	0.0107	0.0026	0.0028	**0.0093**	0.014	0.0095	0.0139	0.0262	0.0074
4	MiSe-GNN	0.0209	0.012	0.014	0.0173	0.0312	0.0176	0.0314	0.0328	0.0111
5	MeshFiLM	0.0217	0.0069	0.0073	0.0256	0.0368	0.0263	0.0365	0.0278	0.0066
6	Augur	0.0221	0.0109	0.0114	0.0208	0.0336	0.0212	0.033	0.0274	0.0188
7	FNO	0.0302	0.0115	0.0117	0.0353	0.0513	0.0359	0.051	0.0329	0.012
8	Vi-Transformer	0.0325	0.0173	0.0172	0.0337	0.0581	0.0343	0.0571	0.0312	0.0113
9	Unet	0.035	0.0291	0.0283	0.0349	0.0498	0.0347	0.05	0.035	0.018
10	MARIO	0.0573	0.0336	0.0377	0.0536	0.1067	0.0539	0.1053	0.0456	0.022
11	GP	0.0902	0.0748	0.075	0.0809	0.167	0.0815	0.1667	0.054	0.0216

Table 32: `2D_profile` PLAID benchmarks.

Rank	Id	Total error	Mach	Pressure	Velocity-x	Velocity-y
1	PXTransolver	**0.0134**	**0.0134**	0.0108	**0.0155**	0.0139
2	NPCo	0.0138	0.0148	**0.0097**	0.0172	0.0136
3	Stochos DIM-GP	0.0152	0.0167	0.011	0.0196	**0.0134**
4	MM-GP+	0.0187	0.0208	0.0139	0.0243	0.016
5	Transolver+	0.0199	0.0209	0.0131	0.0267	0.019
6	MARIO	0.0213	0.023	0.0146	0.0264	0.021
7	MeshFiLM	0.025	0.0265	0.0177	0.0316	0.0243
8	CRT	0.0254	0.0278	0.0171	0.0323	0.0244
9	MuFi-MiSe	0.0276	0.0279	0.018	0.0328	0.0315
10	MiSe-GNN	0.0285	0.0309	0.0212	0.0358	0.026
11	Vi-Transformer	0.0309	0.036	0.0167	0.0403	0.0307
12	MMGP	0.0365	0.0439	0.0208	0.0471	0.0342
13	Augur	0.0425	0.0469	0.0248	0.0538	0.0445
14	FNO	0.0972	0.0988	0.0785	0.1148	0.0967

I ADDITIONAL ABLATIONS

Zero-initialized FiLM heads: Zero-initializing the FiLM heads and using $(1 + \gamma)$ makes the initial model equivalent to an ordinary normalized MLP decoder. Training then gradually introduces modulation rather than starting from a strongly perturbed decoder. On Rotor37, a plain gamma with default initialization increases the error by 37% (Table 37).

Auxiliary losses: The effect of auxiliary losses depends on the dataset and on their weight, and is measured on one seed (Table 39). On `Tensile2d`, gradient matching helps only at $\lambda_\nabla = 0.01$ (-7%) and increases the error by 52-62% at $\lambda_\nabla = 0.005$ and 0.04. It helps at every tested weight on `VKI-LS59`, by up to 22%, and on `2D_MultiScHypEl`, by up to 9%. On `Rotor37` it multiplies the error by 5.9, and on `2D_profile` every weight of the velocity-magnitude loss increases the error by 2–6%. We therefore treat auxiliary losses as dataset-specific regularizers rather than as part of the core architecture. We train MeshFiLM with a weighted combination of field losses, scalar losses, and optional physics-inspired auxiliary losses:

$$\mathcal{L} = \mathcal{L}_{\text{field}} + \lambda_{\text{scalar}}\mathcal{L}_{\text{scalar}} + \sum_k \lambda_k \mathcal{L}_k^{\text{aux}}$$

The field loss is mean-squared error or Smooth-L_1 depending on the dataset. The scalar loss is mean-squared error on standardized scalar outputs. Auxiliary terms are used only when they improve validation performance; they include gradient matching on displacement or flow fields, velocity-magnitude losses for selected CFD cases.

For example, on `Tensile2d` we use

$$\mathcal{L}_{\text{Tensile2d}} = \mathcal{L}_{\text{field}}^{\text{MSE}} + \lambda_{\text{scalar}}\mathcal{L}_{\text{scalar}}^{\text{MSE}} + \lambda_\nabla \mathcal{L}_\nabla(U1, U2)$$

On `VKI-LS59`, we use a Smooth-L_1 field loss and optionally match mach number gradients:

$$\mathcal{L}_{\text{VKI}} = \mathcal{L}_{\text{field}}^{\text{SL1}} + \lambda_{\text{scalar}}\mathcal{L}_{\text{scalar}}^{\text{MSE}} + \lambda_\nabla \mathcal{L}_\nabla(\text{mach})$$

On `2D_MultiScHypEl`, the most effective auxiliary term is displacement gradient matching:

$$\mathcal{L}_{\text{MultiSc}} = \mathcal{L}_{\text{field}} + \lambda_{\text{scalar}}\mathcal{L}_{\text{scalar}} + \lambda_\nabla \mathcal{L}_\nabla(u1, u2)$$

All auxiliary losses are computed on the same train split as the supervised targets and are selected by validation performance. All runs in this section use one seed and train on fields and scalars unless stated otherwise, the reported value is the mean RRMSE over all trained outputs. Training time is the total wall-clock time.

Table 33: Number of FiLM blocks in MeshFiLM-S, fields and scalars, one seed.

	`Tensile2d` (2000 epochs)		`VKI-LS59` (2000 epochs)		`Rotor37` (3000 epochs)	
L	RRMSE	Training Time (s)	RRMSE	Training Time (s)	RRMSE	Training Time (s)
3	0.0023819	10466	**0.0157583**	23657	**0.0014194**	45820
5	**0.0020887**	14342	0.0173760	38847	0.0024414	68976

Five blocks lengthen training by 37-64%. Going from $L = 3$ to 5 blocks helps only on `Tensile2d` (-12%) and costs $+10\%$ and $+72\%$ on `VKI-LS59` and `Rotor37`, so we use $L = 3$ everywhere.

Table 34: Combining the node embedding and the context in the FiLM head in MeshFiLM-S.

	`Tensile2d` (3000 epochs)		`VKI-LS59` (3000 epochs)		`Rotor37` (3000 epochs)	
Combination	RRMSE	Training Time (s)	RRMSE	Training Time (s)	RRMSE	Training Time (s)
Sum (default)	**0.0016375**	16892	**0.0127870**	35400	**0.0014194**	45820
Full linear map of $[\mathbf{z}_j \| \mathbf{c}_j]$	0.0023351	19092	0.0134110	49631	0.0017900	59067
Product	0.0023960	16752	0.0168703	35245	0.0019824	45506

Table 35: Decoder mode. `Tensile2d` dataset, task: fields, 2000 epochs, MeshFiLM-R.

Mode	Mean RRMSE	Training Time (s)
FiLM (default)	**0.0007751**	5570
pointwise (no context, no s_{in})	0.0465472	3074
context concatenated to the input	0.0009886	4704
context concatenated to every block	0.0008279	4740

The weight of the scalar loss matters on two datasets: $\lambda_s = 0.03$ lowers the `VKI-LS59` error by 5% and raises the `Rotor37` error by 47%. It changes the error by 3% on `Tensile2d` and by 0.2% on `2D_MultiScHypEl`.

J Comparison of model properties

The two MeshFiLM variants are the only entries of Table 41 that modulate every decoder layer with node-wise FiLM after a single exchange between nodes, in one training stage and without per-sample fitting.

Table 36: Path of the sample-level inputs $\mathbf{s}_{\text{in}}$ on `Rotor37`. MeshFiLM-R, fields only, 1000 epochs, one seed.

Path of $\mathbf{s}_{\text{in}}$	RRMSE	Training time (s)
$\mathbf{p} = P(\mathbf{s}_{\text{in}})$ in the context (default)	0.0071046	16702
appended to $\mathbf{f}_i$, as in MeshFiLM-S	**0.0046937**	19836

Table 37: Zero-initialized FiLM on `Rotor37`, MeshFiLM-S.

Modulation	RRMSE	Training Time (s)
$(\mathbf{1} + \boldsymbol{\gamma})$, zero-initialized head (default)	**0.0014194**	45820
$\boldsymbol{\gamma}$, default initialization	0.0019437	44234

Table 38: Weight of the scalar loss λ_s.

Dataset	Variant, budget	RRMSE ($\lambda_s = 0.014$)	Training Time (s)	RRMSE ($\lambda_s = 0.03$)	Training Time (s)
`Tensile2d`	MeshFiLM-R, 5000 epochs	**0.0019198**	16003	0.0019736	14576
`VKI-LS59`	MeshFiLM-R, 5000 epochs	0.0114185	59245	**0.0107995**	66764
`2D_MultiScHypEl`	MeshFiLM-R, 1000 epochs	**0.0230475**	2998	0.0230871	3025
`Rotor37`	MeshFiLM-S, 5000 epochs	**0.0011526**	76368	0.0016944	76529

Table 39: Auxiliary losses. Gradient matching penalizes the error on the spatial gradients of the listed fields (Czarnecki et al., 2017), the velocity magnitude loss penalizes the error on $\|\mathbf{V}\|$. `2D_profile` is trained on fields only.

Dataset	Auxiliary loss	λ_∇	RRMSE	Training Time (s)
`Tensile2d`	gradient of $U1$, $U2$	0	0.0017645	**11409**
		0.001	0.0018151	17196
		0.005	0.0028656	16861
		0.01	**0.0016375**	16892
		0.04	0.0026907	16890
`VKI-LS59`	gradient of mach	0	0.0162962	**32514**
		0.001	0.0149394	33759
		0.005	**0.0127870**	35400
		0.01	0.0152224	37468
`Rotor37`	gradient of Density, Pressure	0	**0.0014194**	**45820**
		0.002	0.0083721	57813
`2D_MultiScHypEl`	gradient of $u1$, $u2$	0	0.0248268	**3305**
		0.005	0.0238868	4007
		0.1	**0.0226805**	4125
		0.2	0.0236735	4012
`2D_profile`	velocity magnitude	0	**0.0272164**	**13531**
		0.01	0.0277293	13669
		0.03	0.0280517	13879
		0.05	0.0288213	13571
		0.1	0.0284713	13621
		0.2	0.0282261	13576

Table 40: Routed pooling on the `AhmedML` surface: number of regions and random context/query subsets.

Model	**Error Metrics**				**Efficiency**			
	RRMSE	**L2**	**RMSE**	**MAE**	**Throughput (#nodes/s)**	**Latency (s)**	**#Params (M)**	**Time/Ep (s)**
Transolver	0.01208	0.06337	0.00591	0.00292	3,002,568	45.55	3.05	49.94
Transolver++	0.01049	0.05620	0.00493	0.00218	2,389,651	57.23	2.79	61.01
MeshFiLM-S	0.01130	0.06098	0.00526	0.00240	5,090,633	26.86	0.63	25.63
MeshFiLM-R, context = query								
K=8	**0.00967**	**0.05465**	0.00469	0.00198	4,555,593	30.02	**0.45**	26.20
K=16	0.00979	0.05567	**0.00462**	**0.00196**	4,492,946	30.44	**0.45**	**24.46**
K=32	0.01058	0.05721	0.00506	0.00214	4,409,903	31.01	**0.45**	24.83
MeshFiLM-R, $K = 8$, trained with random context and query subsets; 131,072 query nodes at inference								
8,192 context nodes	0.01273	0.06457	0.00576	0.00242	6,145,430	22.25	**0.45**	45.01
32,768 context nodes	0.01078	0.05875	0.00508	0.00215	6,148,477	**22.24**	**0.45**	45.01
131,072 context nodes	0.01077	0.05861	0.00506	0.00213	**6,149,005**	**22.24**	**0.45**	45.01
131,072 fixed context	0.01105	0.06155	0.00547	0.00237	6,142,350	22.26	**0.45**	48.61

Table 41: Where related models exchange information between nodes and how their decoders are conditioned. Descriptions follow the cited papers.

Model	Exchange between nodes	Conditioning of the output	Stages	Per-sample fitting
MGN	along mesh edges, every layer	node-wise decoder after message passing	1	no
Transolver	through M slice tokens, every layer	–	1	no
UPT, LNO, AB-UPT	encoder to latent tokens, latent processor	cross-attention from query points to tokens	1	no
MARIO	none in the decoder	layer-wise shifts from a sample-level code	2*	yes*
BlendedNet	PointNet over surface points	FiLM from predicted design parameters and flight conditions	2	no
enf2enf	query to latent cross-attention	distance-windowed cross-attention to localized latents	2	yes
PointNet / PointNetCFD	max pooling over points, once	pooled feature concatenated to every point	1	no
MeshFiLM-S	statistics pooling, once	node-wise FiLM from the node embedding and a shared context	1	no
MeshFiLM-R	routed pooling, once	node-wise FiLM from the node embedding and a node-specific context	1	no

*For non-parametric geometries, where the shape code is obtained from a signed-distance neural field.

K ROUTING STATISTICS

We measure the routing of MeshFiLM-R (fields only, equal epoch models of Table 1) on the test samples of four datasets: `Tensile2d` (200 samples, 6.1k-11.7k nodes), `VKI-LS59` (168 samples, 36,421 nodes), `Rotor37` (200 samples, 29,773 nodes) and the `AhmedML` volume (125 samples, 24-30M nodes). The weights r^{c}_{ik} and r^{q}_{jk} depend only on the embedding of node i or j, so the statistics are computed over all nodes of every test mesh and do not depend on how a mesh is split into chunks.

Definitions: For a sample with N nodes and a region k

$$
\begin{aligned}
m^{\mathrm{c}}_k &= \tfrac{1}{N}\textstyle\sum_i r^{\mathrm{c}}_{ik} && \text{write mass: share of the write weight on region } k,\\
m^{\mathrm{q}}_k &= \tfrac{1}{N}\textstyle\sum_j r^{\mathrm{q}}_{jk} && \text{read mass: share of the context read from region } k,\\
n_{\mathrm{eff}}(\mathbf{m}) &= \exp\big(-\textstyle\sum_k m_k \log m_k\big) && \text{effective number of regions, between 1 and } K,\\
s^{\mathrm{c}} &= \tfrac{1}{N}\textstyle\sum_i \max_k r^{\mathrm{c}}_{ik} && \text{write sharpness, in } [1/K, 1] \text{ (read sharpness } s^{\mathrm{q}} \text{ likewise)},\\
\mathrm{ESS}_k &= \textstyle\big(\sum_i r^{\mathrm{c}}_{ik}\big)^2 \big/ \sum_i (r^{\mathrm{c}}_{ik})^2 && \text{effective number of nodes behind the summary } \mathbf{t}_k,\\
I^{\mathrm{q}} &= H(\mathbf{m}^{\mathrm{q}}) - \tfrac{1}{N}\textstyle\sum_j H(\mathbf{r}^{\mathrm{q}}_j) && \text{node-specificity of reading (mutual information)},\\
V^{\mathrm{c}} &= \tfrac{1}{2}\textstyle\sum_k \mathrm{std}\big(m^{\mathrm{c}}_k\big) && \text{variability of the write masses across samples.}
\end{aligned}
$$

A summary $\mathbf{t}_k$ is a weighted mean of the nodes, so a small write mass alone does not mean that few nodes contribute to it, ESS_k counts them, between Nm^{c}_k and N. I^{q} is zero when every node reads the same mixture, so that all nodes receive the same context. V^{c} does not see nodes that swap regions while the masses stay the same. On the two datasets whose samples share one mesh topology, we count directly the nodes that keep their main write region across samples. A region is unread if its mean read mass is below 0.01.

The write sharpness is 0.80-0.98, and the write router uses 2.2-5.5 effective regions out of eight. Reading differs from writing. A router shared by writing and reading, as the slices of Transolver, would put every region near the diagonal of Figure 4. The learned routers do not: on all four datasets some regions are read more than ten times as much as they are written. These regions hold 0.1-4.6% of the write mass, their summaries average 0.1-11% of the nodes (ESS_k/N), and they supply 32-89% of the context. Conversely, on three datasets 18-81% of the write mass goes to regions that supply less than 1% of the context.

I^{q} is 9% of the read entropy on `Tensile2d`, 18% on `VKI-LS59`, 31% on `Rotor37` and 43% on the `AhmedML` volume.

Table 42: Routing of MeshFiLM-R ($K = 8$) on the test samples. $\pm$: standard deviation over samples. Read $> 10\times$ written: regions whose read mass is more than ten times their write mass. Unread: read mass below 0.01. Nodes stable: share of the nodes that keep their main write region in at least 90% of the samples (meshes with one topology only).

	`Tensile2d`	`VKI-LS59`	`Rotor37`	`AhmedML` volume
n_{eff}, write	3.74 ± 0.93	4.81 ± 0.37	5.45 ± 0.04	2.20 ± 0.10
n_{eff}, read	5.62 ± 1.19	1.89 ± 0.09	6.03 ± 0.01	5.76 ± 0.20
Sharpness s^{c} / s^{q}	0.80 / 0.42	0.98 / 0.84	0.95 / 0.50	0.89 / 0.62
I^{q}, nats (share of $H(\mathbf{m}^{\text{q}})$)	0.16 (9%)	0.12 (18%)	0.57 (31%)	0.75 (43%)
Regions read > 10x more than written: read / write mass	0.363 / 0.033	0.888 / 0.046	0.323 / 0.014	0.653 / 0.001
ESS_k/N of these regions	0.11	0.001, 0.045	0.017, 0.045	0.006–0.012
Unread: write / read mass	–	0.807 / 0.005	0.217 / 0.001	0.179 / 0.001
Variability V^{c}	0.165	0.083	0.015	0.033
Nodes stable in $\geq 90\%$ of samples	–	44%	87%	–
Node displacement across samples / size (geometry)	7%	10%	1.6%	–

V^{c} is 0.165 on `Tensile2d`, whose shapes and numbers of nodes change between samples, 0.083 on `VKI-LS59`, 0.033 on the `AhmedML` volume and 0.015 on `Rotor37`. The routers are functions of the node inputs alone, so their assignments can change across samples only through these inputs. On `Rotor37`, whose geometries vary little (node positions vary across the test samples by 1.6% of the size of the blade on average, against 10% on `VKI-LS59`), 87% of the nodes keep their main write region in at least 90% of the test samples, against 44% on `VKI-LS59`.

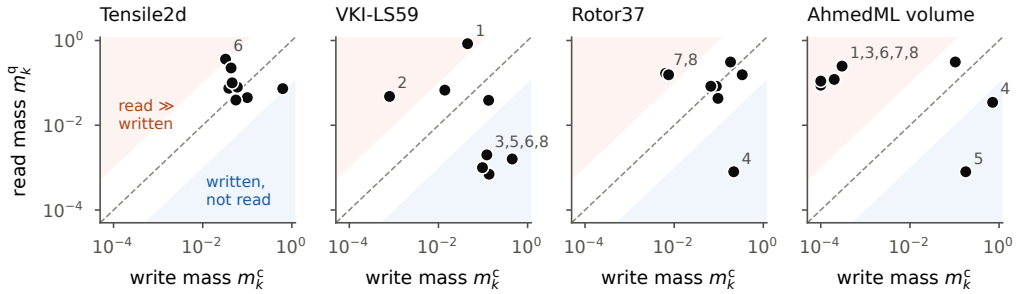

Figure 4: Read mass against write mass of each region (mean over the test samples, log scale). A single router shared by writing and reading, as in the slices of Transolver, would put every region near the diagonal. Shaded: read mass more than ten times the write mass (orange) or less than a tenth of it (blue); labels mark the regions in these areas.

L ADDITIONAL FIGURES

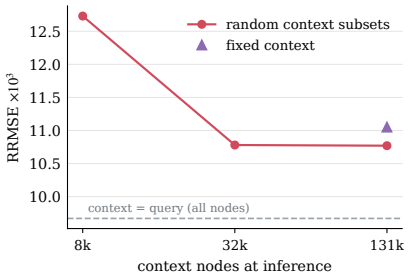

Figure 5: Context subsampling on the `AhmedML` surface. A model trained with independent random context and query subsets reaches its full accuracy from 32,768 context nodes per 131,072 node query chunk. It is 11% behind the default model, whose context is the decoded chunk itself, at 35% higher inference throughput. One seed, raw values in Table 40.

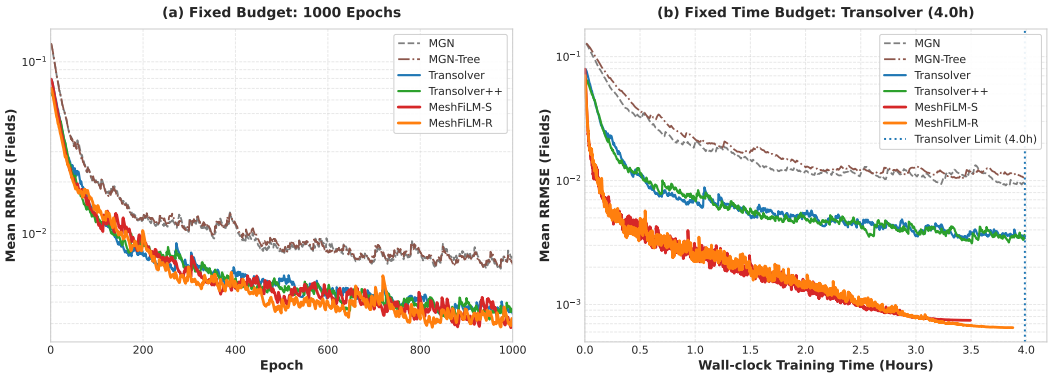

Figure 6: Training curves on `Tensile2d` (macro field RRMSE, validation split, seed 42). (a) Against epochs, up to the equal epoch budget of 1000 epochs. (b) Against wall-clock time; the dotted line marks the equal time budget (about 4h).