THE GRAM 2026 COMPETITION AND WARPED-IFW DATASET

Julian Suk[*, †,1], Gavin Seegoolam[*, †,2], Alison Pouplin[*, †,3], Paul Tiwald[*,3], Ivan Bioli[4], Vedant Bonde[3], Rémi Bourgerie[5], Oscar Breiner[1], Thomas Capelle[6,7], Huaguan Chen[8], Alex Colagrande[9], Aakashnag Davuluri[10], Vinay Edula[11], Vivekananda Edula[12], Bernhard Einberger[13], Massimiliano Ghiotto[4], Francesca Maria Greco[3], Ankit Grover[5], Justin Hodges[7], Theofanis Ifaistos[14,15], Sahib Julka[16,17], Anthony Kalaydjian[18], Samet Kocbay[1], Aakash Kotha[19], Maximilian Leutschafft[1], Ning Lin[8], Morgan McGuire[6,7], Vlad Medvedev[20], Mikel Mendibe[21,22], Deepthi Ravipati[23], Jorge Sarrato-Alós[24,25], Harshit Singh[26], Rajeev Kumar Singh[26], Joshua Stiller[17,27], Vihan Tiwari[28], S. Viswanathan[29], Luis J. Walter[30], Andy Zhang[31]

[1]Technical University of Munich [2]BeyondMath [3]Independent researcher [4]Università di Pavia [5]KTH Royal Institute of Technology [6]Weights & Biases [7]CoreWeave [8]Gaoling School of Artificial Intelligence, Renmin University of China [9]Université Paris Dauphine-PSL [10]Villanova University [11]Indian Institute of Technology Kanpur [12]KL University, Andhra Pradesh [13]AVL List [14]Inria [15]Université Paris-Saclay [16]LMU University Hospital [17]LMU Munich [18]LISN [19]University of California, Davis [20]Fraunhofer Institute for Integrated Systems and Device Technology IISB [21]TECNALIA, Basque Research & Technology Alliance (BRTA) [22]University of the Basque Country (UPV/EHU) [23]Stony Brook University [24]Instituto de Astrofísica de Canarias [25]Universidad de La Laguna [26]Shiv Nadar Institution of Eminence [27]Munich Center for Machine Learning (MCML) [28]Lossfunk [29]Tata Institute of Fundamental Research [30]Heidelberg University [31]Basis Independent Fremont

[*]Co-first authors (equal contribution). [†]Competition organisers.

ABSTRACT

Computational fluid dynamics is central to aerodynamic design. Yet, simulating transient and potentially turbulent flow around a new geometry is very expensive. There are increasing efforts to use machine learning models to reduce the cost of simulation. Given the growing use of geometric inductive biases in machine learning, we wanted to know whether architectural choices meaningfully affect results, and whether a competitive geometric solution could emerge. We turned this question into a competition hosted at the GRaM workshop. With BeyondMath, we release *Warped-IFW* – 181 warped variants of the Imperial Front Wing under detached-eddy flow – and pose one task: given an initial velocity window, predict its continuation. Out of 22 submissions, the top five are all voxel-based. In a post-competition analysis, we find that what actually lowers the error is spatial resolution, not parameter count or architecture family: graph and latent models converge to a plateau that only a finer grid breaks through. Additional details regarding the competition can be found on the competition website.[1]

1 MOTIVATION

Computational fluid dynamics (CFD) is the numerical simulation of fluid flow, obtained by solving the Navier-Stokes equations on a discretisation of space and time. Accurate solutions require fine meshes and small time steps, and a single simulation can take hours, if not days. Data-driven approaches, such as neural operators (Li et al., 2021) or graph-based solvers (Pfaff et al., 2021a), allows for the approximation of fluid flow at a fraction of the computational cost.

In aerodynamic design, one typically wants to predict the flow across a variety of different candidate geometries. For example, iterating over the front wing of a Formula 1 car, in order to improve its

[1]gram-competition.github.io

design with respect to performance indicators. The shape of the wing defines the domain boundary on which a no-slip condition can be enforced, and each geometry then produces a different solution to the same equations. A learned model could in principle recover this geometry-to-flow relationship implicitly from the data, or be equipped with some explicit geometric inductive biases.

This is the exact question that motivated the second edition of the Workshop on Geometry-Grounded Representation Learning and Generative Modeling (GRaM) (Pouplin et al., 2026): *does encoding geometric or generative structure help, or is scaling alone enough?* In order to explore this in practice, we organized a benchmark challenge.

A real-life setting was a priority, which led us to predicting transient 3D flow around airfoils. No such dataset existed publicly, and BeyondMath[2] created one – the Warped-IFW dataset – giving us a new benchmark on which to compare submissions.

The challenge was open for one month, and 22 valid submissions have been received. These spanned a wide range of approaches: neural operators, transformer-based architectures, graph and equivariant models, and MLP baselines. Implementations of all submissions are available on GitHub.[3] The Warped-IFW dataset is available on Hugging Face.[4]

2 Challenge description

2.1 Benchmark task

The following description was given to participants:

Consider a 3D velocity field $u(t, x)$ of airflow around an airfoil where $t_0 \leq t \leq t_1$ and $x \in \Omega \subset \mathbb{R}^3$. Denote the airfoil surface by $\partial\Omega$ and assume no-slip boundary condition:

$$u(t, x) = (0, 0, 0)^\mathsf{T} \quad \text{for} \quad x \in \partial\Omega.$$

Given $u(t, x)$ for $t_0 \leq t \leq t_{0.5}$ our goal is to estimate $u(t, x)$ for $t_{0.5} < t \leq t_1$. In other words, we are looking for a neural operator $G_{\partial\Omega}(t, x)$ conditioned on the geometry of the airfoil $\partial\Omega$ that predicts the velocity field at the following time points based on the preceding ones.

In practice, this can be any model that takes as input a discrete velocity field at a fixed number of time points and points in space, as well as the airfoil geometry. Graph neural networks or transformers could be suitable models for this task.

Aerodynamics usually decompose into low-frequency ("laminar") and high-frequency ("turbulent") components. The preceding velocity field already provides an excellent prior for the low-frequency components of the following dynamics. For this reason, we expect the main difficulty of this challenge to be estimation of high-frequency components in the airflow.

2.2 Evaluation metric

Consider velocity ground truth and estimate $u, \tilde{u}\colon (t_{0.5}, t_1] \times \Omega \to \mathbb{R}^3$ and assume they are square-integrable. In order to compare them, we define a metric

$$\text{"relative } \mathrm{L}^2 \text{ error"} : \sqrt{\frac{\int_{(t_{0.5},t_1]} \int_\Omega \|u(t,x) - \tilde{u}(t,x)\|_2^2 \,\mathrm{d}x \,\mathrm{d}t}{\int_{(t_{0.5},t_1]} \int_\Omega \|u(t,x)\|_2^2 \,\mathrm{d}x \,\mathrm{d}t}}.$$

If we discretise time and space by $\{t^1, t^2, \dots t^{n_t}\}$ and $\{x^1, x^2, \dots, x^{n_x}\}$, respectively, so that $U_{ijk} = u_k(t^i, x^j)$ where $k \in \{1, 2, 3\}$ indexes Cartesian direction (i.e., $u = (u_1, u_2, u_3)^\mathsf{T}$), we can approximate the integrals by Riemann sums:

$$\text{"relative } \mathrm{L}^2 \text{ error"} \approx \sqrt{\frac{\sum_{i=1}^{n_t} \sum_{j=1}^{n_x} \sum_{k=1}^{3} (U_{ijk} - \tilde{U}_{ijk})^2 (\Delta x)_j (\Delta t)_i}{\sum_{i=1}^{n_t} \sum_{j=1}^{n_x} \sum_{k=1}^{3} U_{ijk}^2 (\Delta x)_j (\Delta t)_i}}$$

[2] beyondmath.com

[3] github.com/gram-competition/iclr-2026

[4] huggingface.co/datasets/gram-competition/warped-ifw

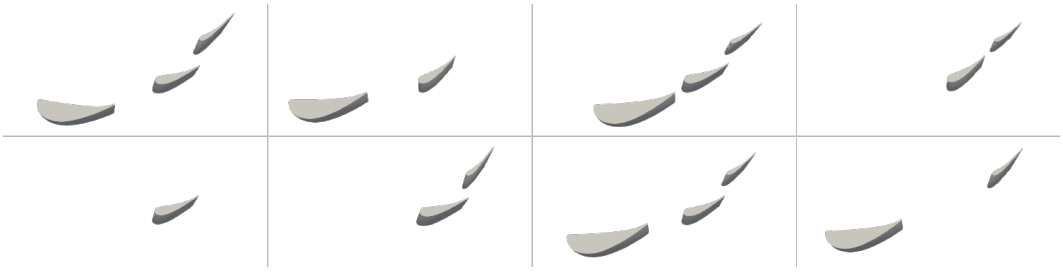

Figure 1: Eight samples from the Warped-IFW dataset.

where $(\Delta x)_j$ are volume elements and $(\Delta t)_i$ are time windows. Note that U and $\tilde{U}$ can be represented by $(n_t \times n_x \times 3)$ tensors. Since dimensions cancel through the division, we can choose $(\Delta x)_j = (\Delta t)_i = 1$ for all i, j.[5]. An implementation of the metric is available on GitHub.

3 DATASET

3.1 GEOMETRY: THE WARPED IMPERIAL FRONT WING

The base geometry is the Imperial Front Wing (IFW), an open source multi-element front-wing assembly in the Formula-1 motorsport domain (Lombard, 2018). We work with a thin spanwise *cross section* of the assembly: each released case contains one to three wing elements whose airfoil sections lie in the x-z plane (chordwise/vertical) and with a width of ($\sim 0.05\,\mathrm{m}$) in the spanwise y direction (Fig. 1). Lift (downforce) acts along $-z$ and drag along $+x$. A single element has a streamwise chord of $\mathcal{O}(0.5\,\mathrm{m})$ and the assembly reference length is $1\,\mathrm{m}$.

Geometric diversity is generated by *warping* the base elements: per-element rigid rotations (angle of attack), translations and (an)isotropic scalings are applied, producing a family of distinct multi-element configurations. The released dataset spans 181 geometries. Figure 1 shows sample geometries from the Warped-IFW dataset.

3.2 FLOW REGIME AND GOVERNING EQUATIONS

The simulations solve the incompressible Navier-Stokes equations for air, $\nu = 1.509\,42 \times 10^{-5}\,\mathrm{m}^2/\mathrm{s}$, $\rho = 1.229\,\mathrm{kg}/\mathrm{m}^3$, at a uniform inflow $U_\infty = 50\,\mathrm{m}/\mathrm{s}$. The Mach number is $\mathrm{Ma} = U_\infty/c \approx 0.15$, comfortably incompressible. The chord-based Reynolds number is $\mathrm{Re}_c = U_\infty c/\nu \approx 2 \times 10^6$ (single element) and $\mathrm{Re}_L \approx 3.3 \times 10^6$ on the $1\,\mathrm{m}$ reference length. The lateral/top/bottom boundaries are held at free-stream velocity (free-air, no wind-tunnel floor boundary layer), the inlet is a fixed velocity and the outlet a fixed pressure.

3.3 TURBULENCE MODEL AND NUMERICS

The flow is computed with the transient incompressible solver `pimpleFoam` (OpenFOAM v2312) using a Spalart-Allmaras Delayed Detached-Eddy Simulation (SA-DDES) (Spalart et al., 2006; 1997) hybrid RANS-LES closure, with the sub-grid length scale $\tilde{d}$ built from the maximum cell edge (`maxDeltaxyzCubeRoot`). DDES runs the attached boundary layer in (unsteady) RANS mode and resolves the separated wake in LES mode. On the wing surfaces the eddy viscosity uses the continuous, all-y^+ `nutUSpaldingWallFunction`, i.e. Spalding's single law of the wall (Spalding, 1961) valid from the viscous sublayer through the logarithmic layer; $\tilde{\nu}$ is fixed to zero at the wall.

Time integration is second-order implicit (`backward`); momentum convection uses `linearUpwindV` and the turbulence transport `limitedLinear`. The PIMPLE loop uses 5 outer correctors, 2 pressure correctors and 4 non-orthogonal correctors with the GAMG pressure

[5]For heterogeneous discretisation, this is at the cost of "locally warping spacetime", or alternatively, weighting more finely resolved regions more highly.

solver. The fixed time step $\Delta t = 2 \times 10^{-4}\,\mathrm{s}$ gives a mean Courant number of 0.05; the maximum Courant number reaches ≈ 43 in the finest near-wall/refinement cells, which the implicit PIMPLE scheme tolerates. Each case is advanced to $t = 0.5\,\mathrm{s}$ and the statistically stationary window $t \in [0.3, 0.5]\,\mathrm{s}$ is retained for the dataset.

Meshes are initialised by generating a coarse background block. This is then refined to surface level with feature-edge capture and distance-based volume refinement, then nine prism layers (expansion ratio 1.43, relative first-layer thickness) are extruded on the wings. The example case has 902 708 cells (765k hexahedra, the remainder split/polyhedral at refinement interfaces), with a near-wall first-layer thickness of $\approx 66\,\mu\mathrm{m}$. Mesh quality is acceptable but not pristine (Section 3.4).

3.4 Limitations

We state the dataset's limitations plainly. (i) No experimental validation. No wind-tunnel or reference data exist for these warped geometries; we make no claim of quantitative fidelity to a physical wing. What we do guarantee is *internal consistency*: a fixed solver, closure and meshing recipe yield a homogeneous, geometry-coupled data distribution with a consistent $y^+ \approx 20$ across all 181 geometries. (ii) No formal mesh-convergence study. (iii) Wall-modelled boundary layer. Fine-scale near-wall structures are modelled, not resolved.

These choices are deliberate trade-offs in service of *feasibility* and *geometric grounding*: the benchmark is small enough to iterate on quickly yet captures the geometry-driven turbulent wake that is the real object of interest.

3.5 Information-preserving subsampling

In order to facilitate model training, we preprocess the data. Our goal is to greatly reduce the memory requirement of each sample while keeping an acceptable level of physical information. Consider a simulation mesh comprising n vertices $X = \{x^1, x^2, \ldots, x^n\}$, $X \subset \Omega$ endowed with velocity vectors $u(t, x^j)$, $x^j \in X$. We first crop out vertices that are further away from the airfoil surface $\partial\Omega$ than a fixed threshold

$$X \leftarrow \{x^j \in X \mid \inf_{x' \in \partial\Omega} \|x^j - x'\|_2 < 0.4\}$$

based on the observation that the velocity field is largely homogeneous beyond that threshold. Subsequently, we subsample the resulting set of points based on the following procedure:

1. We subsample X semi-randomly while ensuring that a fraction of the samples on the Dirichlet boundary $X \cap \partial\Omega$ remains present. Denote as $X_{\text{sub}} \subset X$ the subsampled set.
2. We interpolate the subsampled velocity field back to the source resolution. Each source point $x^j \in X$ receives a linear combination of velocity vectors $u(t, x')$ based on the four nearest neighbours $x' \in \mathcal{N}_4(x^j) \subset X_{\text{sub}}$ in the subset, proportional to Euclidean distance:
$$\tilde{u}(t, x^j) = \frac{\sum_{x' \in \mathcal{N}_4(x^j)} \alpha_{x', x^j} u(t, x')}{\sum_{x' \in \mathcal{N}_4(x^j)} \alpha_{x', x^j}}, \quad \alpha_{x', x^j} = \frac{1}{\|x' - x^j\|_2^2 + 10^{-16}}$$
3. We compute, for $x^j \in X - \partial\Omega$ (excluding the Dirichlet boundary), the average relative error between the interpolated and source field and *reject* the subsampled set X_{sub} if
$$\frac{1}{|X|} \sum_{x^j \in X} \frac{\|u(t, x^j) - \tilde{u}(t, x^j)\|_2}{\|u(t, x^j)\|_2} \geq 11\%$$
for any point in time $t_0 \leq t \leq t_1$. If X_{sub} is rejected, we repeat the procedure.

The above procedure allows us to reduce the number of points for each simulation to 100,000. We store positions and velocity vectors in 32-bit precision leading to 2.4 MB memory per time step.

4 Competition results

In total, we received 22 valid submissions. We evaluate submissions via relative L^2 error (section 2.2). In Table 1 we provide the mean and standard deviation for each submission across the test split.

Ranking	Pull request	Model	Class	Relative L^2 error (↓)
1st	#14	SmoothSplatNet	Voxel	**0.0480** $\pm$ 0.0146
2nd	#17	CDFDoubleGridNet	Voxel	**0.0498** $\pm$ 0.0144
3rd	#19	VRTEnsemble	Voxel	<u>0.0510</u> $\pm$ 0.0151
4th	#4	Kagent	Voxel	0.0553 $\pm$ 0.0168
5th	#9	ResMLP	Voxel	0.0560 $\pm$ 0.0169
6th	#2	EnsembleSpatioTemporalModels	Graph	0.0589 $\pm$ 0.0184
7th	#21	TransolverAR	Latent	0.0638 $\pm$ 0.0198
8th	#13	TransolverCorrector	Mixed	0.0739 $\pm$ 0.0203
9th	#10	gEGNO	Graph	0.0819 $\pm$ 0.0233
10th	#16	FiniteGraphV4	Graph	0.0837 $\pm$ 0.0230
11th	#20	AB-UPT	Latent	0.0838 $\pm$ 0.0227
12th	#6	AirFormer	Latent	0.0850 $\pm$ 0.0227
13th	#12	AeroChronoMixer	Pointwise	0.0892 $\pm$ 0.0232
14th	#5	Transolver Residual	Latent	0.0896 $\pm$ 0.0244
15th	#11	LeversTailV2Submission	Graph	0.0970 $\pm$ 0.0230
16th	#8	ImprovedMLP	Pointwise	0.1002 $\pm$ 0.0238
17th	#7	FNO3DTimeRes	Latent	0.1224 $\pm$ 0.0218
18th	#25	DeltaGraph	Graph	0.1258 $\pm$ 0.0278
19th	#23	SpatiotemporalMNO	Mixed	0.1292 $\pm$ 0.0284
20th	#18	WaveletLatentOperator	Voxel	0.1978 $\pm$ 0.0163
21st	#3	PerceiverFlow	Latent	0.2436 $\pm$ 0.0406
22nd	#24	ZonalMoE	Graph	0.5807 $\pm$ 0.1091
OOC	#22	AIRLiquid	Graph	0.1613 $\pm$ 0.0204
OOC	#26	PT-PointNet (+ voxel U-Net)	Voxel	0.0629

Table 1: **Ranking of submissions.** We list model ID and the corresponding error mean $\pm$ standard deviation. Bold and underlined numbers reflect significance testing explained in Section 4. The architectural class of each model (voxel, graph, latent, and others) is defined in the post-competition analysis (Section 5). The final two rows are out-of-competition (OOC) submissions with their own self-reported measures.

The most accurate model is SmoothSplatNet, followed by CDFDoubleGridNet and VRTEnsemble. (One-way) analysis of variance (ANOVA) reveals that the top three models do not differ significantly in accuracy (p-value > 0.05). Moreover, the fourth-place model Kagent is statistically comparable to VRTEnsemble (p-value ≈ 0.067).

5 Post-competition analysis

Three architectural classes across the submissions. Across the twenty-two valid submissions, the submissions fall roughly into three architectural classes, distinguished by *where* each model represents and processes the velocity field (See Table 1 and the appendix for model descriptions). **Voxel** (or grid) models scatter the point cloud onto a regular three-dimensional lattice and process it with convolutions. **Graph** models leave the cloud untouched and allows each point to exchange information with its nearest neighbours, by message passing or attention. **Latent** models compress the point cloud into a small set of internal tokens (a few learned "slices", anchor points, or supernodes) before undergoing transformations in that compressed space followed by decoding back to the point cloud. The three classes make genuinely different choices about *where the field lives while it is being computed*— on a grid, on the points themselves, or in a compressed bottleneck — and that distinction organises everything that follows.

Top solutions pair local encoding with spatial coupling. One feature of the leaderboard is hard to miss: the entire top five (relative L^2 errors from 0.048 (#1) to 0.056 (#5)) are voxel-based. SmoothSplatNet (#1), CDFDoubleGridNet (#2), VRTEnsemble (#3), Kagent (#4), and ResMLP (#5) differ in their details but share one skeleton—a *point–voxel hybrid* built from two branches. A **point**

branch encodes each of the 100,000 points independently, producing a feature vector that captures what happens at that point but nothing about its surroundings. A **voxel branch** provides the coupling: it places those features onto a grid, runs a three-dimensional convolutional U-Net so neighbouring cells exchange information, samples the result back to the points, and adds it as a correction (`h = h + unet(h, pos)`). The point branch encodes each point in isolation; the voxel branch lets nearby points share information. Two of the five sharpen this by warping the grid so cells cluster where the flow is complex (#1 and #2), but the other three reach the top five on a uniform grid, so what the leaderboard rewards is membership in the voxel class itself, not the warping refinement.

The graph and latent classes are well represented, yet their best entries stop just short of the top five: the strongest graph model placed sixth (0.0589) and the strongest latent model seventh (0.0638). This near-miss is what the rest of the section sets out to explain, through controlled, from-scratch ablations in which every configuration is trained under one fixed protocol.

Question 1: which part of the winning design does the work? The top five all pair a point branch with a voxel branch, but the leaderboard alone cannot tell whether the accuracy comes from the point branch, the voxel branch, or the fusion itself. We answer this by rebuilding the design from its bare parts: training each branch alone, then the fused pair, and then swapping one component at a time, so that every change in accuracy can be attributed to the single component that changed.

Question 2: what does the voxel class have that graph and latent do not? Spatial coupling cannot be the answer on its own, since graph and latent models couple points too by message passing and attention. Yet they still fall short. We answer this by isolating which property of the grid actually drives its accuracy, then handing the graph class the same structural advantages under the same protocol, to test whether the voxel edge survives once the other classes get a fair chance to close the gap.

5.1 METHOD

The ablation ladder. The analysis rebuilds each architecture from its bare components and adds them back one at a time, forming a *ladder* of configurations in which each rung changes a single component. As such, any change in accuracy is attributable to that component. Each rung of this ladder varies only how the model represents and processes the field *in space*: its grid, its graph, or its resolution. How the model handles *time*, i.e. the way it turns the inputs into the five output frames, is held fixed across every rung, so its effect reflects the spatial mechanism alone and not a temporal confound. We build the ladder first and most extensively for the winning voxel design (See Sections 5.2–5.3), then for a hierarchical graph model [6] to test whether the same lessons carry to a different class.

Fixed temporal stage. Every configuration predicts all five output frames in a single pass, as a residual against the last observed frame, through a fixed output head (linear for the voxel and point-cloud models, a small per-node attention across the five frames for the graph models). This is a scope boundary, not a claim that temporal modelling is unimportant: several competitors do vary it, such as the autoregressive teacher forcing of TransolverAR (#7) (Wu et al., 2024) or the pushforward training that CDFDoubleGridNet (#2) applies to half of each batch.

Training protocol. Every rung is trained under one fixed protocol: learning rate 5×10^{-4} on a cosine schedule for at most 150 epochs, bf16 mixed precision, and an exponential moving average of the weights (decay 0.999) at evaluation. This protocol is deliberately minimal with one single-model and single-pass, and without any ensembling, test-time augmentation, composite losses, or grid warping. Its one augmentation is a training-time y-mirror (negating the spanwise coordinate and velocity to exploit left-right symmetry, doubling the training set), not to be confused with the test-time y-flip averaging that entries like VRTEnsemble (#3) use to lower their error. Most runs share a single per-point input feature set (point position, the five input velocity frames, and distance to the airfoil surface). The few that use a reduced set are flagged where they appear and do not change the conclusions.

[6] A hierarchical graph model pools the point cloud to coarser sets of points and restores detail through skip connections. It can be seen as the graph counterpart of the voxel U-Net (Section 5.4)

Evaluation. Performance is reported on a 95-sample, geometry-disjoint holdout from the post-competition test release, split three ways: (*All*) all points; (*Near-surface*) the 10% with the highest temporal velocity variance: a thin layer near the airfoil that carries most of the error; and (*Rest*) the remaining 90% . Because our protocol omits the heavier machinery some submissions rely on (three- and four-member ensembles, test-time augmentation, adaptive warps), any later comparison between a single model of ours and an ensembled entry should be read only as a direction, not a like-for-like result.

5.2 While both branches are necessary, only the voxel branch is decisive

We answer Question 1 in two steps: we first remove each branch in turn, then vary the identity of the branch that remains swappable.

Each branch alone plateaus. We begin by testing the two halves of the design separately. A **voxel-only** family omits the point branch entirely: it averages the raw inputs onto the grid, processes them with a three-dimensional convolutional network, and samples the result back to the points. Two rungs span it: a flat-convolution baseline (VoxelBaseline) and the same network with a U-Net encoder-decoder hierarchy (VoxelUNet). The two reach almost identical error, a relative L^2 near 0.0845 (Table 2), and the hierarchy on its own does not improve accuracy. A **point-only** family performs no better. Its point branch is a stack of residual-MLP blocks (each a LayerNorm, an expanding linear layer, a GELU, and a contracting linear layer, wrapped in a skip connection), applied independently at every point. Run with no grid branch (ResMLPNoUNet), it plateaus at 0.0878. In isolation, each branch settles in the mid-0.08s.

Combined, they leave the plateau. Inserting the voxel-U-Net branch between the point branch's pre- and post-blocks, as a residual addition (ResMLPMin), lowers the holdout error to 0.0605, which is a 31% relative improvement over the same point branch without it.[7] It is the first configuration to approach the top-five band, and since neither branch alone drops below the mid-0.08s, the gain comes from combining them rather than from either part.

Model	Point	Voxel	Params	All	Near-surface	Rest
VoxelBaseline	✗	✓[a]	0.69M	0.0844	0.2868	0.0357
VoxelUNet (+ U-Net hierarchy)	✗	✓[b]	5.78M	0.0847	0.2878	0.0360
ResMLPNoUNet	✓[c]	✗	1.61M	0.0878	0.3003	0.0361
ResMLPMin	✓[c]	✓[b]	7.72M	**0.0605**	0.2048	0.0264
PTPointNet (+ voxel-U-Net)	✓[d]	✓[b]	6.34M	0.0629	0.2127	0.0274

Table 2: **The voxel ablation ladder.** Relative L^2 on the 95-sample holdout, over all points, the near-surface layer, and the rest. Each single-branch model plateaus in the mid-0.08s; fusing the two (rung B) leaves the plateau for the 0.06s. **Branch variants:** [a]flat 3D convolution; [b]U-Net encoder–decoder hierarchy; [c]residual-MLP point branch; [d]kNN-attention point branch.

The point branch is interchangeable. To identify which branch drives this gain, we vary the point branch while keeping the voxel branch fixed. Replacing the residual-MLP point branch with the kNN-attention branch of PTPointNet[8] gives 0.0629 (Table 2), within sample noise of ResMLPMin. Once the grid-based voxel branch is present, the point branch is interchangeable, whether or not it couples points itself.

This leaves the complementary question: **does the voxel branch design matter as little?** In the next subsection, we aim to answer this question by varying the receptive field and the resolution of the voxel branch.

[7]ResMLPMin is exactly ResMLPNoUNet with the voxel branch, so the two differ only in that branch. It is also a faithful single-pass reconstruction of the fifth-place ResMLP submission, without that entry's test-time augmentation and four-member ensemble.

[8]A point transformer in which each point attends over its k nearest neighbours.

U-Net levels	Reach (streamwise)	Params	All	Near-surface	Rest
1	$\sim$8%	2.18M	0.0658	0.2206	0.0297
2	$\sim$27%	3.29M	0.0641	0.2163	0.0281
3	$\sim$64%	7.72M	0.0632	0.2135	0.0277

Table 3: **A larger receptive field brings little gain.** U-Net depth sets the receptive field at a fixed 64^3 grid; tripling the parameter count to extend it from 8% to 64% of the streamwise span improves the holdout by only $\sim$4%. These runs use batch size 16; the three-level row is the same configuration that gives 0.0605 at batch size 4 in Table 2.

5.3 Varying the voxel branch: receptive field saturates, resolution does not

The voxel branch supplies what the point branch cannot compute: spatial coupling. The point branch extracts features point by point, so a feature at one location knows nothing about the field a chord-length away; the voxel U-Net places the per-point features on a grid, lets convolutions exchange information across cells, and samples the result back. Establishing that this coupling is necessary, however, does not yet say *what* about it matters. The coupling has two separable properties, set by two different design choices. The first is the **receptive field**: how far across the domain information propagates, governed by the depth of the convolutional stack (the number of U-Net levels). The second is the **resolution**: how finely the field is represented on the grid the convolutions run on, governed by the grid size. We vary each in turn.

The receptive field saturates early. One might expect the convolution to require a large receptive field. Almost all of the error concentrates in a thin, high-variance layer hugging the airfoil, spanning roughly 1.15 chord-lengths in the streamwise direction, so covering it could plausibly demand chord-scale context. We test this by sweeping the U-Net depth at a fixed 64^3 grid, extending the receptive field from about 8% to 64% of the streamwise span (Table 3). The results do not support this expectation: the holdout error decreases only from 0.0658 to 0.0632, and the near-surface layer, where a larger receptive field should help most, improves comparably little (0.2206 to 0.2135). Beyond a modest size, enlarging the receptive field produces no further gain. This early saturation is itself informative: for this model, each point is best predicted from its local neighbourhood rather than from the far field, consistent with, though not proof of, a short correlation length in the error-driving structures.

Resolution lowers the error monotonically. Resolution behaves entirely differently. Holding the point branch, the depth, and the training fixed, we sweep the grid from 32^3 to 256^3 (Table 4, Figure 2). The error falls monotonically and steeply, $0.0738 \rightarrow 0.0605 \rightarrow 0.0527 \rightarrow 0.0467$, without saturating. The most controlled single step is $64^3 \rightarrow 128^3$, which doubles the grid at identical depth, parameter count (7.72M), and batch size, yet still reduces the error by 0.0078, a gain attributable to resolution alone rather than to added capacity. At the finest resolution, the single 256^3 model reaches 0.0467, below the best in-competition entry (SmoothSplatNet, 0.0480). This comparison carries caveats: it is a single model evaluated in one forward pass, and its per-sample spread ($\sigma \approx 0.014$) overlaps the winner's. The top two submissions reach comparable accuracy more economically, concentrating cells where the field varies most,[9] whereas our uniform 256^3 grid distributes cells evenly across the domain. The warped models place cells cleverly; our uniform grid places them evenly. Since the even grid still matches the clever ones, accuracy seems to depend on how much resolution reaches the critical region, not on how the cells are arranged.

The resolution gains have not yet saturated. Two observations indicate that finer grids would keep helping. First, the error curve is still steep at the finest grid: the $128^3 \rightarrow 256^3$ step lowers the error as much as the earlier steps, with no sign of diminishing returns. Second, the data carry more detail than the grid resolves. Figure 3 compares the spacing between neighbouring cloud points with each grid's cell width: even at 256^3, whose cells are 0.0088 wide, the grid is about four times coarser

[9] SmoothSplatNet concentrates cells through a learned grid warp; CDFDoubleGridNet through a per-sample density warp with dual-resolution grids.

Grid ($x \times y \times z$)	Levels	Params	All	Near-surface	Rest
$32 \times 16 \times 16$	2	3.29M	0.0738	0.2493	0.0322
$64 \times 32 \times 32$	3	7.72M	0.0605	0.2048	0.0264
$128 \times 64 \times 64$	3	7.72M	0.0527	0.1794	0.0225
$256 \times 128 \times 128$	4	25.4M	**0.0467**	0.1594	0.0196

Table 4: **A finer grid brings large gains.** Grid size is swept with the point branch and depth otherwise fixed. Grids have $N \times N/2 \times N/2$ cells, abbreviated by their streamwise count N (e.g. 64^3). Batch sizes are 8/4/4/3. The $64 \rightarrow 128$ step is matched in depth, parameters, and batch, isolating resolution; the 256 grid also adds a U-Net level and channels, hence the parameter jump.

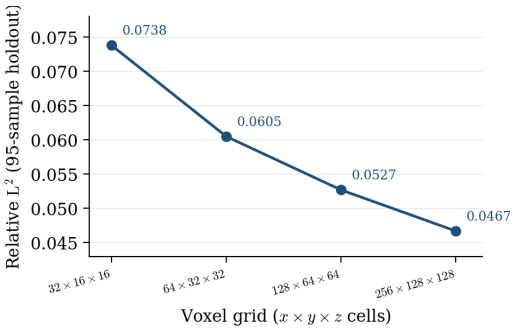

Figure 2: **A finer grid brings large gains.** Relative L^2 on the 95-sample holdout against streamwise cell count ($\log_2$ scale: each tick doubles the resolution). The error falls monotonically and is still steep at the finest grid, suggesting further gains from finer grids.

than the cloud in the near-surface region, where neighbouring points sit roughly 0.0022 apart. At the resolutions we could afford, the limiting factor is therefore the grid, not the data.

This motivates the next step. The point cloud carries detail that no grid we trained has matched, and a graph model computes on the cloud directly, at that native resolution. If resolution is what drives accuracy, a graph should reach at least the accuracy of a refined grid. The next subsection tests this prediction.

5.4 The graph has native resolution, and still plateaus

Question 2 asks what the voxel class has that the others do not. The graph class is a natural test case, because a graph never coarsens its input: it computes on the full 100,000-point cloud, which is finest precisely where the error concentrates. If resolution is what drives accuracy, a graph should reach at least the accuracy of a refined grid. We test this prediction by building a graph transformer from scratch and ablating it as we did the voxel design.

A graph at native resolution falls short of the refined grids. We begin with a flat graph transformer modelled on the strongest graph submission: Ensemble-SpatioTemporal (ranked #6). It uses the same kNN attention with edge-conditioned values[10], the same per-node temporal head, and it operates on the full cloud with no pooling. Our version is deliberately smaller: narrower, with far fewer parameters, and evaluated as a single model in a single pass. The Ensemble-SpatioTemporal model, by contrast, averages the predictions of two trained copies and, where its own prediction is unreliable, falls back to repeating the last observed frame. The two models together indicate the range this architecture covers. Ours reaches 0.0700, and the Ensemble-SpatioTemporal, whose two averaged models are each roughly five times larger, reaches 0.0589 (Table 5). Scale and inference machinery evidently help, yet both results remain above the range the refined grids reached, and the prediction already looks doubtful.

Multi-scale hierarchy helps, but does not close the gap. The comparison so far may be unfair to the graph. The voxel U-Net exchanges information at several scales at once: it repeatedly coarsens the grid, so that convolutions on the coarse levels connect regions that are far apart, while skip

[10]Each point attends over its k nearest neighbours, with the relative position of each neighbour supplied to the attention as an edge feature.

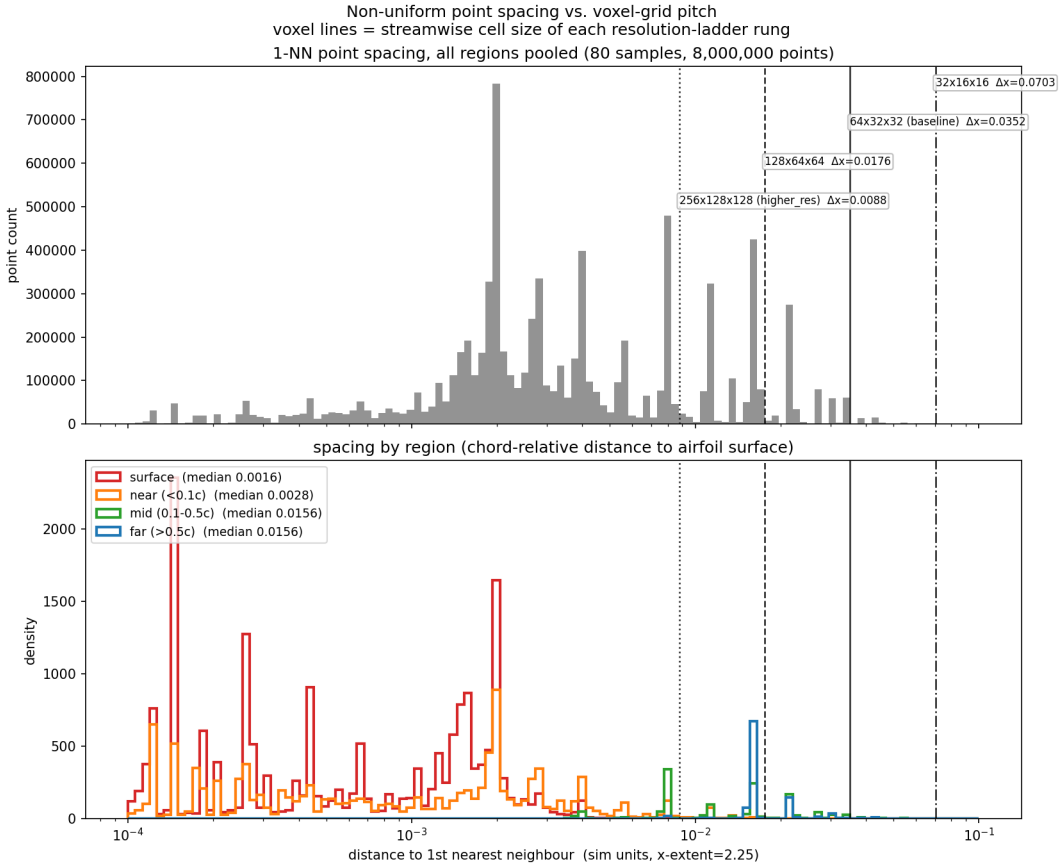

Figure 3: **The cloud is finer than the grid.** Distribution of nearest-neighbour point spacing in the Warped-IFW cloud, with each grid's cell width marked. Even the 256^3 lattice (cell width 0.0088) is coarser than the cloud in the near-surface region (spacing ≈ 0.0022).

Graph model	k	Depth	Hidden	Params	Rel. L^2
Flat graph transformer (ours)	16	8 blocks	128	1.60M	0.0700
+ Graph-U-Net hierarchy (ours)	16	3 levels $\times$ 2	128	2.03M	0.0640
EnsembleSpatioTemporal (#6)	24	10 blocks	256	7.5M $\times$ 2	0.0589

Table 5: **Graph transformers, 95-sample test set.** All three models share the same backbone: a kNN graph transformer with edge-conditioned attention and a per-node temporal head. Ours are single models evaluated in one pass. The EnsembleSpatioTemporal model averages two trained copies (the $\times 2$ on its parameter count) with a persistence fallback. The multi-scale hierarchy with skip connections is the only structural change that lowers the graph error.

connections carry the fine detail back. Our flat graph has no counterpart to this: each attention layer mixes a point only with its k nearest neighbours. The graph's shortfall could therefore be due to this missing hierarchy rather than to the graph itself. To test this, we add the graph analogue: the cloud is pooled to successively coarser subsets of points, processed there, and restored to full resolution through skip connections (a Graph-U-Net). The hierarchy lowers the error from 0.0700 to 0.0640, and it is the only structural change we found that helps; the skip connections are essential, since pooling without them fails to train. The gap remains, however: the graph reaches 0.0640, whereas the 128^3 and 256^3 grids reach 0.0527 and 0.0467 respectively.

Added capacity is unlikely to close it either. The hierarchical model is the best single-pass graph we obtained, and scaling it is unlikely to change the picture, for two reasons. First, capacity was not what improved the voxel models: the decisive step in the resolution ladder occurred at a fixed parameter count ($64^3 \rightarrow 128^3$, both 7.72M, $0.0605 \rightarrow 0.0527$). Second, the EnsembleSpatioTemporal model already shows what scale buys a graph: several times wider and deeper than ours and ensembled over two trained models, it reaches 0.0589, which is the level the 64^3-grid already occupies. Enlarging a graph adds parameters over the same fixed set of points, but it does not refine the locations on which the field is computed. We note that we ran no controlled scaling sweep of the graph, so this argument reads the available results rather than a dedicated experiment.

5.5 Different architectures converge near 0.06, only finer grids fall below

Different architectures converge near 0.06. The separate ablations combine into a single picture (Table 6). The best graph submission (EnsembleSpatioTemporal, #6), the best latent submission (TransolverAR, #7), and our own best graph model all reach errors within a narrow range around 0.06, partly overlapping the ResMLP submission (#5, 0.0560) and our single-model reconstruction of it at 64^3 (0.0605). These models share almost no architectural detail, yet they arrive at nearly the same accuracy. The only models clearly below this range are regular grids refined beyond 64^3: 0.0527 at 128^3 and 0.0467 at 256^3. Stated numerically, the range spanned by the different architecture classes is about 0.005 wide, from our Graph-U-Net at 0.0640 to the sixth-place ensemble at 0.0589, while refining the grid lowers the error by a further 0.012, to 0.0467. Changing the grid resolution therefore moves the error more than twice as much as changing the architecture class.

Resolution, not capacity, separates the grids from the rest. Every change that only adds parameters stops at the same level. A voxel-only network eight times larger matches its baseline (Table 2); tripling the parameter count to widen the receptive field improves the error by about 4% (Table 3); and scaling the graph backbone severalfold, with ensembling, reaches only 0.0589 (Table 5). Resolution is the only form of scaling that continues to lower the error, and among the classes tested, only the grid offers it.

Model	Class	Computes on	Params	Rel. L^2
Graph-U-Net (ours)	graph	full cloud (10^5 points)	2.03M	0.0640
TransolverAR (#7)	latent	64 learned slices	—	0.0638
PTPointNet + voxel branch (ours)	point + voxel	grid, $64\times32\times32$	6.34M	0.0629
ResMLPMin (ours)	point + voxel	grid, $64\times32\times32$	7.72M	0.0605
EnsembleSpatioTemporal (#6)†	graph	full cloud (10^5 points)	7.5M $\times$ 2	0.0589
ResMLP (#5)†	point + voxel	grid, $64\times32\times32$	$\sim$7.7M $\times$ 4	0.0560
ResMLPMin (ours)	point + voxel	grid, $128\times64\times64$	7.72M	0.0527
SmoothSplatNet (#1)†	point + voxel	learned warped grid	8.0M $\times$ 3	0.0480
ResMLPMin (ours)	point + voxel	grid, $256\times128\times128$	25.4M	**0.0467**

Table 6: **All routes compared**, on the 95-sample test set. The third column gives the set of locations on which each model computes the field; the fourth its parameter count, where $\times k$ denotes a k-member ensemble ("—": not reported). †Full competition submission with ensembling and/or test-time augmentation; our models are single models evaluated in one pass, so these comparisons are directional rather than matched.

Two caveats. First, the TransolverAR and EnsembleSpatioTemporal results are reported numbers from the competition submissions rather than our own runs, and the EnsembleSpatioTemporal model averages two models, so its 0.0589 is not a matched single-model figure. Second, although every configuration predicts a residual on the last observed frame, the graph models use a small attention head over the five output frames[11] whereas the voxel models use a linear one. This difference lies in the fixed temporal stage rather than in the spatial mechanism under test, but it means the gap

[11] We kept the graph models close to the EnsembleSpatioTemporal (#6) submission it resembles, rather than matching the voxel track's linear head.

between the graph and voxel classes cannot be attributed to the spatial design alone down to the last digit.

6 CONCLUSION

In summary, adding parameters left every class near 0.06; refining the grid took the voxel models, and only the voxel models, below it. Our experiments establish this pattern but do not explain it, so what follows is interpretation rather than measurement.

Two explanations are already excluded. The ablations rule out the two most natural answers. It is not the receptive field, which saturates within a few U-Net levels (Table 3). And it is not sampling density, because the point cloud is finer than every grid precisely where the error concentrates (Figure 3), yet the models that compute directly on the cloud do not benefit from that fineness.

The remaining candidate is the operator itself. A convolution on a regular grid computes each output from a fixed pattern of neighbouring cells, with one learned weight per neighbour, reused at every location. In numerical analysis this pattern is called a *stencil*, and it is how finite-difference schemes represent derivatives. The construction relies on the grid regularity: every cell neighbours sit at the same relative positions, so the same weights apply everywhere. This makes the convolution a reliable representation of the local spatial derivatives of the velocity field, the quantities that govern its evolution, and refining the grid makes that representation finer. On a point cloud, each point's neighbours sit at different distances and directions, so message passing must approximate the same derivatives with one shared function that takes each neighbour's relative position as an input; this is a weaker approximation, and no parameter of the graph refines it. One measurement supports this account directly, though it is confounded: at matched receptive field, the convolution outperforms the graph, 0.0658 to 0.0700, but with more parameters.[12]

The hypothesis remains to be tested. No experiment we ran isolates the operator effect. Two would: a convolution-versus-graph comparison at matched parameter count, swept across resolutions, and per-layer diagnostics measuring how accurately each architecture represents derivative operators. Until such tests exist, the operator account is a hypothesis consistent with our results, not a demonstrated mechanism.

Limits. The benchmark comprises 181 geometries assembled for accessibility rather than validated against experiment (Section 3.4), the temporal design was held fixed throughout, and we ran no controlled scaling study of the non-grid classes. These boundaries, together with the two tests above, define the future work.

Does geometry help, or is scaling enough? The competition asked whether encoding geometric structure helps, or whether scaling alone suffices. The answer this analysis suggests is not one-sided. The models that operate natively on the geometry (graphs on the exact point cloud, equivariant message passing) were outperformed by a plain convolution scaled in resolution. Yet the two winning entries used geometry where it pays most: in the discretisation, through warps that concentrate grid cells near the surface. On this benchmark, geometric structure entered most profitably through the discretisation, not through the operator.

ACKNOWLEDGMENTS

We are deeply grateful to BeyondMath for building the dataset tailored for this competition: 181 detached-eddy simulations of the Imperial Front Wing were generated, curated and released openly for this challenge. That is a considerable gift of compute, and expertise. We hope Warped-IFW long outlives this competition. We also warmly thank the Munich Center for Machine Learning (MCML) for sponsoring the winner prize.

[12] Both cover roughly 8% of the streamwise span; the convolution has 2.18M parameters against the graph's 1.60M (Tables 3 and 5).

We would also like to thank the GRaM organizing team and the ICLR 2026 organizers for hosting us. Above all, we are incredibly grateful to all the teams who submitted their model and analysis. Every conclusion in this report is built on work they chose to share.

REFERENCES

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. *arXiv preprint arXiv:2502.09692*, 2025.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Lennart Bastian, Samuel Leventhal, Mustafa Hajij, and Tolga Birdal. Topological neural operators, 2026. URL https://arxiv.org/abs/2606.09806.

Lukas Biewald. Experiment tracking with Weights and Biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL https://arxiv.org/abs/1412.3555.

Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. In Sébastien Ourselin, Leo Joskowicz, Mert R. Sabuncu, Gözde B. Ünal, and William M. Wells III (eds.), *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016 - 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II*, volume 9901 of *Lecture Notes in Computer Science*, pp. 424–432, 2016. doi: 10.1007/978-3-319-46723-8_49. URL https://doi.org/10.1007/978-3-319-46723-8_49.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs*, 2021.

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. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2017.12.012. URL https://www.sciencedirect.com/science/article/pii/S0893608017302976. Special issue on deep reinforcement learning.

Fabian B. Fuchs, Daniel E. Worrall, Volker Fischer, and Max Welling. SE(3)-transformers: 3D roto-translation equivariant attention networks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 9224–9232. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00961. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Graham_3D_Semantic_Segmentation_CVPR_2018_paper.html.

GRaM Competition Organizers. GRaM ICLR 2026 competition track. GitHub repository, 2026a. URL https://github.com/gram-competition/iclr-2026.

GRaM Competition Organizers. GRaM ICLR 2026 competition results. Competition website, 2026b. URL https://gram-competition.github.io/. Accessed 2026-06-24.

Jakob Hansen and Thomas Gebhart. Sheaf Neural Networks. In *TDA & Beyond*, 2020. URL https://openreview.net/forum?id=GgcgIJsT8HD.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, March 1964. ISSN 0003-4851. doi: 10.1214/aoms/1177703732. URL http://dx.doi.org/10.1214/aoms/1177703732.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J. Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations (ICLR)*, 2022.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8500–8509, 2022.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Hannah Lawrence, Elyssa Hofgard, Yuxuan Chen, Tess Smidt, and Robin Walters. Detecting symmetry-breaking in molecular data distributions. In *AI for Accelerated Materials Design - ICLR 2025*, 2025a. URL https://openreview.net/forum?id=yEvdOXW5iY.

Hannah Lawrence, Vasco Portilheiro, Yan Zhang, and Sékou-Oumar Kaba. Improving equivariant networks with probabilistic symmetry breaking. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=ZE6lrLvATd.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Zongyi Li, Nikola B. Kovachki, Christopher B. Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Animashree Anandkumar. Geometry-informed neural operator for large-scale 3d pdes. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/70518ea42831f02afc3a2828993935ad-Abstract-Conference.html.

Yi-Lun Liao and Tess Smidt. Equiformer: Equivariant graph attention transformer for 3D atomistic graphs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=KwmPfARgOTD.

Levi Lingsch, Mike Y Michelis, Emmanuel De Bézenac, Sirani M Perera, Robert K Katzschmann, and Siddhartha Mishra. Beyond regular grids: Fourier-based neural operators on arbitrary domains. *arXiv preprint arXiv:2305.19663*, 2023.

Jingzhe Liu, Haitao Mao, Zhikai Chen, Tong Zhao, Neil Shah, and Jiliang Tang. Towards neural scaling laws on Graphs, 2024. URL https://arxiv.org/abs/2402.02054.

Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel CNN for efficient 3D deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019a.

Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel CNN for efficient 3d deep learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 963–973, 2019b. URL https://proceedings.neurips.cc/paper/2019/hash/5737034557ef5b8c02c0e46513b98f90-Abstract.html.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Jean-Eloi Lombard. *A high-fidelity spectral/hp element LES study of Formula 1 front-wing and exposed wheel aerodynamics*. PhD thesis, Imperial College London, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019a.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019c.

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(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 Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/luo25o.html.

Songrit Maneewongvatana and David M. Mount. Analysis of approximate nearest neighbor searching with clustered point sets, 1999. URL https://arxiv.org/abs/cs/9901013.

Morgan McGuire, Thomas Capelle, and Justin Hodges. SENPAI: Self-ExperimentatioN for physical AI: An observability-based research harness. OpenReview submission, 2026. URL https://openreview.net/forum?id=g0bJFA9gVT. Manuscript under review for ICML 2026.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017. URL https://arxiv.org/abs/1710.03740.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=roNqYL0_XP.

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)*, 2021b.

Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

Alison Pouplin, Sharvaree Vadgama, Sékou-Oumar Kaba, Manuel Lecha, Jakub M Tomczak, Robin Walters, Stefanie Jegelka, and Erik J Bekkers. Geometry-grounded representation learning and generative modeling. In *ICLR 2026 Workshop Proposals*, 2026.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. *Advances in neural information processing systems*, 35:23192–23204, 2022.

Xuebin Qin, Zichen Zhang, Chenyang Huang, Chao Gao, Masood Dehghan, and Martin Jagersand. BASNet: Boundary-aware salient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Osborne Reynolds. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society of London. A*, 186:123–164, 1895.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

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.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks, 2022. URL https://arxiv.org/abs/2102.09844.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

P. R. Spalart, W.-H. Jou, M. Strelets, and S. R. Allmaras. Comments on the feasibility of les for wings, and on a hybrid rans/les approach. In *1st AFOSR International Conference on DNS/LES*, 1997.

P. R. Spalart, S. Deck, M. L. Shur, K. D. Squires, M. Kh. Strelets, and A. Travin. A new version of detached-eddy simulation, resistant to ambiguous grid densities. *Theoretical and Computational Fluid Dynamics*, 20(3):181–195, 2006.

D. B. Spalding. A single formula for the law of the wall. *Journal of Applied Mechanics*, 28(3): 455–458, 1961.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020a.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020b.

Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24261–24272, 2021. URL https://arxiv.org/abs/2105.01601.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017a. Curran Associates Inc. ISBN 9781510860964.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017b. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ. accepted as poster.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Qinxuan Wang, Chuang Wang, Mingyu Zhang, Jingwei Sun, Peipei Yang, Shuo Tang, and Shiming Xiang. Mno: Multiscale neural operator for 3d computational fluid dynamics, 2026. URL https://arxiv.org/abs/2510.16071.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.*, 38(5):146:1–146:12, 2019. doi: 10.1145/3326362. URL https://doi.org/10.1145/3326362.

Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. ConvNeXt V2: Co-designing and scaling ConvNets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16133–16142, 2023.

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*, pp. 53681–53705. PMLR, 2024.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10524–10533. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/xiong20b.html.

Minkai Xu, Jiaqi Han, Aaron Lou, Jean Kossaifi, Arvind Ramanathan, Kamyar Azizzadenesheli, Jure Leskovec, Stefano Ermon, and Anima Anandkumar. Equivariant graph neural operator for modeling 3D dynamics. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 55015–55032. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/xu24j.html.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 28877–28888. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f1c1592588411002af340cbaedd6fc33-Paper.pdf.

APPENDIX

We asked every team to describe its own entry, in its own words, along four lines:

1. a general description of the method;
2. what did *not* work during prototyping, and what led to the final model;
3. what the team would improve given more time;
4. training details (validation split, additional data, and so on).

The second point is the one we cared about the most, as those "failure modes" are almost never disclosed in a publication, yet they might help us better understanding what directions to investigate in the future.

The sections that follow are the authors' own accounts. A few submissions came with no write-up, or with a summary alone. For those, we reconstructed the missing material from the code in the competition repository, and the section opens with a **Provenance** note saying so. Table 7 lists every submission and where it is described.

Rank	PR	Model	App.	Teams (with affiliations)
1st	#14	SmoothSplatNet	A	Sahib Julka (LMU Munich)
2nd	#17	CDFDoubleGridNet	B	Jorge Sarrato-Alós (IAC & Universidad de La Laguna)
3rd	#19	VRTEnsemble	C	Vinay Edula (IIT Kanpur)
4th	#4	Kagent	D	Thomas Capelle (W&B & CoreWeave), Justin Hodges (CoreWeave), Morgan McGuire (W&B & CoreWeave)
5th	#9	ResMLP Ensemble	E	Andy Zhang (Basis Fremont)
6th	#2	SpatioTemporalModels	F	Aakashnag Davuluri (Villanova), Vivekananda Edula (KL University), Aakash Kotha (UC Davis), Deepthi Ravipati (Stony Brook)
7th	#21	TransolverAR	G	Alex Colagrande (Paris Dauphine-PSL), Theofanis Ifaistos (Inria & Paris-Saclay), Anthony Kalaydjian (LISN)
8th	#13	TransolverCorrector	H	Joshua Stiller (LMU Munich, MCML)
9th	#10	gEGNO	I	Rémi Bourgerie (KTH), Ankit Grover (KTH)
10th	#16	FiniteGraphV4	J	
11th	#20	AB-UPT	K	Paul Tiwald (independent researcher)
12th	#6	AirFormer	L	Vedant Bonde (independent researcher), S. Viswanathan (TIFR)
13th	#12	AeroChronoMixer	M	Huaguan Chen, Ning Lin (Renmin University of China)
14th	#5	Transolver Residual	N	Ivan Bioli (Univ. di Pavia), Massimiliano Ghiotto (Univ. di Pavia), Mikel Mendibe (TECNALIA & UPV/EHU)
15th	#11	LeversTailV2Submission	O	
16th	#8	ImprovedMLP	P	Harshit Singh, Rajeev Kumar Singh (Shiv Nadar IoE, Delhi-NCR)
17th	#7	FNO3DTimeRes	Q	Vlad Medvedev (Fraunhofer IISB)
18th	#25	DeltaGraph	R	Luis J. Walter (Heidelberg University)
19th	#23	SpatiotemporalMNO	S	Oscar Breiner, Samet Kocbay, Maximilian Leutschafft (TUM)
20th	#18	WaveletLatentOperator	T	Alex Brown
21st	#3	PerceiverFlow	U	
22nd	#24	ZonalMoE	V	Vihan Tiwari (Lossfunk)
OOC	#22	AIRLiquid	X	Francesca Maria Greco (independent researcher)
OOC	#26	PT-PointNet	Y	Paul Tiwald (independent researcher)

Table 7: **Submissions metadata.** The *App.* column points to the model's appendix section. A few submissions supplied no write-up of their own, or only a summary; for those, we reconstructed the missing part from the model code and README in the competition repository, and say so at the top of the section concerned. We received two out-of-competition (OOC) submissions. Within each team, authors are listed in alphabetical order.

CONTENTS

A SMOOTHSPLATNET

SMOOTHSPLATNET forecasts the five future velocity frames of a 100k-point unsteady flow; the far field is near-persistent and the difficulty is the sharp, unsteady near-wall field. Because coordinate networks are biased towards low frequencies, position and velocity history are Fourier-encoded to resolve steep near-wall gradients. Graph and neural-operator families never beat a voxel backbone in experiments, so it was strengthened: points are splatted onto a 3D grid with trilinear weights, a 3D U-Net mixes the grid globally, a learned warp redistributes its resolution, and a distance-gated head sharpens near-wall points from raw features. The splat is the dual of the `grid_sample` read-back, so encode and decode are symmetric. A residual on the last frame is trained with plain L_2 on a geometry-disjoint split and no external data; three seeds are averaged. Aligning training with the official relative-L_2 metric is the clearest next step.

A.1 GENERAL METHOD

Setup. Given five past frames $u_i^{1:5} \in \mathbb{R}^3$ on $N \approx 10^5$ fixed points $p_i \in \mathbb{R}^3$ with a surface subset $\mathcal{A}$, the task is to predict the next five $u_i^{6:10}$. The competition scores the per-sample relative L_2 error,

$$\mathcal{E}_b = \Big(\textstyle\sum_{k,i} \|\hat{u}_i^k - u_i^k\|^2 \,/\, \sum_{k,i} \|u_i^k\|^2\Big)^{1/2}, \tag{1}$$

averaged over the test point clouds. Training and model selection instead used the simpler absolute per-point error $\frac{1}{TN}\sum_{k,i}\|\hat{u}_i^k - u_i^k\|$ ($T = 5$), the hint metric published with the task and a close surrogate for the relative score; all figures below are reported in it. The error is not spread evenly: the far field is smooth and well approximated by persistence ($\hat{u}_i^k \approx u_i^5$), so almost all of it comes from the unsteady region near the wing. As a design-motivating observation, the per-point persistence error, averaged over the data, is roughly an order of magnitude larger near the surface (within a few percent of chord) than beyond a third of a chord, and decays smoothly with wall distance; the surface is held at exact no-slip ($u_i = 0$ for $i \in \mathcal{A}$). The mesh is refined to match, with roughly two-thirds of the points within 2% of chord (an indicative figure, not a precise tally), so a *uniform* grid would spend most of its cells in the sparse far field and under-resolve exactly where the error lives. The field also evolves slowly across the ten-frame window (frame-to-frame changes average ~ 1.5 m/s, while velocities span roughly $[-50, 82]$ m/s), so a residual on the last frame is a strong starting point; timestamps are uniformly spaced at $\Delta t = 10^{-3}$ s, and the model does not use t. A model therefore needs a global receptive field for the bulk flow and sub-cell accuracy at the surface at once. Two design choices follow.

Encoding. The difficulty lies in steep near-wall gradients, yet a network reading raw coordinates is biased towards low frequencies (Rahaman et al., 2019) and blurs exactly those gradients. Each scalar coordinate and velocity component is lifted through a Fourier feature map (Tancik et al., 2020a; Mildenhall et al., 2020),

$$\gamma(x) = \big[\, x,\ \sin(2^0\pi x), \cos(2^0\pi x),\ \ldots,\ \sin(2^{L-1}\pi x), \cos(2^{L-1}\pi x)\,\big], \tag{2}$$

which supplies the high-frequency bases that a raw-coordinate MLP struggles to represent; distance-to-surface features (unsigned distance and its logarithm) and a surface indicator complete the per-point input r_i.

Backbone. A voxel U-Net (Ronneberger et al., 2015) gives a global receptive field cheaply, but its standard form is asymmetric: the decoder reads the grid back to points with smooth trilinear interpolation, while a standard encoder snaps each point to one nearest cell, a lossy scatter exactly where near-wall detail matters. Let $\tilde{p}_i = \phi(p_i)$ be the point coordinate after a learned monotone per-axis warp ϕ (a piecewise-linear CDF with softmax bin widths, identity at initialisation) that lets the grid allocate resolution non-uniformly (Durkan et al., 2019), and let c index the eight grid cells around $\tilde{p}_i$ with trilinear weights

$$\omega_i(c) = \prod_{a \in \{x,y,z\}} \big(1 - |\tilde{p}_{i,a} - c_a|\big). \tag{3}$$

The decoder gathers, $\hat{f}_i = \sum_c \omega_i(c)\, V[c]$ (trilinear `grid_sample`). The encoder is made its *dual* by splatting with the same weights,

$$V[c] = \frac{\sum_i \omega_i(c)\, f_i}{\sum_i \omega_i(c)}, \tag{4}$$

so writing to and reading from the grid share one interpolation kernel (Liu et al., 2019a), unlike nearest-cell scatter $\omega_i(c) = \mathbf{1}[\,c = \lfloor \tilde{p}_i \rfloor\,]$; Table 8 summarises these changes against a standard voxel U-Net. Squeeze-and-excitation recalibrates channels (Hu et al., 2018) and residual-MLP blocks (He et al., 2016) surround the U-Net. A distance-gated, zero-initialised head then corrects near-wall points directly from the raw features r_i,

$$h_i \leftarrow h_i + g(r_i) \odot \psi(h_i, r_i), \qquad g(r_i) = \sigma\big(\mathrm{MLP}(r_i)\big) \in (0,1)^d, \tag{5}$$

whose gate increases towards the wall, so the correction concentrates near the surface, where the grid loses the most detail (Qin et al., 2019). This is consistent across the three members, with gate

Table 8: What SMOOTHSPLATNET changes relative to a standard voxel U-Net: a high-frequency input encoding, a smooth point↔grid interface, and a near-wall bypass, rather than a bigger network.

Aspect	Standard voxel U-Net	SMOOTHSPLATNET (this work)
Input encoding	raw coordinates / velocities	Fourier features (position + history)
Point→grid encode	nearest-cell scatter (lossy)	trilinear splat to 8 cells
Encode/decode symmetry	asymmetric with `grid_sample`	dual of `grid_sample`
Grid layout	uniform	learned monotone warp
Near-wall pathway	routed through the grid	distance-gated head on raw features
Channel mixing	plain convolutions	+ squeeze-and-excitation
Inference	single model	3-seed equal-weight ensemble

values of ≈ 0.36 at the surface against ≈ 0.13 in the far field (indicative rather than precisely logged). The output is a residual on the last frame with no-slip imposed exactly,

$$\hat{u}_i^k = u_i^5 + \sigma_{\text{vel}} \odot \Delta_i^k \;\; (i \notin \mathcal{A}), \qquad \hat{u}_i^k = 0 \;\; (i \in \mathcal{A}), \tag{6}$$

with Δ zero-initialised so training starts from persistence. The submission averages three independently seeded members of 8.0M parameters each, run in parallel. The smooth trilinear splat, in place of the hard nearest-cell scatter, gives SMOOTHSPLATNET its name; here "smooth" is continuous, not Gaussian.

The encoding and the backbone are complementary. The global-mixing half of that claim has direct support from the development history: a graph hierarchy with the same per-point encoding but only *local* message passing, and no global aggregation, plateaued at a geometry-holdout error of ≈ 1.08, and adding a full-resolution voxel U-Net for global mixing, with the encoding unchanged, cut it to 0.84 (the first dual-stream model). Global coupling, not more local capacity, was the lever: a pointwise or purely local model cannot propagate the global flow, and the reference pointwise MLP, which also ignores the surface mask, is weak for the same reason. A strong global model, in turn, still leaves the error near the wall, which is exactly what the Fourier encoding and the raw-feature boundary bypass target on top of the grid. That half rests on the spectral bias of coordinate networks (Rahaman et al., 2019) and on the observation that the final near-wall gains came from these changes; an isolated encoding-only vs. backbone-only comparison is the first controlled ablation to run. Either way, the design was reached by elimination across model families, not from any single prior system.

A.2 WHAT DID NOT WORK, AND HOW THE DESIGN CONVERGED

The design did not start from a voxel model; it was reached by elimination over many iterations. Across those iterations the line of work drifted more than once towards added complexity – richer routing, factored heads, auxiliary pressure supervision – that seldom repaid its cost. *Graph / message-passing* models fit the unstructured mesh and are naturally wall-aware, but a k-NN layer only mixes local neighbourhoods (far-field coupling needs many layers), the spatial encoder is rerun per frame though the geometry is fixed, and attention over 100k points is costly; autoregressive rollout compounded error. *Neural operators* (a Fourier neural operator (Li et al., 2021) stream, latent operator-transformers) give a global receptive field by construction but never beat a well-tuned voxel U-Net while adding complexity. The voxel family won; its only real liability was the point-to-grid interface above, which this work redesigns.

Three recipe choices then mattered as much as the backbone, each a negative result. Weighting the loss towards the wall and later timesteps raised the plain-L_2 validation error (Table 9), so the unweighted objective was kept. Y-flip test-time augmentation hurt everywhere; the domain is mirror-symmetric about $y = 0$, which made the flip natural to try, but the flow itself is not y-symmetric, so the final path uses none (Table 11). A fourth seed gave no gain over the best three, and the weakest single seed did not improve any ensemble it joined, so the submission averages three equal-weight seeds (Table 10).

Ensembling also helps on unseen geometries: on an 85-window geometry holdout, a three-seed average cut a single member's error from 0.696 to 0.532. That holdout used the 42/52/57 triple, not the submitted 52/57/85. The submitted trio was instead selected on `hard162`, a 162-window

in-distribution stress set: the hardest window per geometry, with most geometries seen in training. Table 10 therefore measures fit rather than generalisation. The genuinely held-out evidence is the geometry-holdout val L_2 (Table 9) and the per-seed 85-window holdouts (Table 11, right). A later richer-geometry variant did not displace the submission (Table 12). Numbers are mean per-point L_2 (lower is better); RAW is a single pass, TTA is Y-flip; holdouts are unseen geometry windows.

Table 9: Objective alignment (geometry-holdout val L_2, same backbone).

Training loss	best val L_2
plain L_2 (unweighted)	**0.8415**
wall + time weighted	0.8867

Table 10: Seed and ensemble selection on `hard162` (RAW, equal weights).

System	L_2	System	L_2
single seed 42	0.55957	seeds 52+85	0.43342
single seed 52	0.47657	seeds 57+85	0.43056
single seed 57	0.47043	seeds 42+52+57	0.44469
single seed 85	0.46830	**seeds 52+57+85**	**0.41872**
seeds 52+57	0.43547	seeds 42+52+57+85	0.42880

Table 11: RAW vs. Y-flip TTA. Left: `hard162`. Right: per-seed 85-window holdouts with the fraction of samples on which RAW wins.

`hard162`	RAW	TTA	holdout	RAW	TTA	RAW wins
seed 52	0.47657	0.72505	seed 52	0.63154	0.77042	84/85
seed 57	0.47043	0.71647	seed 57	0.69592	0.83646	85/85
ensemble	0.44469	0.68416	seed 42	0.74730	0.85798	82/85

A.3 Why a voxel operator here, and where it would not be

This is a benchmark on *simulated* data whose mesh is fixed within each sequence, with only the field evolving; that suits a grid-based operator, since one voxelisation serves all frames and pooling gives global coupling cheaply. This result is not evidence that voxel grids dominate flow modelling in general. On real-world data, with irregular or adaptive meshes, moving geometry, or strong out-of-distribution generalisation, mesh-native, point-based, and operator-learning models carry inductive biases this setting happens not to need: graph-network simulators (Pfaff et al., 2021a; Sanchez-Gonzalez et al., 2020), operator transformers such as Transolver (Wu et al., 2024), and operator learning (Lu et al., 2021; Li et al., 2021) are built for exactly those conditions. Newer or more general architectures are not automatically better on a fixed-mesh simulation; here, a well-engineered voxel U-Net with a high-frequency encoding is the right tool for the data at hand.

A.4 What to try with more time

The cheapest untried change is to align the training loss and model selection with the official relative L_2, the standard metric in neural-operator work (Li et al., 2021), rather than the absolute surrogate used here; the relative metric reweights samples by their velocity energy, which the absolute loss ignores. Two voxel-family directions were prototyped but never properly evaluated and deserve a full training budget: ConvNeXt-style 3D blocks (Liu et al., 2022) for larger per-block receptive fields, and a deeper multiscale stack for longer-range correlations. Genuinely open directions include a physics-aware constraint such as a continuity (divergence) penalty (Raissi et al., 2019); ensemble diversity beyond random seeds (Lakshminarayanan et al., 2017); an asymmetry-aware augmentation in place of the Y-flip that failed here; and the clean single-component ablations noted above (encoding-only versus backbone-only, and removing splatting, the warp, or the boundary head in isolation). Richer wall-geometry conditioning, by contrast, was tried and did not help (Table 12).

Table 12: A later richer-geometry variant on `hard162` does not displace the submission.

System	L_2
richer-geometry variant, single seed	0.48851
richer-geometry variant, 2-seed ensemble	0.44651
submitted smooth-splat 3-seed ensemble	**0.41872**

A.5 TRAINING DETAILS

The optimiser is AdamW (Loshchilov & Hutter, 2019c) at peak learning rate 3×10^{-4} and weight decay 10^{-5}; the schedule uses three warm-up epochs then cosine decay (Loshchilov & Hutter, 2017) up to 150 epochs with early-stopping patience 20; and training runs in mixed precision (float32 coordinates) with gradient clipping at 1.0. The objective is the plain per-point L_2 (Euclidean) error, with no loss weighting or auxiliary loss in the final recipe; this is the absolute hint metric, not the official leaderboard score, which is a per-sample relative L_2 error. Batch size is 1, since the point count and surface-index set vary per sample. Validation is a geometry-based 10% holdout whose key drops only the trailing window suffix, so held-out geometries are truly unseen (no window leakage); seed, ensemble, and TTA decisions were additionally checked on `hard162` and per-seed holdouts. No external data is used, only the provided files. The train-time-only pressure field, explored as auxiliary supervision in intermediate variants, is not used by the final model. All figures here are absolute per-point L_2 on internal splits (`hard162` is an in-distribution stress set), not the official relative L_2, so absolute numbers differ from any leaderboard score even though the trends hold.

B CDFDOUBLEGRIDNET

CDFDoubleGridNet is a dual-resolution voxel U-Net with density-adaptive CDF coordinate mapping. For each axis the CDF of the point density is computed and coordinates are warped so that voxel boundaries concentrate where points are densest. Two warped grids (32^3 coarse, 80^3 fine) process features with 3D U-Nets receiving per-cell geometric context; per-point features (Fourier encodings, learned SDF embedding, temporal self-attention) are fused with the voxel output. The model predicts velocity residuals with a hard no-slip mask on airfoil surfaces. Training uses a 98/2 split of 905 samples with AdamW, a relative + absolute L_2 loss over fluid points only, pushforward unrolling, and decaying input noise. Before this design, GNNs hit memory limits; sparse convolutions degenerated to per-voxel MLPs at low occupancy; and log-scale warping was less adaptive. With more time, ensembling and CDF regularization refinement would be priorities.

B.1 CDFDOUBLEGRIDNET: FULL ARCHITECTURE DETAILS

CDF-based adaptive grid mapping. Each sample contains $N = 100{,}000$ points with positions $\mathbf{x}_i \in \mathbb{R}^3$, which we normalize to $[0, 1]^3$. For each spatial axis $d \in \{x, y, z\}$ we build a density-adaptive coordinate warp as follows:

1. Compute a histogram of point positions with $R_{\text{fine}} \times 16 = 1{,}280$ bins.
2. Blend the histogram with a uniform distribution (weight $\beta = 0.3$) to reduce empty regions and large jumps in adjacent voxel sizes. This value was empirically calibrated; a learned mapping could improve results.
3. Smooth with a Gaussian kernel ($\sigma = 2.0$).
4. Compute the cumulative distribution function (CDF) and use it as a monotonic coordinate warp $\phi_d \colon [0, 1] \to [0, 1]$.

The warped coordinates $\tilde{\mathbf{x}}_i = (\phi_x(x_i), \phi_y(y_i), \phi_z(z_i))$ are used for all point-to-voxel and voxel-to-point operations. The inverse CDF gives non-uniform voxel boundaries in physical space, from which per-cell widths $\Delta_d^{(j)}$ are derived and passed as geometric context to the U-Net.

Dual-resolution voxel U-Nets. Two independent paths operate at resolutions $R_c = 32$ (coarse) and $R_f = 80$ (fine). Each path proceeds as follows:

1. **Point-to-voxel** Liu et al. (2019b)**:** average point features within each voxel cell using the CDF-warped coordinates.
2. **Context injection:** concatenate a 7-channel geometric tensor to the voxel features (log-scaled cell widths $\log \Delta_x, \log \Delta_y, \log \Delta_z$; log-volume; and physical cell centers X, Y, Z), then fuse with a $1\times1\times1$ convolution.
3. **3D U-Net** Çiçek et al. (2016)**:** a 3-level encoder-decoder with $3\times3\times3$ convolutions, GroupNorm, GELU activations, skip connections, average-pooling for downsampling, and trilinear interpolation for upsampling. The coarse path uses 32 mid-channels; the fine path uses 128.
4. **Voxel-to-point:** interpolate processed features back to point locations via trilinear `grid_sample` using the CDF-warped coordinates.

Coarse and fine outputs are concatenated and fused with a LayerNorm + Linear + GELU layer.

Per-point feature pipeline.

- **Fourier positional encoding** Tancik et al. (2020a)**:** 8 frequency bands applied to the normalized (x, y, z) coordinates, producing 48 features.
- **SDF embedding:** Euclidean distance to the nearest airfoil point, encoded by a 2-layer MLP (inputs: raw and log-scaled distance) into 16 dimensions.
- **Airfoil mask:** binary indicator.
- **Temporal attention** Vaswani et al. (2017b)**:** the 5 input velocity frames (5×3) per point are projected to 32 dimensions, processed by a single-layer multi-head self-attention module (4 heads) across the time axis, and flattened to $5 \times 32 = 160$ features.

All features are concatenated and projected to the hidden dimension (512) by a linear layer, then processed by 2 residual blocks before being aggregated into voxel grids. After the voxel outputs are interpolated back to points, 4 additional residual blocks refine the result. Each residual block consists of LayerNorm + Linear + GELU + Linear with a skip connection.

Output head and boundary conditions. A final LayerNorm + Linear layer projects to $5 \times 3 = 15$ output channels (zero-initialized), interpreted as velocity residuals for 5 future frames. These are added to the last input frame (persistence baseline). A hard no-slip mask zeros all outputs at airfoil surface points.

Training details.

- **Data:** all 905 samples; 98% train / 2% validation split (sorted by filename). All 100k points used, no subsampling.
- **Optimizer:** AdamW, lr $= 3 \times 10^{-4}$, weight decay $= 10^{-4}$.
- **Schedule:** 5-epoch linear warmup followed by cosine decay over 200 total epochs.
- **Loss:** relative L_2 + $0.01\times$ absolute L_2, computed over fluid points only. Airfoil points are excluded because their velocity is always zero, and including them would bias the loss toward samples with larger airfoils (which occupy more of the fixed 100k points).
- **Pushforward training** Brandstetter et al. (2022)**:** with 50% probability, instead of predicting all 5 target frames in a single forward pass, the model is unrolled autoregressively: the first pass predicts only frame 1, which is appended to a shifted input window (dropping the oldest frame) for a second pass that predicts the remaining 4 frames. Gradients flow through both passes, training the model to be robust to its own prediction errors.
- **Noise injection:** Gaussian noise on velocity ($\sigma_{\text{max}} = 0.01$) and positions ($\sigma_{\text{max}} = 0.005$), linearly annealed to zero over 150 epochs. Noise is zeroed for airfoil points.
- **Data augmentation:** random y-axis flip with 50% probability.
- **Precision:** bfloat16 mixed precision with FSDP; gradient checkpointing on U-Nets and temporal attention.

- **Hardware:** 2× NVIDIA RTX 6000 Ada (48 GB each), batch size 2 per GPU, 4 gradient accumulation steps (effective batch size 16).
- **Geometry caching:** SDF values, CDF coordinates, and cell widths are precomputed on GPU and cached in RAM before training to avoid repeated computation.

Test-time augmentation. At inference we average predictions from the original sample and its y-axis reflection (flipping the y-component of input positions and velocities, then unflipping the predicted y-velocity). The CDF mapping is simply recomputed from the reflected positions.

Approaches explored during development.

- **Message-passing GNNs** Wang et al. (2019) (EdgeConv with radius graphs and geometry-conditioned FiLM layers): $O(N \cdot k)$ graph construction and per-edge features made 100k points prohibitive. Memory forced subsampling to 16–64k points, losing boundary-layer detail.
- **Hybrid voxel + GNN:** a single 32^3 coarse grid for global context combined with a short-range GNN for local refinement. Compute spent on the GNN component turned out to be more valuable when reallocated to a finer adaptive grid. We also likely did not find the best GNN configuration (e.g. radius size, maximum neighbour count).
- **Sparse 3D convolutions** Graham et al. (2018) (`spconv` at 128^3): motivated by the observation that a small fraction of voxels are occupied. However, at this occupancy the submanifold $3\times3\times3$ kernel almost never finds occupied neighbours, so the convolution reduces to a per-voxel pointwise transform with no convolutional benefit.
- **Log-scale grid mapping:** warping coordinates via $\log(1 + \alpha x)$ to concentrate resolution near the airfoil. Less adaptive than the data-driven CDF and performed worse on geometries with multiple separated airfoils.
- **Physics-informed losses** (divergence penalty via SPH approximation): introduced training instability without clear accuracy gains; dropped in favor of noise-based regularization.

C VRTENSEMBLE

VRTEnsemble, the Volumetric Routing Transformer, is a hybrid point–volume–point model for spatiotemporal velocity forecasting on 3D CFD point clouds. Each member builds geometry- and dynamics-aware pointwise features (Fourier position encodings Tancik et al. (2020a), velocity history and spectra, boundary distance/direction, and a learned temporal encoder), refines them with SwiGLU residual blocks Shazeer (2020), then *routes* information point → lattice → point through a 3D ConvNeXt-V2 UNet Woo et al. (2023) on a $64 \times 32 \times 32$ grid to inject global context. A zero-initialised residual head predicts five future frames relative to the last observation, with no-slip enforced on airfoil nodes. The deployed predictor is a four-member ensemble with y-reflection test-time augmentation that averages eight outputs.

C.1 PROBLEM SETTING AND TENSOR INTERFACE

VRTEnsemble addresses spatiotemporal velocity forecasting on unstructured 3D point clouds. The model maps $T_{\text{in}}= 5$ historical frames to $T_{\text{out}}= 5$ future frames over $N = 100{,}000$ points with $C = 3$ velocity channels. The competition callable signature is

$$\mathsf{model}\big(t \in \mathbb{R}^{B\times10},\ \text{pos} \in \mathbb{R}^{B\times N\times3},\ \text{idcs\_airfoil},\ \mathbf{v}_{\text{in}} \in \mathbb{R}^{B\times5\times N\times3}\big) \ \rightarrow\ \mathbf{v}_{\text{out}} \in \mathbb{R}^{B\times5\times N\times3}.$$

The time grid t is accepted for interface compatibility but is not consumed by the network. A no-slip boundary condition is explicitly enforced on the airfoil index set $\mathcal{A}_b$.

C.2 POINTWISE FEATURE CONSTRUCTION

For every point the model concatenates the following descriptors before the initial linear projection to the hidden width $d = 256$:

- **Position Fourier features** Tancik et al. (2020a) with 10 frequency bands ($3(1+2\cdot 10) = 63$ dims).
- **Flattened velocity history** ($5\times 3 = 15$ dims), channel-wise normalised by stored statistics.
- **Last-frame velocity spectral features** with 4 bands ($3 \cdot 2 \cdot 4 = 24$ dims).
- **Temporal encoder features** ($d_{\text{temp}}= 64$ dims) from a 1D convolutional stack over time, fusing mean- and last-step pooled codes.
- **Boundary geometry**: nearest-surface distance, $\log(1+d)$, per-sample normalised distance, inverse distance, an 8-band Fourier encoding of $\log(1 + d)$, and a unit direction vector to the nearest boundary point ($4 + 2 \cdot 8 + 3 = 23$ dims).
- **Flow statistics**: per-point mean and standard deviation of speed over the input window (2 dims).
- **Boundary mask** (1 dim).

Nearest-boundary geometry is computed with chunked `cdist` over surface points for memory safety.

C.3 Pointwise SwiGLU blocks

Pre- and post-routing refinement uses gated residual MLP blocks Shazeer (2020). For a point feature $x \in \mathbb{R}^d$,

$$\tilde{x} = \mathrm{LN}(x), \quad g = \mathrm{SiLU}(W_g\tilde{x}) \odot (W_u\tilde{x}), \quad y = x + W_o\, g,$$

with bias-free projections and hidden width $\approx \frac{8}{3}d$ rounded to an even value. The architecture applies $N_{\text{pre}}= 3$ pre-routing and $N_{\text{post}}= 6$ post-routing blocks; the design is token-local (no pairwise attention), keeping per-point cost low while remaining expressive.

C.4 Volumetric routing core

The routing operator maps unordered point tokens to a Cartesian latent volume, processes it, and maps back to points.

Scatter. With normalised coordinates $x_i \in [0, 1)^3$, each point is assigned a voxel $v_i = \lfloor x_i G_x \rfloor (G_y G_z) + \lfloor y_i G_y \rfloor G_z + \lfloor z_i G_z \rfloor$ on a $G = 64 \times 32 \times 32$ grid, and features are averaged per voxel via `scatter_add` of both sums and counts, $F_v = \big(\sum_{i:v_i=v} f_i\big)/\max(1, n_v)$.

Backbone. A four-scale 3D ConvNeXt-V2 UNet Woo et al. (2023); Liu et al. (2022) with channel pyramid $64\to128\to256\to512$ processes the volume; each block uses a depthwise 7^3 convolution, channel-last LN, a $4\times$ pointwise expansion, GELU Hendrycks & Gimpel (2016), Global Response Normalization (GRN), and pointwise contraction with a residual connection. Skips are *additive* (not concatenated) to reduce memory and bandwidth.

Sample back. The final volume is read at point locations by trilinear `grid_sample` (border padding), and the routed signal is added as a residual: $h \leftarrow h + \mathrm{route}(h)$.

C.5 Decoder and physical projection

A final LN feeds a linear head predicting $3T_{\text{out}}$ residual channels per point, reshaped to $[B, T_{\text{out}}, N, 3]$. The head weights and bias are zero-initialised, so the model starts as a persistence predictor. The residual is added to the last observed frame, and predicted velocities on $\mathcal{A}_b$ are zeroed to enforce no-slip.

C.6 Normalisation and statistics

Each member stores external statistics (`flow_channel_mean`, `flow_channel_scale`, `spatial_bounds_lo`, `spatial_bounds_hi`) in `VRTEnsemble_flow_stats.pt`. These are held as non-persistent buffers (popped on `state_dict` load) and reloaded per-member at inference. Channel statistics normalise velocities; spatial bounds map positions into $[0, 1)^3$ for the lattice, with NaN/Inf guards.

C.7 TRAINING OBJECTIVE

The composite loss combines reconstruction, temporal, boundary, and gradient terms:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{mse}} + \beta\mathcal{L}_{\ell_1} + \gamma\mathcal{L}_{\text{temp}} + \delta\, w_a\, \mathcal{L}_{\text{airfoil}} + \lambda_g\mathcal{L}_{\text{gmse}},$$

where $\mathcal{L}_{\text{temp}}$ matches frame-to-frame deltas $\Delta\hat{u}_{b,t} = \hat{u}_{b,t+1} - \hat{u}_{b,t}$ to the target deltas, $\mathcal{L}_{\text{airfoil}} = \frac{1}{B}\sum_b \text{mean}_{t,n\in\mathcal{A}_b,c}\hat{u}^2$ is the no-slip penalty, and $\mathcal{L}_{\text{gmse}}$ is the MSE of finite differences along the point dimension. The submission uses $\alpha = 1.0$, $\beta = 0.1$, $\gamma = 0.5$, $\delta = 0.2$, $w_a = 5.0$ (effective airfoil multiplier 1.0), and $\lambda_g = 0.5$.

C.8 OPTIMISATION AND DATA PROTOCOL

Members are trained with AdamW Loshchilov & Hutter (2019b) at learning rate 2×10^{-4}, weight decay 10^{-4}, batch size 1, gradient clipping 1.0, and mixed precision, for 300 epochs under a warmup (5 epochs) plus cosine schedule Loshchilov & Hutter (2017). Each VRTEnsemble member has 18,249,359 parameters. The data pipeline uses a simulation-aware split with y-reflection augmentation applied to training data only; validation is kept on the original orientation. Members differ by their (seed, split_seed) pairs to promote ensemble diversity.

C.9 INFERENCE-TIME ENSEMBLING

The deployed predictor (`VRTEnsembleEnsemble`) is a composite inference system rather than a single network. It runs all four members on the original input and on a y-reflected copy (unreflected before aggregation), then averages the eight predictions. An optional hard fallback returns persistence (the repeated last frame, followed by no-slip projection) when the input has very high norm but low temporal change. Both reflection TTA and the hard fallback are enabled by default. Weights are pulled from the HuggingFace repository `vinayedula/VRTEnsemble-ensemble`.

C.10 EVALUATION RESULTS

Final mean metrics on the evaluation set are reported in Table 13.

Table 13: VRTEnsemble ensemble: final mean evaluation metrics.

Metric	Value
Relative L2 error	0.01658
Temporal rollout error	0.01653
High-frequency residual L2	0.20932
Boundary condition error	1.0×10^{-6}
vs. persistence	0.13581
L2 (competition metric)	0.30194
L1	0.14754
MSE	0.18547
GMSE	0.28834

D KAGENT

Kagent used an autonomous multi-agent search over GRaM models. Sixteen coding agents ran in GPU-backed Kubernetes pods on the `gram-mar29/kaggler/*` branches, repeatedly reading the leaderboard, editing the model and training code, training under a 30-minute per-run cap, scoring, committing, and iterating. The submitted model is a two-member Voxel–U-Net ensemble with Fourier position and velocity features, airfoil-distance features, residual prediction from the last input frame, hard no-slip output masking, and y-flip test-time augmentation. Prototyping showed that larger graph/kNN-style models were fragile under the time budget, often OOM or too slow; point subsampling also lost important spatial detail. With more time, multi-seed validation, more diverse full-resolution spatial models, and ensemble-weight tuning on a cleaner held-out split would

be priorities. Models were trained on the public warped-ifw splits; the official submission ranked 4th of 22 valid entries.

Kagent is a competition-oriented product of SENPAI, an observability-based research harness for physical AI McGuire et al. (2026). It runs several coding agents concurrently against the same task. For GRaM, each agent had its own git branch, GPU-backed Kubernetes pod, training script, checkpoint directory, and experiment journal. The organizer pod periodically scored new prediction files, updated a shared leaderboard, and pushed that leaderboard to git. Agents communicated only through the shared volume, git, and Weights & Biases logging; they did not call each other directly. The submitted model was packaged as PR #4 to the GRaM competition repository and was listed as Kagent on the official held-out leaderboard, where it ranked 4th of 22 valid submissions with relative-L_2 error 0.0553 ± 0.0168 GRaM Competition Organizers (2026b).

D.1 Autonomous Run Setup

The run used the `gram-mar29/kaggler/<name>` branch family. Sixteen agents were launched for roughly 30 wall-clock hours. Each agent received a single GPU pod and followed the same loop: inspect the current leaderboard, formulate a local hypothesis, edit the training or inference code, train under the configured 30-minute per-training-run cap, write validation predictions, commit the result, and repeat. The data were the public GRaM warped-ifw splits: five input velocity timesteps over approximately 100k mesh points, with the target being the following five timesteps around Formula-1-style front-wing geometries GRaM Competition Organizers (2026a). Training telemetry and validation metrics were logged to Weights & Biases Biewald (2020).

D.2 Submitted Model

The submitted Kagent model is a two-member Voxel–U-Net ensemble. Each member first builds pointwise features from normalized positions, the five input velocity frames, Fourier encodings of position and last-frame velocity, distance to airfoil-surface points, and an airfoil-surface indicator. A small residual MLP embeds these features, a 3D voxel U-Net scatters point features to a $(64, 32, 32)$ grid and gathers spatial context back to points, and post-MLP blocks predict a normalized residual. The final velocity is the last input velocity plus this predicted residual. At inference time, each member is evaluated directly and with a y-mirror test-time augmentation; mirrored predictions are flipped back and averaged. The two members are then averaged with learned validation weights. Airfoil-surface velocities are hard-masked to zero in the output to enforce the no-slip prior.

D.3 What Did Not Work

The strongest negative result was that more complex spatial models were not automatically better under a strict 30-minute training cap. Several graph, kNN, and larger attention-style ideas were brittle: they hit import issues, out-of-memory failures, or spent too much time constructing neighborhoods over 100k points. Larger models often required subsampling points to fit in memory and time, but this removed important spatial detail and reduced the number of effective full-resolution updates. The internal leaderboard also showed high variance across single runs, so individual architecture comparisons were noisy without repeated seeds. These failures were useful because they pushed the final recipe toward cheap spatial mixing, residual prediction, test-time augmentation, and small diverse ensembles rather than a single very large model.

D.4 Future Improvements

Given more time, we would use a cleaner hidden validation split and multiple random seeds to separate real architecture gains from initialization variance. We would also explore full-resolution spatial methods with cheaper locality than kNN, such as fixed-grid or sparse-neighborhood operators, and expand the ensemble with deliberately decorrelated members rather than only nearby checkpoints. The autonomous harness itself would benefit from stronger safeguards around validation leakage, automatic resubmission of incomplete prediction directories, and branch-level summaries that make successful ideas easier for later agents to reuse without overfitting to public validation feedback.

E RESMLP

The ResMLP Submission's model is built on a point-voxel backbone. For each point, the model receives fourier transforms of coordinates, five velocity-history frames, airfoil-mask, and nearest airfoil-distance features for inference. Pressure integration as a surrogate objective was attempted but failed. Each feature demonstrated noticeable improvement when added. Features pass through a residual MLP and dense voxel U-Net, predicting residual velocity increments. Airfoil points are zeroed for physical accuracy. The main failure cases were late horizon near-wall or boundary-layer examples due to their non-linear nature, while mostly linear far-field regions were nearly perfect. To improve robustness, four checkpoints with unique hyperparameters were selected for the final ensemble. The loss was shell and component MSE on velocity weighted towards later times. The final design prioritized a strong backbone for fast experimentation on weaker hardware.

E.1 RESMLP SUBMISSION

Model details. Fourier features for each point contain the 3D position encoded with ten frequency bands, last-frame features with three frequency bands, and distance to airfoil encoded with 6 frequency bands.

Each checkpoint used has the same overall architecture with hidden dimension of 256:

- Two residual MLP blocks
- Voxelization
- Dense Voxel U-Net on regular grid
- Devoxelization
- Four residual MLP blocks
- Layer Normalization
- Linear Residual Velocity Head

The voxel branch averages point features within voxels. The U-Net contains two downsampling stages.

Training data and objective. The train split used 560 train cases and 125 cases of validation and test each.

Item	Value
Dataset	public warped-IFW
Train cases	560
Validation cases	125
Test cases	125
Inference inputs	coordinates, velocity history, airfoil-mask, nearest airfoil-distance
Pressure input	not used at inference
Additional data	none documented
Objective	shell and component MSE on velocity weighted towards later times
Model selection metric	mean L2 error

Table 14: Training data and objective summary for the ResMLP Submission.

Ensemble. The four checkpoint ensemble is implemented as two weighted pairs where they have weights 0.8 and 0.2 respectively. Pair one has weighted 0.325 and 0.675 while pair 2 has weights 0.35 and 0.65. Test time augmentation in the form of flipping the transverse coordinate and velocity component and averaging TTA and regular result after running it through the same checkpoint.

Failure cases. Development on this model had to be quick for discovering performance improvements and making the deadline. Increasing the capacity of the model through voxels both slowed down training significantly and did not affect performance.

While identification of failure cases being near specific areas was obvious, solving this problem proved to be difficult without damaging far-field performance. Many attempted changes slightly improved on the harder families but suffered catastrophic forgetting on their trained families.

One proposed solution was surface-density voxel features which yielded small gains on one family but suffered in turn on all other families. Local reasoning using both kNN graph paths and attention did not apply anything useful to the model. PointNet++ boundary correctors worsened aggregate validation and test metrics while marginally improving near-airfoil points. Using embeddings for the geometry of the object did not give the model enough meaning likely due to undertraining or lack of data. This resulted in no noticeable difference in performance relative to the baseline. Other objective functions that MSE were attempted such as L1, Huber, and divergence but failed to contribute anything promising.

The most promising next steps would definitely be creating a solid meaningful topological embedder of the structures as a geometric signal to the models. Local testing of the final model achieved L2 error of 0.535062786 on the validation set.

F EnsembleSpatioTemporalModels

EnsembleSpatioTemporalModels is a two-member mean ensemble of full-resolution spatiotemporal GNNs for velocity forecasting on 3D CFD point clouds. Each member builds a k-NN graph ($k = 16$), encodes nodes (Fourier position encoding Tancik et al. (2020a), velocity history, airfoil mask), applies a graph backbone (GAT Veličković et al. (2018), MeshGraphNet Pfaff et al. (2021b), or Graph Transformer Dwivedi & Bresson (2021)) and a temporal self-attention head Vaswani et al. (2017b), then decodes residual velocity deltas with no-slip on airfoil nodes. A base branch and a physics-feature branch (adding distance/direction-to-surface) are averaged. Hierarchical subsampling hurt boundary accuracy, so native 10^5-point fidelity was kept, and absolute decoding was unstable, motivating the residual head. Future work: learned ensemble weights and rollout training. Members are trained independently (300 and 100 epochs), frozen, and mean-combined at inference.

F.1 Problem setting and tensor interface

The model forecasts a 3D velocity field on unstructured point clouds with input horizon $T_{\text{in}}= 5$, output horizon $T_{\text{out}}= 5$, point count $N = 100{,}000$, and $C = 3$ velocity channels. The competition callable signature is

$$\mathsf{model}\big(t \in \mathbb{R}^{B\times 10},\ \text{pos} \in \mathbb{R}^{B\times N\times 3},\ \text{idcs\_airfoil},\ \text{velocity\_in} \in \mathbb{R}^{B\times 5\times N\times 3}\big) \to \mathbb{R}^{B\times 5\times N\times 3}.$$

Boundary indices are sanitised (clamped to $[0, N-1]$) and all tensors are aligned to the active device before indexing. The forward path enforces no-slip by zeroing predicted velocity on $\mathcal{A}$.

F.2 Top-level composition (mean ensemble)

The submitted predictor is a *composite system*, not a single network. Two pretrained full-resolution members are loaded, frozen, and run in eval mode; the prediction is their unweighted mean,

$$\hat{u} = \tfrac{1}{2}\big(\hat{u}_{\text{base}} + \hat{u}_{\text{physfeat}}\big).$$

A **base branch** (`SpatioTemporalGNN`) provides general graph-dynamics, while a **physics-feature branch** (`SpatioTemporalGNNPhysFeat`) adds explicit wall-proximity geometry. An optional *hard persistence fallback* (App. F.9) can replace the ensemble output for pathological inputs. Weights are pulled from the HuggingFace repository `vinayedula/spatiotemporal_models`.

F.3 Shared computational pipeline

Both branches share the same macro-architecture and run at full resolution (no inference-time subsampling).

- **Airfoil mask.** A binary boundary indicator $m_n \in \{0, 1\}$ marks $n \in \mathcal{A}$; it is concatenated to node features and reused for the final hard projection.
- **k-NN graph.** A spatial graph is built from positions with $k = 16$ neighbours, returning neighbour indices, relative positions Δp, and scalar distances d. An exact CPU `cKDTree` backend is used when available.
- **Node encoding.** Features are projected by an MLP with a GELU nonlinearity and LN to the hidden width.
- **Spatial backbone.** A configurable graph network performs neighbourhood message passing (App. F.7).
- **Temporal head.** Each node state is expanded to T_{out} tokens and refined by stacked multi-head self-attention (App. F.6).
- **Decoder + residual.** A pointwise MLP maps each future token to a 3D velocity delta added to the last observed frame.
- **No-slip projection.** $\hat{u}_n(t) = 0$ for $n \in \mathcal{A}$.

F.4 Node features and Fourier encoding

For node n with position $p_n \in \mathbb{R}^3$, flattened history $v_n \in \mathbb{R}^{3T_{\text{in}}}$ (15 dims), and mask m_n, the optional Fourier position encoding with $F = 8$ bands is

$$\phi(p_n) = \big[p_n,\ \sin(2^0 p_n), \cos(2^0 p_n), \ldots, \sin(2^{F-1} p_n), \cos(2^{F-1} p_n)\big] \in \mathbb{R}^{51}.$$

The **base** node input is $[\phi(p_n),\ v_n,\ m_n]$ ($51{+}15{+}1 = 67$ dims). The **physics-feature** branch appends geometry $g_n = [d_n,\ r_n] \in \mathbb{R}^4$ (App. F.5) for a 71-dim input.

F.5 Physics-feature geometry

For each point the nearest boundary point is $q_n^* = \arg\min_{q \in \mathcal{S}_{\text{airfoil}}} \|p_n - q\|_2$, giving

$$d_n = \|p_n - q_n^*\|_2, \qquad r_n = \frac{q_n^* - p_n}{\max(d_n, \epsilon)},$$

a smooth distance-and-direction signal that varies continuously away from the wall. Distances are computed with a chunked `cdist` (chunk size 4096) for memory safety, and r_n is zeroed where $d_n < \epsilon$.

F.6 Temporal attention head

Given a node embedding $h_n \in \mathbb{R}^D$, the head projects to $T_{\text{out}} \times d_t$, reshapes to a length-$T_{\text{out}}$ token sequence, adds a learnable temporal positional embedding, and applies 2 blocks of multi-head self-attention (4 heads) with a feed-forward sublayer Vaswani et al. (2017b):

$$z = \mathrm{LN}(z + \mathrm{MHSA}(z)), \qquad z = \mathrm{LN}(z + \mathrm{FFN}(z)).$$

This models dependencies across the T_{out} future steps per spatial node.

F.7 Spatial backbone options

A factory selects one of three interchangeable backbones; all consume node states $x \in \mathbb{R}^{N \times D}$, a neighbour table, relative coordinates, and distances, and are shape-compatible with the same temporal head and decoder.

GAT Veličković et al. (2018). Local multi-head attention with an edge-conditioned bias on the logits:

$$\alpha_{n,j,h} = \mathrm{softmax}_{j \in \mathcal{N}(n)}\Big(\tfrac{\langle W_q^h x_n, W_k^h x_j \rangle}{\sqrt{d_h}} + b_{\text{edge}}(n, j, h)\Big), \quad x_n \leftarrow \mathrm{LN}\big(x_n + \textstyle\sum_j \alpha_{n,j} W_v x_j\big),$$

followed by a residual feed-forward block.

MeshGraphNet Pfaff et al. (2021b). Residual edge and node updates with mean aggregation:

$$e_{n,j} \leftarrow \mathrm{LN}\big(e_{n,j} + \mathrm{MLP}_e([x_n, x_j, e_{n,j}])\big), \quad x_n \leftarrow \mathrm{LN}\big(x_n + \mathrm{MLP}_x([x_n,\ \tfrac{1}{k}\textstyle\sum_j e_{n,j}])\big).$$

Graph Transformer Dwivedi & Bresson (2021). Local attention where edge features modulate the value projection:

$$v_{n,j}^{(h)} = W_v^h x_j + W_e^h e_{n,j}, \qquad x_n \leftarrow \mathrm{LN}\big(x_n + \textstyle\sum_j \alpha_{n,j} v_{n,j}\big) \rightarrow \mathrm{LN}(x_n + \mathrm{FFN}(x_n)).$$

Edge features are encoded from $[\Delta p,\ d]$ by a shared edge MLP.

F.8 Decoder and no-slip projection

A pointwise decoder (Linear $\rightarrow$ GELU $\rightarrow$ Linear) maps each future latent token to a 3D delta $\Delta u_n(t)$. The prediction is residual, $\hat{u}_n(t) = \Delta u_n(t) + u_n(T_{\mathrm{in}} - 1)$, and the boundary condition is enforced by $\hat{u}_n(t) = 0$ for $n \in \mathcal{A}$.

F.9 Hard persistence fallback

The submission wrapper optionally substitutes persistence when the input norm exceeds a threshold *and* the mean frame-to-frame change is below a threshold; the output is the repeated last frame followed by no-slip projection. This conservative safeguard targets rare high-magnitude, low-dynamics regimes where extrapolation is unstable.

F.10 Deployment and member configuration

Each member is instantiated from its checkpoint `args` (backbone, hidden dimension, number of layers, heads, k, Fourier on/off). The physics-feature branch defaults to the MeshGraphNet backbone with hidden width 128, 8 layers, 4 heads, and $k = 16$. The base branch is trained for 300 epochs and the physics-feature branch for 100 epochs; both are frozen at inference. Runtime device resolution and boundary-index sanitisation are applied to prevent mixed-device and out-of-range failures.

G TransolverAR

TransolverAR adapts Transolver Wu et al. (2024) to autoregressive velocity prediction on irregular aerodynamic meshes. Slice-based physics attention aggregates nodes into a fixed number of learned slices, reducing attention cost while supporting variable mesh sizes.

The model receives mesh coordinates and the previous five velocity fields and predicts the next five steps autoregressively. It uses six Transolver blocks with hidden dimension 256, eight attention heads, and 64 learned slices.

Each training sample is uniformly subsampled to 20,000 nodes, while inference is performed directly on full meshes of about 100,000 nodes. Performance remains nearly unchanged, with nRMSE 0.1145 on the full mesh and 0.1164 on the subsampled mesh.

Training runs for 500 epochs using an 80/20 geometry-level split, AdamW, cosine annealing, and learning rate 2×10^{-4}. During the first 250 epochs, the probability of using the ground-truth previous state as input decreases linearly from 1 to 0, after which training uses only the model's own previous predictions. We found the training schedule mattered more than the architecture. Training the autoregressive rollout from scratch failed to converge, as the model never stabilised when fed its own noisy predictions early on. Pure teacher forcing was stable but suboptimal: it minimised single-step error while leaving the model fragile under the compounding errors of rollout evaluation. On the input side, conditioning on only the previous timestep was clearly worse than a longer window, so the final model relies on a 5-step input window. We also experimented with alternative optimizers such as Lion, but none matched AdamW.

Given more time, a promising direction would be to condition explicitly on the timestep, which the current model does not exploit. We would also revisit the uniform subsampling used for low-resolution training: inspecting Transolver's physical tokens in the first layer would reveal which

regions of the domain drive the prediction, and these could inform a more guided, physics-aware sampling scheme. A lightweight refinement stage fine-tuning the subsampled-mesh model onto the full mesh would further offer a cheap way to close the remaining resolution gap. Finally, a more sophisticated rollout objective along the lines of the pushforward trick, where several steps are predicted but only the last is supervised, should improve robustness while keeping training efficient.

H TRANSOLVERCORRECTOR

TransolverCorrector predicts future velocities as a time-conditioned residual on the last step. A Transolver++ backbone (Luo et al., 2025) ingests the five-step window $v(t_{0:4})$ and a query offset $\Delta t_j = t_j - t_4$, emitting an additive update to $v(t_j)$; time enters via a log-spaced sinusoidal encoder driving FiLM scale/shift at every norm. As global slice tokens smooth shear layers, an EdgeConv-V2 GNN corrector refines only the top 40% of points from a soft variance-based wake mask, co-trained jointly. No-slip is a hard mask on airfoil points. Dead ends: a GINO (Li et al., 2023) latent-grid operator overfit patches, generalized poorly, and was too costly; a single-step head could not extrapolate. The full-window FiLM flow-map was the biggest win, then the corrector. Ablation of losses, schedules, and the corrector gate remains future work. Training uses the full dataset (simulation-level val split, wake-weighted L_1, no extra data).

Provenance. The summary above was supplied by the author. The details below were reconstructed from the model code and README shipped with the submission in the competition repository.[13]

H.1 ARCHITECTURE

TransolverCorrector is a two-stage model: a global backbone that predicts the whole field, followed by a local corrector that refines only the points where the backbone is expected to be weakest.

Stage 1: Transolver++ backbone. The backbone is a Transolver++ (Luo et al., 2025), whose Physics-Attention with Eidetic States assigns points to a small set of learned physical slices and attends across the slices rather than across the $\sim$100k points, making the cost linear in the number of points. It is used as a *flow map* rather than a single-step predictor: it conditions on the full five-step input window and, for a query offset $\Delta t_j = t_j - t_4$, emits an additive residual on top of the last input frame $v(t_4)$. Time enters through a sinusoidal encoder (16 log-spaced frequencies) that drives FiLM (Perez et al., 2018) scale and shift parameters at every LayerNorm, so a single set of weights serves all five output steps and the model is queried once per step. The submitted configuration uses 3 blocks of width 192, 8 heads, 32 slices, and dropout 0.21.

Stage 2: wake corrector GNN. Because the backbone's slice tokens are global, they tend to smooth the sharp structures of the shear layer and the wake. A second network corrects them locally. A variance gate ranks points by the temporal variance of their input velocity and selects the top 40%, which concentrates on the unsteady wake and excludes both the no-slip wall (where the variance is zero) and the steady free stream. On this subset the corrector builds a k-NN graph (k=16) and runs 10 layers of EdgeConv-V2 message passing (Wang et al., 2019) with sum aggregation and edge features, predicting an additive correction to the backbone output. The two stages are co-trained jointly rather than staged.

Inputs and boundary condition. Beyond position and the five input velocity frames, each point carries a distance-to-airfoil feature, encoded as $\log(1 + d_{\min})$ to compress its dynamic range. The no-slip condition is enforced exactly, as a hard mask that zeroes the velocity at every airfoil point after both stages.

[13] github.com/gram-competition/iclr-2026/models/

Table 15: Submitted TransolverCorrector configuration.

Component	Hyperparameter	Value
Backbone	Hidden width	192
Backbone	Blocks	3
Backbone	Attention heads	8
Backbone	Slices	32
Backbone	Dropout	0.21
Backbone	Time encoder frequencies	16
Corrector	Type	EdgeConv V2
Corrector	Hidden width	192
Corrector	Layers	10
Corrector	k-NN neighbours	16
Corrector	Selected fraction	0.4 (top, by temporal variance)

H.2 Training details

Loss. A wake-weighted L_1 loss (α=1.0): the per-point L_1 error is weighted by the local velocity variance, which puts the optimisation pressure on the same unsteady regions that the corrector refines.

Optimisation. AdamW (Loshchilov & Hutter, 2019b) with learning rate 5×10^{-4} and weight decay 10^{-5}, on a cosine-annealing schedule (Loshchilov & Hutter, 2017), for roughly 400 epochs. Training used a batch size of 2 per GPU across 3 NVIDIA A100-80GB GPUs under DDP. The full dataset was used, with a simulation-level validation split and no external data.

Preprocessing and augmentation. No input normalisation: both velocity and position are consumed in raw physical units. Augmentation is by reflection across the y and z axes (50% probability each) and by variance-scaled additive noise (base standard deviation 0.005, scale 0.02), zeroed at airfoil points so the no-slip condition is never corrupted.

I gEGNO

Airflow prediction is fundamentally geometric: rotating or translating the coordinate system should not change the underlying physics, only how the velocity field is represented. gEGNO therefore models the flow with an $E(3)$-equivariant graph neural operator, enforcing consistency under rigid motions while allowing different flow regimes to communicate differently through learned per-edge gates adapted from EGNN edge inference (Satorras et al., 2022). gEGNO combines four EGNO blocks (Xu et al., 2024) with Fourier Neural Operator style temporal convolution over five input frames (Li et al., 2021) and a one-shot Reynolds (Reynolds, 1895) mean-residual velocity decoder. Temporal spectral convolution substantially outperformed standard 1-D convolutions, suggesting that wake and advection dynamics are poorly captured by purely local temporal operators. While equivariance provided a useful inductive bias and strong performance, the gains were smaller than expected, suggesting that the benchmark may contain preferred aerodynamic frames (Lawrence et al., 2025a).

I.1 Architecture overview

The submitted model follows the EGNO idea of stacking temporal Fourier layers over equivariant graph networks (Xu et al., 2024), but adapts it to the competition setting in three ways. First, the temporal axis is the real five-frame input history rather than replicated copies of a current state. Second, the spatial graph is a fixed k-NN connectivity pattern over the point cloud, so coordinates are not updated by the EGNN coordinate branch. Third, the decoder predicts all future frames in one shot as a residual around the temporal mean of the observed velocity window. This is an Eulerian-style formulation, since the point locations remain fixed while the velocity vectors attached to those points evolve.

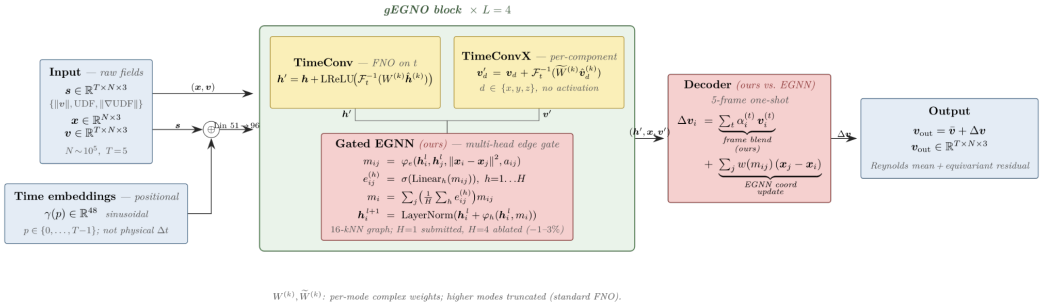

Figure 4: gEGNO uses five real input frames unlike EGNO. Blue boxes denote raw inputs and outputs, the green block is the repeated gEGNO block, and the red boxes mark the main adaptations over a plain EGNN, namely gated edge aggregation and the one-shot Reynolds mean-residual decoder.

Table 16: The final architecture favours a compact Eulerian design, with fixed point locations, gated local interactions, temporal spectral convolution, and residual prediction around the input mean.

Component	Submitted setting
Spatial connectivity	fixed, directed k-NN with $k = 16$
Coordinate Updates	disabled
EGNO blocks	4
Hidden width	96
Fourier modes	3 retained modes, $k \in \{0, 1, 2\}$
Gate Heads	1
Activation	SiLU
Normalization	LayerNorm after message-passing updates
Decoder	one-shot Reynolds mean-residual decoder
Boundary condition	hard no-slip mask of airfoil-surface nodes

Each block uses residual temporal Fourier updates on scalar hidden states and velocity vectors, followed by a residual gated EGNN update. We used SiLU activations in the encoder and MLPs (Elfwing et al., 2018), together with LayerNorm after message-passing updates, to make the four-block network easier to optimize. The architecture table lists the exact submitted settings, while the equations below describe where the residual connections enter.

I.2 Task and modelling assumptions

We initially approached the airflow prediction task through the lens of Sheaf Neural Networks (Hansen & Gebhart, 2020). The motivation was that flow around airfoils is not locally homogeneous, since boundary-layer regions, wake regions, and free-stream regions can obey different interaction patterns even when points are spatially close. Sheaf Neural Networks were attractive because they attach local feature spaces to graph elements and use restriction maps to define how neighbouring regions should communicate. However, scaling this idea to the competition setting was impractical since each simulation contains roughly 100k points, and learning or applying restriction-map transformations across a large neighbourhood graph would introduce substantial memory, compute, and implementation overhead.

We therefore reframed our approach around symmetries and constraints that could be exploited efficiently. Since the velocity field should transform consistently under rotations and translations of the coordinate system, we used an $E(3)$-equivariant graph network rather than an unconstrained point-cloud model. The graph itself was kept simple, with a fixed directed k-NN graph using $k = 16$. This gave a stable local geometric scaffold while avoiding the overhead of dynamically learning graph structure or sheaf maps. To retain some of the original heterogeneous-communication intuition, we

Table 17: The inputs separate invariant scalars from equivariant vectors, which motivates using scalar MLP features for local compatibility while preserving vector-valued velocity behaviour under rigid motions.

Inputs	Symmetry Type	Role in gEGNO
Point position $x_i \in \mathbb{R}^3$	equivariant	Defines geometry and k-NN connectivity
Relative displacement $x_i - x_j$	equivariant	Direction for vector-valued messages
Squared distance $\|x_i - x_j\|_2^2$	invariant	Edge scalar for message MLPs
Velocity $v_i^{(t)} \in \mathbb{R}^3$	equivariant	Predicted physical field
Velocity projection $\langle v_i^{(t)}, \hat{r}_{ij} \rangle$	invariant	Local flow-direction edge feature
Truncated distance $\tilde{d}(x_i)$	invariant	Distance-to-airfoil scalar feature
Surface-normal direction $\hat{n}(x_i)$	equivariant	Boundary-direction signal
Time index t	invariant	Encoded with sinusoidal embeddings

adapted the EGNN edge-inference mechanism into unnormalised per-edge sigmoid gates (Satorras et al., 2022). Unlike softmax graph attention (Veličković et al., 2018), these gates do not force neighbouring edges to compete for fixed attention mass, so each existing neighbour can be retained or suppressed independently.

I.3 Point-cloud preprocessing and spatial connectivity

For each point x_i, let $s^\star(x_i)$ denote the nearest airfoil-surface point,

$$s^\star(x_i) = \arg\min_{s \in S} \|x_i - s\|_2. \tag{7}$$

where $x_i \in \mathbb{R}^3$ is the coordinate of point i, S is the set of airfoil-surface points, and $s^\star(x_i)$ is the nearest surface point to x_i. We then compute a truncated unsigned distance feature

$$\tilde{d}(x_i) = \min\left(\|x_i - s^\star(x_i)\|_2, d_{\max}\right), \qquad d_{\max} = 0.5, \tag{8}$$

where $\tilde{d}(x_i)$ is the truncated unsigned distance feature and $d_{\max}$ is the truncation threshold. We also compute an inward surface-normal direction

$$\hat{n}(x_i) = \frac{s^\star(x_i) - x_i}{\|s^\star(x_i) - x_i\|_2}, \tag{9}$$

where $\hat{n}(x_i)$ is the normalized direction from point x_i to its nearest airfoil-surface point.

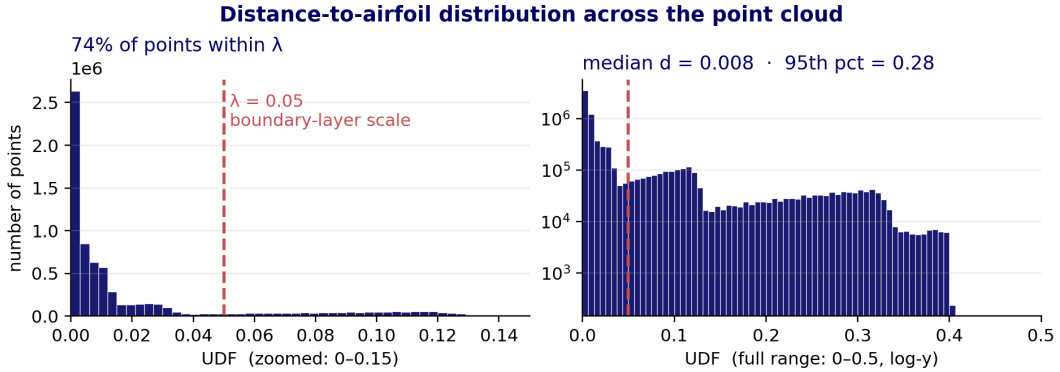

Figure 5: The distance-to-airfoil distribution is highly concentrated near the surface, so changing neighbourhood density mainly affects boundary-layer regions rather than the whole point cloud.

The submitted model uses a directed fixed k-NN graph with $k = 16$, recomputed per sample in Euclidean 3D space using `scipy.spatial.cKDTree` Maneewongvatana & Mount (1999). This

connectivity is invariant to rigid translations and rotations because it depends only on Euclidean distances. Edge features include invariant distances and velocity projections onto the local edge direction.

We also ablated a spatially varying, UDF-adaptive k-NN rule designed to allocate more neighbours near the airfoil and fewer neighbours in the far field,

$$k_i = k_{\text{far}} + (k_{\text{near}} - k_{\text{far}}) \exp\left(-\tilde{d}(x_i)/\lambda\right). \tag{10}$$

where k_i is the neighbour count used for point i, $k_{\text{near}} = 32$ is the near-surface neighbour count, $k_{\text{far}} = 8$ is the far-field neighbour count, and $\lambda = 0.05$ controls how quickly the neighbour count decays with distance from the airfoil surface.

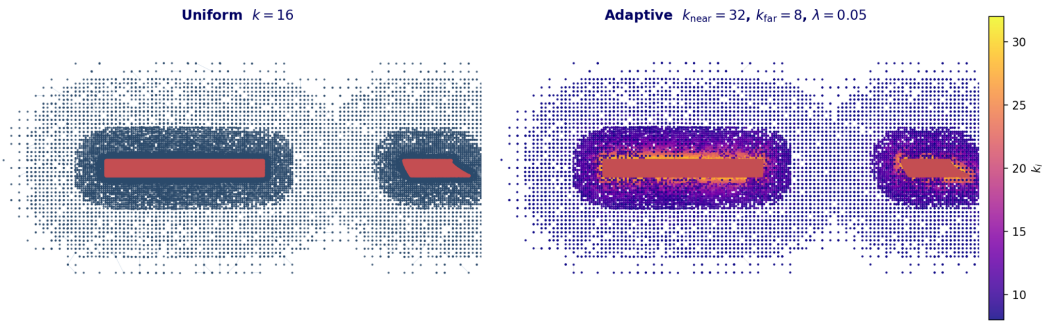

Figure 6: The adaptive rule concentrates connectivity near the airfoil surface, which matches the boundary-layer motivation but produced worse test error than the simpler fixed k-NN graph in our ablations.

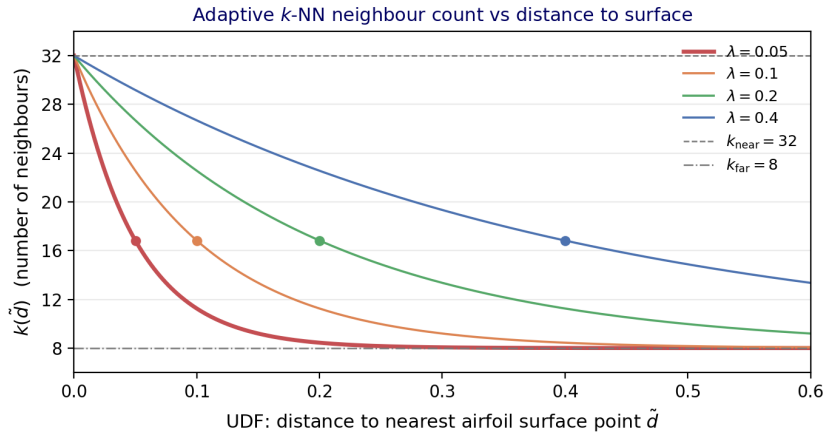

Figure 7: The schedule with $\lambda = 0.05$ rapidly decays from near-surface to far-field connectivity, making it an aggressive boundary-focused graph rather than a smooth global reweighting.

I.4 Spectral Convolution updates and gated EGNN layer

Temporal evolution is handled by a Fourier Neural Operator (FNO)-style convolution along the five-frame input axis (Li et al., 2021). The operation transforms each temporal signal into Fourier space, multiplies the retained modes $k \in \{0, 1, 2\}$ by learned spectral weights, truncates higher modes, and

transforms the result back to the time domain. For scalar hidden states, the residual update is

$$h' = h + \mathrm{LeakyReLU}\left(\mathcal{F}_t^{-1}\left(W^{(k)}\hat{h}^{(k)}\right)\right). \tag{11}$$

where h is the hidden scalar feature sequence, h' is the updated sequence, $\hat{h}^{(k)}$ is the Fourier coefficient at temporal mode k, $W^{(k)}$ is the learned spectral weight for mode k, and $\mathcal{F}_t^{-1}$ is the inverse Fourier transform along the time axis.

For velocity vectors, the same temporal spectral idea is applied component-wise with shared scalar spectral weights,

$$v_d' = v_d + \mathcal{F}_t^{-1}\left(\widetilde{W}^{(k)}\hat{v}_d^{(k)}\right), \qquad d \in \{x, y, z\}. \tag{12}$$

where v_d is one spatial component of the velocity sequence, v_d' is its updated value, $\hat{v}_d^{(k)}$ is the Fourier coefficient at mode k, and $\widetilde{W}^{(k)}$ is the learned scalar spectral weight shared across spatial components. This preserves equivariance because rotations commute with scalar weighting of vector components.

For each edge (i, j), the EGNN message is computed from node states, squared inter-point distance, and edge attributes,

$$m_{ij} = \phi_e\big(h_i, h_j, \|x_i - x_j\|_2^2, a_{ij}\big), \tag{13}$$

$$\tilde{e}_{ij} = \sigma(\mathrm{Linear}(m_{ij})), \tag{14}$$

$$m_i = \sum_{j \in \mathcal{N}(i)} \tilde{e}_{ij} m_{ij}, \tag{15}$$

$$h_i^{\ell+1} = \mathrm{LayerNorm}\big(h_i^{\ell} + \phi_h(h_i^{\ell}, m_i)\big). \tag{16}$$

where m_{ij} is the message from neighbour j to node i, ϕ_e is the edge MLP, h_i and h_j are hidden node states, a_{ij} is the edge-attribute vector, $\tilde{e}_{ij}$ is the learned sigmoid gate, σ is the logistic sigmoid, $\mathcal{N}(i)$ is the neighbourhood of node i, m_i is the aggregated gated message, ϕ_h is the node-update MLP, and $h_i^{\ell+1}$ is the updated hidden state after residual addition and LayerNorm. In the original EGNN edge-inference formulation, $\tilde{e}_{ij}$ can be interpreted as a soft edge-existence indicator (Satorras et al., 2022). In gEGNO, the graph topology is fixed, so $\tilde{e}_{ij}$ instead modulates how strongly each existing neighbour contributes.

I.5 Decoder and prediction target

The decoder predicts a velocity increment as a sum of a learned per-frame blend of input velocities and an EGNN-style vector update,

$$\Delta v_i = \sum_{t=0}^{T-1} \alpha_i^{(t)} v_i^{(t)} + \sum_{j \in \mathcal{N}(i)} w(m_{ij})(x_j - x_i). \tag{17}$$

where Δv_i is the predicted velocity increment at point i, $T = 5$ is the number of input frames, $\alpha_i^{(t)}$ is the learned blend weight for input frame t, $v_i^{(t)}$ is the input velocity at point i and time t, and $w(m_{ij})$ is a learned scalar weight applied to the relative displacement $x_j - x_i$.

The model predicts a residual relative to the temporal mean of the input window, following the classical Reynolds mean/fluctuation decomposition (Reynolds, 1895),

$$\bar{v}_{\mathrm{in}}(x_i) = \frac{1}{T}\sum_{t'=0}^{T-1} v_{\mathrm{in}}(x_i, t'), \qquad \hat{v}_{\mathrm{out}}(x_i, t) = \bar{v}_{\mathrm{in}}(x_i) + \Delta v(x_i, t). \tag{18}$$

where $\bar{v}_{\mathrm{in}}(x_i)$ is the temporal mean of the input velocity at point x_i, $v_{\mathrm{in}}(x_i, t')$ is the input velocity at time index t', $\Delta v(x_i, t)$ is the predicted residual for output time t, and $\hat{v}_{\mathrm{out}}(x_i, t)$ is the predicted output velocity.

The loss is ordinary per-point MSE over output frames,

$$\mathcal{L} = \frac{1}{TN}\sum_{t=0}^{T-1}\sum_{i=1}^{N} \|\hat{v}_{\mathrm{out}}(x_i, t) - v_{\mathrm{out}}(x_i, t)\|_2^2. \tag{19}$$

Table 18: The final training setup used a single model and conservative optimization settings, so the reported gains come from the architecture rather than ensembling.

Training choice	Final setting
Training data	full training split
Validation/test split	95/2.5/2.5 by simulation, stratified by geometry
Optimizer	AdamW
Learning Rate	$2 \cdot 10^{-3}$
Weight decay	10^{-2}
Warmup	15 epochs
Scheduler	Cosine A nnealing
Batch size	1
Gradient accumulation	8 steps
Maximum epochs	120
Early Stopping	patience 30 on validation $L2$
Time Encoding	Sinusoidal embeddings for indices $0, \ldots, T-1$
Seed	42
Ensembling	none
Final GPU	NVIDIA B200, approximately 15 h 37 min

where $\mathcal{L}$ is the training loss, N is the number of points, and $v_{\text{out}}(x_i, t)$ is the ground-truth output velocity. Predictions at airfoil-surface nodes are hard-masked to respect the no-slip condition.

I.6 Training and validation details

The final submission used the full training split, while ablations used 15% of the training data. The final run used a geometry-stratified 95/2.5/2.5 split by simulation and no ensembling. Ablations were intentionally cheaper and used a 70/15/15 split with 30 epochs per run.

Timestep indices were encoded using sinusoidal positional embeddings (Vaswani et al., 2017a). The ablation runs used an NVIDIA L40S, while the final submission used one NVIDIA B200 GPU.

I.7 Prototyping lessons and model selection

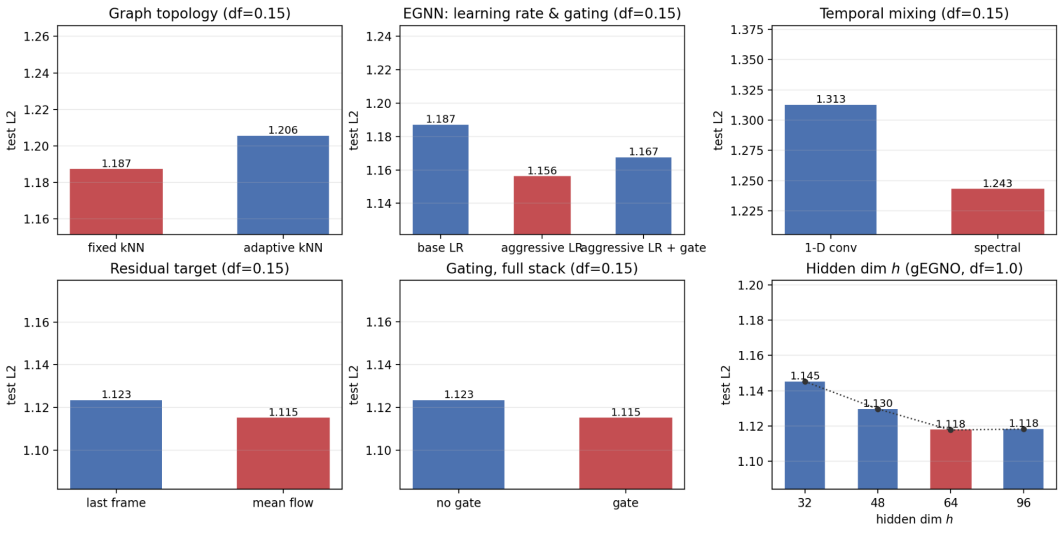

Figure 8: The ablation grid explains why the final model stayed relatively simple. Fixed k-NN connectivity, temporal spectral convolution, mean-residual prediction, and one gate head outperformed several physically plausible or higher-capacity alternatives.

The strongest lesson from prototyping was that physically motivated complexity was not automatically better. The UDF-adaptive k-NN graph was worse than fixed k-NN with $k = 16$, despite allocating more neighbours near the boundary layer. Spectral convolution applied temporally was substantially better than a standard 1-D temporal convolution, which missed wake and advection dynamics in our ablation. Predicting the Reynolds mean-residual was slightly better than predicting a last-frame residual. Increasing hidden width improved validation $L2$ but did not meaningfully improve test $L2$, and additional gate heads introduced higher variance. We did not increase the depth beyond four EGNO blocks because deeper local message-passing stacks can suffer from smoothing and signal dilution, while graph-scaling behaviour is often task- and data-dependent rather than monotonic (Liu et al., 2024). These results led us to keep the final model relatively compact, with fixed k-NN connectivity, one gate head, temporal spectral convolution, and a mean-residual decoder.

The most important implementation failure concerned the decoder. We initially zero-initialized the two-layer decoder heads to force $\Delta v = 0$, which gave stable initial predictions. In practice, this created a dead-gradient regime in the velocity gate. Using the default random PyTorch initialization for the decoder MLPs restored gradient flow, despite starting from a higher initial loss. This was a useful reminder that physically conservative initialization can still be optimization-unfriendly.

I.8 Results and qualitative behaviour

The submitted model achieved test $L2 = 1.118$, compared with the organiser MLP baseline at 3.16 and a baseline that simply copies the last observed input frame at 1.68. This corresponds to about a 65% reduction relative to the MLP baseline and about a 33% reduction relative to copying the last observed frame. Per-frame error increases across the five-frame rollout but remains lower than the compared baselines at every timestep.

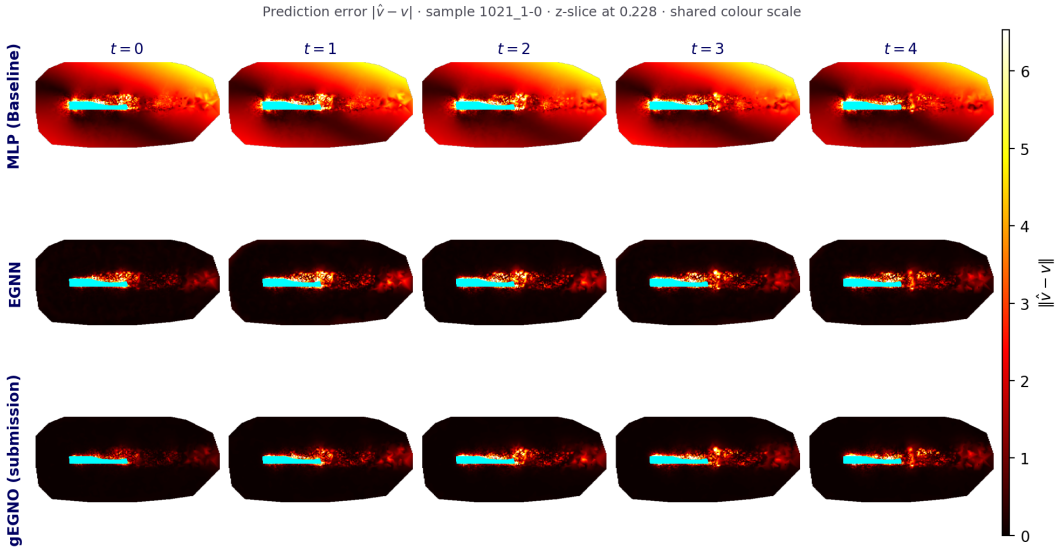

Figure 9: The qualitative error maps show that gEGNO reduces broad field error and concentrates the remaining error into the wake, whereas the MLP baseline saturates across much of the domain and the plain EGNN leaves stronger scattered wake noise.

I.9 Limitations and future work

A limitation of gEGNO is that message passing is local on fixed k-NN connectivity. Boundary-layer effects and wake structures involve both short-range geometric interactions and longer-range flow dependencies, which may be inefficient to propagate through purely local aggregation. Future work could therefore explore U-Net-style skip connections between encoder and decoder layers, multi-resolution message passing, dynamic graphs, or hybrid equivariant-attention architectures (Fuchs

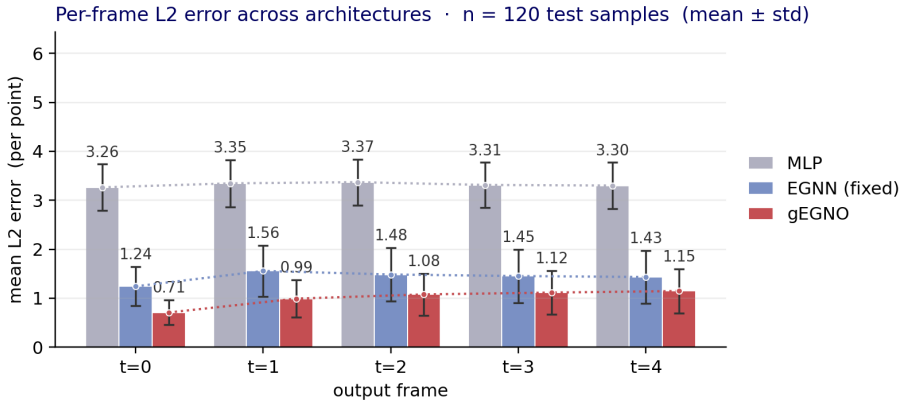

Figure 10: Per-frame error increases with prediction horizon, but gEGNO remains below the MLP and plain EGNN baselines across all five output frames.

et al., 2020; Liao & Smidt, 2023) to capture complex wake dynamics while preserving local geometric structure.

A second limitation concerns the equivariance assumption itself. While $E(3)$-equivariance constrains the learned velocity predictor to respect rotations and translations, the dataset distribution may still contain preferred frames (Lawrence et al., 2025a). In the warped IFW setting, airfoil orientation, freestream direction, and boundary conditions likely define canonical aerodynamic coordinates, so arbitrary global rotations may not be equally likely or physically equivalent. Future work could test this using a two-sample classifier for symmetry breaking and explore frame-aware features or symmetry-breaking positional encodings, such as SymPE (Lawrence et al., 2025b), to balance local equivariance with global coordinate information.

J FiniteGraphV4

FiniteGraphV4 is a graph forecaster built on the observation that a neighbourhood in a convecting flow is not isotropic: a point is driven by what lies upstream of it and drives what lies downstream. Instead of a single symmetric stencil, each centre point carries three separate graphs, isotropic, upstream, and downstream, and each is extended to a second hop that preserves flow orientation, with the second-hop neighbours drawn from an oversized candidate pool to loosen the dependence on mesh resolution. A temporal encoder reconstructs the five velocity snapshots from delta channels and encodes them with 1D convolutions; messages then flow inward from the second hop to the centre through two message-passing blocks (Brandstetter et al., 2022) that pool neighbours by attention, maximum, and variance, the last making local turbulence intensity directly visible to the network. The three directions are merged by a gated fusion that passes on not only their weighted consensus but also their disagreement. Points carry the standard vortex diagnostics, vorticity, Q-criterion, and strain magnitude (Pope, 2000), computed from a local least-squares velocity gradient, and edges carry Fourier positional encodings (Tancik et al., 2020a) non-dimensionalised by the local extent. The network predicts residuals on the last input frame, and no-slip is enforced by zeroing the airfoil points. Training uses a weighted $L_1 + L_2$ loss with a near-wall upweight and low-weight soft-divergence and Navier–Stokes momentum penalties.

Provenance. No write-up was supplied with this submission. The summary above and the details below were reconstructed from the model code and README shipped with the submission in the competition repository,[14] and the author's own hedges about which mechanisms did what are preserved.

[14] github.com/gram-competition/iclr-2026/models/

J.1 ARCHITECTURE

Directional two-hop neighbourhoods. For every non-surface centre node the model builds three directional graphs, isotropic, upstream, and downstream, each with k_1 first-hop neighbours. For every first-hop neighbour it then gathers k_2 *directional* second-hop neighbours drawn from a larger k_{pool} k-NN candidate pool, so each (centre, direction) pair carries $k_1 \times k_2$ two-hop paths whose flow orientation is preserved at hop 2 as well as at hop 1. Drawing the second hop from an oversized pool was intended as a regulariser, to make the stencil less dependent on the local mesh resolution; the author records this as the design intent rather than as a measured effect. Neighbour search uses a k-d tree (Maneewongvatana & Mount, 1999) (SciPy's `cKDTree`); there are no custom CUDA kernels, and the model runs on CPU or GPU.

Temporal node encoder. The 26-channel per-point input is first passed through a temporal node encoder that reconstructs the five raw velocity snapshots from the delta channels, encodes them with two `Conv1d` blocks, and fuses the resulting temporal summary with a static-feature projection through a residual LayerNorm (Ba et al., 2016). The encoded latent drives all downstream message passing.

Message passing. Within each direction, messages flow hop 2 $\rightarrow$ hop 1 $\rightarrow$ centre through two message-passing blocks (Brandstetter et al., 2022). Each block combines a centre-derived multi-head attention query (Vaswani et al., 2017b) with a three-way pool over the neighbourhood: attention, max, and *variance*. The variance channel is what makes local turbulence intensity directly visible to the subsequent MLP instead of being averaged away.

Gated cross-direction fusion. The isotropic, upstream, and downstream latents are combined by a gated fusion module: a centre-conditioned softmax produces three branch weights, and the fused vector concatenates the weighted summary with the inter-branch spread (variance) and the inter-branch maxima, so the head sees not only the consensus across directions but also their disagreement. An MLP head decodes this into $5 \times 3 = 15$ outputs.

Features. The 26 input channels are: four velocity deltas against $v[0]$ (12 channels), v_{t_4} (3), wall distance (1), signed wall distance (1), normalised position (3), an airfoil indicator (1), and, at t_4, vorticity (3), the Q-criterion (1), and strain magnitude (1). The last three are computed from the local k-NN least-squares velocity-gradient tensor at t_4, which gives the network the standard vortex-identification diagnostics of the flow (Pope, 2000) as explicit inputs rather than asking it to infer them.

The 39-channel edge features carry the relative position and distance; Fourier positional encodings (Tancik et al., 2020a) (3 wavenumbers $\times$ 3 axes $\times$ $\sin$ / $\cos$), non-dimensionalised by the per-chunk extent L_{ref} to counter the MLP's spectral bias on raw Cartesian inputs (Rahaman et al., 2019); wall distance (neighbour and delta); the flow-aligned and perpendicular components of the displacement; and neighbour-versus-centre differences in speed, velocity, vorticity, Q-criterion, strain magnitude, and acceleration.

Output and boundary condition. The network emits five normalised residuals $r_k = v[5{+}k] - v_{t_4}$. The wrapper adds v_{t_4} back and zeroes the velocity at the airfoil indices, so no-slip holds exactly.

J.2 TRAINING DETAILS

Split and targets. A simulation-level split, stratified by geometry prefix. The targets are the residuals $r_k = v[5 + k] - v[4]$ for $k = 0, \ldots, 4$.

Loss. A weighted $L_1 + L_2$ on the normalised residual with a mild near-wall upweight. Differentiable soft-divergence and Navier–Stokes momentum penalties with the physical properties of air ($\rho = 1.225$, $\mu = 1.81 \times 10^{-5}$) are implemented and were mixed in at low weight ($\sim$0.01) during training (Raissi et al., 2019).

Optimisation and normalisation. AdamW (Loshchilov & Hutter, 2019b) with linear warmup followed by cosine decay (Loshchilov & Hutter, 2017). Per-channel mean and standard deviation

Table 19: Default FiniteGraphV4 configuration (matching the training repository). The checkpoint carries its own `model_config`, which is used at load time, so alternative hyperparameters are picked up automatically.

Hyperparameter	Value
Input channels	26
Hidden / latent width	192 / 192
First-hop neighbours k_1	24
Second-hop neighbours k_2	12
Attention heads	4
Output heads $\times$ channels	5×3
Temporal encoder width	96
Dropout	0.05
Shared weights	no

are fit on the training split, with the airfoil-indicator channel left unscaled; these statistics travel with the checkpoint.

K AB-UPT

The AB-UPT submission is a scaled-down version of the Anchored-Branched Universal Physics Transformer (Alkin et al., 2025). Input points are embedded from position and its Fourier features, the five velocity frames, the start time, and an "on-airfoil" flag. A set of *supernodes* on the airfoil and in the volume (biased to high-variance wake) is filled by k-NN message passing, processed by a branched transformer whose surface/volume streams couple via cross-attention every second block, and decoded by a Perceiver into five per-point velocity deltas; no-slip is enforced by zeroing wall points. Variance-targeting supernodes helped slightly, and scaling capacity or supernode count barely moved the metric, so the model was kept small (5.3M parameters). Trained on reflection-doubled data with EMA and bf16, it reaches relative L^2 error 0.0838. With more time, the divergence-free decoder would be restored and latent assignment improved, rather than scaling up.

AB-UPT is a deliberately small adaptation of the Anchored-Branched Universal Physics Transformer (Alkin et al., 2025). The original model targets industrial automotive meshes with tens of millions of cells. Our variant keeps its three defining ideas (anchored supernodes, branched surface/volume processing, and an anchored-field decoder) but is sized to train end-to-end on a single 24 GB GPU for the warped Imperial-Front-Wing task.

K.1 Architecture

The forward pass has six stages.

Per-point embedding. Each of the N points is embedded from its position, a Fourier positional encoding of that position (8 frequencies, up to 128), the five input velocity vectors (15 values), the start time, and a binary wall flag, through a two-layer MLP to a d=192 feature.

Supernode sampling. Rather than attend over all N points, we sample a fixed set of M *supernodes* per sample: 128 on the airfoil surface and, in the volume, 384 "high-variance" and 128 "far-field" nodes. We rank volume points by the temporal variance of the input velocity (summed over components), which is zero at the no-slip wall and in the steady free stream and peaks in the unsteady near-surface region. Crucially, we do not take the highest-variance points outright: we form a candidate pool from the top 20% of volume points by this score (`wake_pool_frac`=0.2) and draw the 384 high-variance supernodes *uniformly at random* from within it, with the 128 far-field nodes drawn uniformly from the remaining 80%. This draw is resampled on every forward pass, acting as a regulariser, so the model sees a fresh soft sample of the high-variance region at each step rather than a fixed top-k set. This concentrates capacity where the prediction is hard while keeping the anchor set stochastic.

Message passing into supernodes. Each supernode gathers its k=16 nearest neighbours (by position) and mean-pools relative-position-encoded messages from them. The k-NN candidate pool is restricted to volume points: the wall carries $v \equiv 0$ for all input frames, so including surface points would dilute the pooled message; wall location is already supplied through the per-point wall flag and through the surface branch. A learned type embedding (surface / high-variance / far) is added to each supernode latent.

Branched approximator. The M latents are split into a surface stream and a volume stream that run through 12 transformer blocks. Self-attention and the FFN share weights across the two streams; every second block adds a cross-branch attention step so surface and volume tokens exchange information. Keeping the streams separate lets surface tokens specialise on the boundary without being swamped by the much larger volume.

Anchored-field (Perceiver) decoder. The decoder follows Perceiver IO (Jaegle et al., 2022), whose defining mechanism is a query-based readout from a fixed latent set: all N point embeddings act as queries that cross-attend (two decoder blocks) to the M processed supernode latents, turning the compressed latent simulation back into a per-point field at arbitrary query resolution.

Output head and boundary condition. A final MLP emits five per-point velocity *deltas* that are added to the last input frame (the network predicts the change rather than the absolute field). The no-slip condition is then enforced exactly, by construction, by hard-zeroing the velocity at every airfoil-surface point.

The submitted configuration has $5{,}294{,}627$ parameters. Hyperparameters are listed in Table 20.

Table 20: Submitted AB-UPT configuration.

Hyperparameter	Value
Hidden width d	192
Attention heads	4
Approximator blocks	12
Cross-branch attention	every 2nd block
Perceiver decoder blocks	2
FFN multiplier	2
Surface / high-variance / far supernodes	128 / 384 / 128
Wake candidate-pool fraction	0.2
Encoder k-NN neighbours	16
Fourier frequencies	8 (up to 128)
Parameters	5.29M

K.2 What did not work / what we converged on

There was no single breakthrough. The submitted model is the result of a handful of small, mostly local corrections to the first working prototype. The clearest finding was a negative one: *where* we placed supernodes mattered a little, but *how many* we used, and the overall model capacity, mattered surprisingly little. The points worth recording, in roughly the order we hit them:

- **Pooling over all points (including the wall).** The first encoder ran k-NN over every point. Because the airfoil surface has $v \equiv 0$ across all input frames, surface neighbours contributed near-constant, information-poor messages that diluted the pooled supernode latent. Restricting the k-NN candidate pool to volume points removed that dilution; the wall is still represented, through the per-point wall flag and the dedicated surface branch.
- **Uniform vs. variance-based supernode placement.** The first prototype sampled volume supernodes *uniformly at random*. One might expect uniform sampling to waste most supernodes on the steady free stream, but the training points are themselves distributed densely near the airfoil and only sparsely away from it. Uniform selection therefore already

places most supernodes near the surface; what it does *not* do is distinguish the steady near-surface regions (attached flow) from the unsteady ones. We switched to ranking volume points by the temporal variance of their input velocity and drawing most volume supernodes from this high-variance set. This refines the placement *within* the surface-dense cloud, concentrating supernodes on the dynamically active regions where the last-frame baseline is wrong and away from the steady regions (free stream and attached flow alike) where it is already almost correct. At a *matched* budget (128 surface + 384 volume supernodes, d=192, 6 blocks; only the volume *selection* changes), this lowered the internal validation loss by $\sim$2%, a small gain from placement alone.

- **Scaling model capacity barely helped.** Widening the model from d=192 to 256 and deepening it from 6 to 8 approximator blocks (a $2.25\times$ parameter increase, from 2,765,583 to 6,218,255 parameters) moved our internal validation metric only from 1.0499 to 1.0291 ($\sim$2%). For that reason, the submitted model went *back* to the narrow d=192 width (with 12 blocks, 5.3M parameters) rather than the wider variant.

- **More supernodes did not help either.** Increasing the "high-variance" supernode budget from 256 to 384 (volume nodes $384 \rightarrow 512$) produced no clear gain on the internal split. Together with the capacity result, this is why the final model uses a deliberately small, fixed supernode budget.

- **Low-frequency-only positional encoding and unnormalised inputs.** Adding higher Fourier frequencies and standardising positions/velocities with dataset statistics both gave small but consistent improvements and stabilised training.

The internal-metric figures above come from two different evaluation setups (the capacity comparison and the supernode-budget comparison were logged separately, on the internal validation split), so only the deltas *within* each comparison are meaningful, and none of these numbers is the official competition metric. In short, we adapted the AB-UPT architecture (Alkin et al., 2025) to this task and used prototyping to find a compact configuration that captures most of the attainable accuracy: additional width, depth, and supernodes bought only marginal gains, so we deliberately kept the model small (5.3M parameters), with variance-targeted supernodes and a volume-only encoder.

K.3 What we would improve next

Restore the divergence-free output head. The published AB-UPT decodes a vector potential so that the predicted velocity field is divergence-free by construction. We dropped this for simplicity; reinstating it is the most principled next step and would build in a physical constraint we currently ignore.

A better latent assignment, not a bigger one. All cross-point interaction and all learned dynamics happen on the M=640 supernode latents, and the k-NN encoder ingests only each supernode's 16 nearest points, so roughly 90% of the $\sim$100k points never enter the latent "simulation." In the decoder, queries are conditionally independent given the anchors—each point predicts from its own input features plus the shared latents, with no point-to-point interaction—so those 90% rely entirely on this shared context. Notably, the original AB-UPT (Alkin et al., 2025) reports that accuracy *saturates* in the number of anchors, which matches our own finding that adding supernodes did little: the headroom is not in the latent *count*. Yet our submission's error sits well above the best latent-based entry in the competition (0.0838 vs. 0.0638 relative L^2), so substantial headroom clearly remains *within* the latent paradigm, however, in how the latents are assigned and read, not in their number. The AB-UPT paper points to levers for using the latents better than we did. (i) *A richer encoder*: AB-UPT pools geometry with a radius-based graph-neural-operator, whereas we take a plain mean over the 16 nearest points. (ii) *Smarter anchor placement*: AB-UPT notes that anchor and query tokens may come from entirely different sets, suggesting a structured or geometry-informed anchor set in place of our fixed variance heuristic. (iii) *Relaxing query independence* to allow limited local point-to-point interaction, at higher cost. These, rather than a larger transformer, are where we would look next.

K.4 TRAINING DETAILS

Data and split. Training used the official warped-IFW dataset. We did not hold out an additional validation split beyond a fixed internal train/test split used to monitor convergence. The training set was doubled by a y-mirror augmentation: for each training sample we add a copy reflected across the $y{=}0$ plane, negating the y component of the point positions and of the input and output velocities (time, pressure, and airfoil indices unchanged). This gives 1290 training and 165 internal-test samples (the test split is left un-mirrored); no external data was used.

Optimisation. Adam with learning rate 5×10^{-4} on a cosine schedule decaying to 0 over 600 epochs, batch size 8, mixed-precision (bf16) autocast throughout. We track an exponential moving average of the weights (decay 0.999, 100-step warmup) and submit the EMA weights. The supernode set is resampled on every forward pass, which acts as an additional regulariser.

Loss. The per-point Euclidean norm of the velocity error, averaged over the five output steps and over all non-wall points (wall points are excluded since they are fixed to zero). This mirrors the structure of the competition metric (relative L^2 error of the 3D velocity field).

Hardware. A single NVIDIA RTX 4090 GPU with 24 GB.

L AIRFORMER

AirFormer's main contribution is an anchor-based design for 100k-point airfoil meshes: it samples 4096 anchor points and applies 12 layers of Transolver-style slice attention Wu et al. (2024), where tokens are soft-routed into 128 learnable physical-state slices with linear-time attention. Per-point output is recovered via inverse-distance softmax readout from the 8 nearest anchors, and five independent decoder MLPs produce residual velocity deltas with no-slip enforcement. Earlier designs that either downsampled the mesh or applied slice attention to all 100k tokens both failed. The former destroyed local correlations, the latter caused slices to degenerate into global means. The anchor-based design maintains a healthy point-to-slice ratio while preserving full-resolution output. The model was trained on 689 samples (121 val) at full resolution using AdamW with cosine scheduling, achieving a best validation L2 of 1.0541.

L.1 ARCHITECTURE DETAILS

AirFormer is built on three stages: per-point encoding, anchor-level slice attention, and per-point readout with decoding.

- **Per-point encoding**: Each mesh point receives a 67-dimensional input feature: 51 dimensions from multi-scale Fourier positional encoding with 8 frequency bands (Tancik et al. (2020a)), 15 from the flattened 5-step velocity history, and 1 binary surface mask. A 2-layer MLP with GELU and LayerNorm projects this into a 128-dimensional local feature, which is retained as a skip connection to the output stage.
- **Stratified anchor selection**: 4096 anchor indices are drawn with a 50/50 split between airfoil surface points and field points, motivated by stratified sampling principles from point cloud learning (Lai et al. (2022), Qian et al. (2022)) and the observation that boundary-layer regions concentrate prediction error. Each anchor is seeded by mean-pooling the local features of its 16 nearest mesh points found via a single `scipy.spatial.cKDTree` query, then lifted to the 256-dim anchor working space by a 2-layer MLP.
- **Physics Attention backbone**: 12 pre-norm Transolver blocks (Wu et al., ICML 2024) process the 4096 anchor tokens. Each block performs slice attention: anchors are softly assigned to $G = 128$ learnable physical-state slices via a softmax over learned slice projections. Attention is computed between the 128 slice tokens (16 heads, 16 dims/head), and results are broadcast back to anchors through the same soft routing. A feed-forward network with $4\times$ expansion follows each attention layer. Gradient checkpointing is applied to all blocks to fit within 16 GB of GPU memory.

Table 21: AirFormer model specifications (left) and training configuration (right).

Hyperparameter	Value	Setting	Value
Per-point feature dim	128	GPU	1× NVIDIA Tesla T4 (16 GB)
Anchor feature dim	256	Precision	AMP (float16)
Anchor count M	4,096	Optimiser	AdamW (weight decay 10^{-4})
Physics Attention blocks	12	Base learning rate	2×10^{-4}
Attention heads	16	LR schedule	Cosine decay to 10^{-6}, 2-epoch warmup
Head dim	16	Gradient clipping	Max-norm 1.0
Slices per block G	128	Batch size	4
Surface anchor fraction	0.50	Augmentations	None
Anchor seeding neighbors	16	Train / Val split	689 / 121 (85/15)
Per-point readout neighbours	8	Epochs	25
Fourier frequency bands	8	Training time	∼155 minutes
Dropout	0.05		
Total parameters	9,676,880		

- **Distance-weighted readout**: Each of the 100k mesh points queries its 8 nearest anchors (a second one-time `cKDTree` lookup) and pools their features with an inverse-distance softmax $w_i = \text{softmax}_k(-\tau \cdot d_i)$, where τ is a single learned positive scalar following the inverse distance weighting. The pooled 256-dim anchor feature is concatenated with the 128-dim per-point skip feature and projected through a fusion MLP to produce the final per-point representation.
- **Per-step temporal decoding**: Five independent 3-layer decoder MLPs, each preceded by a learnable step-specific bias vector, produce velocity deltas $(N, 5, 3)$. The final prediction adds the last observed velocity and hard-sets airfoil-surface velocities to zero for exact no-slip enforcement.

L.2 Model Specifications and Training Configuration

Table 21 presents the full architectural hyperparameters and training configuration of AirFormer. The model stacks 12 Physics Attention blocks over 4,096 stratified anchors, totaling approximately 9.7M trainable parameters. All experiments are conducted on a single NVIDIA Tesla T4 GPU with mixed-precision training and complete in under three hours.

L.3 Loss Functions

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{L2}} + 0.1 \cdot \mathcal{L}_{\text{L1}} + 0.1 \cdot \mathcal{L}_{\text{mag}} + 0.5 \cdot \mathcal{L}_{\text{temp}}$$

where $\mathcal{L}_{\text{MSE}}$ is mean-squared error, $\mathcal{L}_{\text{L2}}$ is the differentiable competition metric (mean pointwise L2 velocity error), $\mathcal{L}_{\text{L1}}$ is mean absolute error for outlier robustness, $\mathcal{L}_{\text{mag}}$ matches predicted and target velocity magnitudes, $\mathcal{L}_{\text{temp}}$ penalises step-to-step prediction inconsistency via MSE on consecutive-step differences.

L.4 Results

Table 22 illustrates the loss metrics for 25 epochs on a single Tesla T4 GPU (155 min total).

Both training and validation loss were still decreasing at epoch 25 with no sign of overfitting, indicating the model would benefit from longer training. The gap between train and val L2 narrowed from 0.09 at epoch 10 to 0.08 at epoch 25, suggesting the 9.7M-parameter model has not yet saturated the 689-sample training set.

Table 22: Training and validation performance across epochs.

Epoch	Train Loss	Train L_2	Val Loss	Val L_2
1	13.91	1.6761	12.70	1.5824
2	12.87	1.6400	11.08	1.5023
3	11.01	1.4967	9.53	1.3509
4	10.03	1.4179	9.05	1.3182
5	11.16	1.3886	10.22	1.3073
6	10.91	1.3696	9.96	1.2880
7	10.75	1.3554	9.76	1.2707
8	10.55	1.3383	9.63	1.2543
9	10.27	1.3152	9.18	1.2108
10	9.75	1.2627	8.79	1.1783
15	10.12	1.1880	9.21	1.1097
20	9.58	1.1492	8.62	1.0681
25	9.32	1.1314	**8.45**	**1.0541**

M AEROCHRONOMIXER

AeroChronoMixer models unsteady aerodynamic evolution directly on native airfoil point clouds. Its prototype grew out of prior experience solving roughly 13-million-point aircraft-grid physical fields, which motivated a pointwise formulation that preserves geometry without dense uniform-grid lifting. Short velocity histories provide local temporal signal; Fourier and hash-grid coordinate features expose spatial variation Tancik et al. (2020a); Müller et al. (2022); and a lightweight temporal mixer summarizes point histories Tolstikhin et al. (2021). The decoder conditions on boundary proximity, a sampling-density proxy, a compact global token summary, and future time. The design targets near-wall fidelity with a compact inference path, returning five future velocity fields from a frozen 5,833,825-parameter PyTorch model.

Design origin. The architecture was motivated by our earlier prototype for predicting physical fields on aircraft grids with roughly 13 million points. That experience led us to favor representations that operate directly on point sets and preserve geometric information without first constructing a dense volumetric surrogate. AeroChronoMixer therefore keeps most operations pointwise, uses only a small token subset for global conditioning, caches geometry-derived features where possible, and predicts the future fields directly.

Inference interface. AeroChronoMixer follows the competition interface and maps the ten time stamps t, the point positions $x \in \mathbb{R}^{N\times 3}$, the airfoil boundary indices, and the first five velocity fields to the next five velocity fields. The packaged class `AeroChronoMixer` loads `state_dict.pt` during construction, switches to evaluation mode, and freezes all parameters. A direct parameter count gives 5,833,825 parameters before freezing; for batch size B and point count N, the output tensor has shape $(B, 5, N, 3)$, matching the expected horizon and velocity-vector dimensions.

Architecture. The model combines pointwise temporal encoding, coordinate encoding, boundary-aware conditioning, and a direct future-time decoder. Coordinates are encoded with six-frequency Fourier features Tancik et al. (2020a) and an eight-level multiresolution hash grid with two features per level, following the same general motivation as learned hash encodings for continuous spatial fields Müller et al. (2022). Each point's five-frame velocity history is processed by two MLP-Mixer-style temporal blocks Tolstikhin et al. (2021), after which the hidden state is concatenated with explicit statistics of the recent trajectory, including the last velocity, first differences, and a second-difference acceleration proxy.

Geometry and boundary conditioning. For each case, the implementation builds boundary features from the provided airfoil indices: a boundary mask, nearest-boundary distance, inverse distance, direction to the nearest sampled boundary anchor, and a Gaussian near-boundary weight. A separate sampling-density proxy is computed by voxelizing normalized coordinates and using local point counts to estimate a log spacing feature. A small subset of points, biased toward boundary

points when available, is projected into tokens and pooled by learned latent queries to produce a compact global descriptor of the case geometry and flow state.

Decoder. For each of the five future steps, the decoder receives the absolute future time, the elapsed time since the last observed frame, the ratio to the previous time interval, and the normalized horizon index. A near/far zone gate blends two hidden representations based on boundary proximity, and the final velocity vector is produced by a three-expert softmax-gated head Shazeer et al. (2017). Predicted velocities on indexed airfoil boundary points are set to zero before returning the final tensor.

Training objective. The accompanying loss implementation combines a competition-metric proxy with auxiliary terms:

$$L = L_{\text{vec}} + 0.5L_{\text{smooth}} + 0.25L_{\text{spec}} + 0.15L_{\Delta t} + 0.10L_{\partial\Omega}.$$

Here L_{vec} is a spatially weighted vector L2 error, L_{smooth} is a SmoothL1 reconstruction term, L_{spec} compares Fourier coefficients of residuals relative to a linear extrapolation baseline, $L_{\Delta t}$ aligns consecutive future-step differences, and $L_{\partial\Omega}$ emphasizes exact boundary points. Spatial weights increase the contribution of boundary and near-boundary points and are normalized per sample.

N TRANSOLVERRESIDUAL

TransolverResidual uses the Transolver (Wu et al., 2024), whose Physics-Attention maps $\sim$100k points into M=32 learned slices and attends only across those ($\mathcal{O}(NM)$), but gives no local interaction. A per-point degree-2 polynomial is fit to the input snapshots and the network learns only the residual, with zero-initialized decoder weights. What fell short: k-NN features as a local message-passing substitute; no cheap way to inject temporal structure; and the geometry encoding (distance and x-offset to the surface) was too shallow. Learned SDF embeddings, local graph convolutions, autoregressive rollout, and proper slice-count analysis remain to be explored. Training runs for 400 epochs on all 905 samples (90/10 split, seed 42) with AdamW, cosine annealing, BF16, and y-flip augmentation on one L40S ($\sim$6 h, $\sim$2.9M parameters).

N.1 ARCHITECTURE

Polynomial baseline. For each spatial point independently, we construct the Vandermonde matrix from the 5 input time values (normalized to $[0, 1]$), solve the least-squares system via `torch.linalg.lstsq`, and evaluate the degree-2 polynomial at the 5 output times. This closed-form extrapolation captures the bulk of the laminar flow signal at zero learned-parameter cost.

Feature vector (79 dimensions). Each point receives: normalized position (3), flattened input velocities (15), polynomial fit residual on the input window (15), temporal mean and standard deviation of velocity (6), binary airfoil surface flag (1), Euclidean distance to nearest surface point (1), signed x-offset to nearest surface point (1), all 10 time values (10), mean velocity of k=8 nearest spatial neighbors (15), and frame-to-frame velocity differences (12). Distance and k-NN features are precomputed per simulation geometry and cached as sidecar files to avoid recomputation each epoch.

Encoder. A two-layer MLP: Linear$(79 \to 512) \to$ GELU $\to$ Linear$(512 \to 256)$.

Transolver backbone. Eight TransolverBlocks, each consisting of: (i) Physics-Attention (H=8 heads, d_h=32, M=32 slices, orthogonally initialized slice projections, learned temperature), and (ii) a feed-forward network with expansion ratio 1. Both sub-layers use pre-LayerNorm and residual connections; dropout is set to 0.1.

Decoder. LayerNorm$(256) \to$ Linear$(256 \to 15)$, reshaped to $(B, 5, N, 3)$. Both weight and bias are zero-initialized so the model starts as the pure polynomial baseline. The correction is added to the polynomial prediction, and surface-point velocities are zeroed (no-slip enforcement).

Parameter count. $\sim$2.9M trainable parameters.

N.2 TRAINING

Table 23: Training configuration.

Setting	Value
Dataset	905 samples (181 simulations $\times$ 5 time windows)
Validation split	90/10 random, seed 42
Additional data	None
Batch size	1 (gradient accumulation over 4 steps)
Optimizer	AdamW (Loshchilov & Hutter, 2019a), lr $= 10^{-3}$, weight decay $= 10^{-4}$
Scheduler	Cosine annealing (Loshchilov & Hutter, 2017), $T_{\max}$=400, $\eta_{\min}$=10^{-5}
Epochs	400
Precision	BF16 mixed precision
Gradient clipping	Max norm 1.0
Augmentation	y-axis flip (50% probability)
Hardware	Single NVIDIA L40S (40 GB)
Wall-clock time	$\sim$6 hours

Loss function. Variance-weighted relative L^2. Per-point weights $w_i \in [1, 4]$ are derived from the temporal standard deviation of the input velocity, upweighting turbulent and wake regions:

$$\mathcal{L} = \sqrt{\frac{\sum_i w_i \|\hat{v}_i - v_i\|^2}{\sum_i w_i \|v_i\|^2}}.$$

Validation uses the unweighted relative L^2. The checkpoint with the lowest validation loss (evaluated every 5 epochs) was selected.

N.3 INFERENCE

Distance-to-surface and k-NN features are cached by a geometry fingerprint. The first call on a novel geometry incurs a one-time CPU precompute ($\sim$5–10 s for 100k points); subsequent calls on the same geometry are instant. No test-time augmentation was used.

N.4 ACKNOWLEDGMENTS

This work was funded by the Marie Sklodowska-Curie Action MSCA-DN-101119556 (IN-DEEP).

O LEVERSTAILV2SUBMISSION

LeversTailV2Submission is a pointwise MLP given just enough local structure to see its neighbours. Each point carries its position, the five input velocity frames, the broadcast timestamps, a surface flag, and distance-to-surface features $[d, \log(1 + d)]$, the last two encoding the fact that near-wall physics differs sharply from the bulk and that wall influence decays with distance. Because the data is a point cloud rather than a mesh, a k-NN graph stands in for a stencil: neighbours are scored by dot-product attention (Vaswani et al., 2017b), on the reasoning that flow dependence is local but anisotropic, so which neighbours matter is better learned than assumed. The resulting attention weights are reused to aggregate two residual message-passing blocks (Brandstetter et al., 2022) whose edges carry relative position, relative mean velocity, and inverse distance, the relative signals being a natural bias for a field driven by local gradients. The MLP trunk then sees the raw features alongside both message-passing states, which keeps the unsmoothed signal available, and five separate linear heads emit one velocity frame each, on the grounds that each horizon step has its own error structure. The name refers to the training recipe rather than the architecture: the submitted checkpoint was trained to put extra optimisation pressure on the later horizons (the "tail") while retaining stability levers such as EMA and a weighted loss, none of which survive into inference.

Provenance. No write-up was supplied with this submission. The summary above and the details below were reconstructed from the model code and design notes shipped with the submission in the competition repository.[15]

The implementation is deliberately self-contained, importing nothing from the other model packages, to keep the submission auditable and portable. The backbone module is named `StrongMLPKnnMPv2`; the submitted checkpoint is the v2 backbone.

O.1 Architecture

Per-point features. Each point carries a concatenation of: its position (3 dims), for geometry conditioning; the flattened input velocities (T_{in}=$5 \times 3 = 15$ dims), giving the local temporal state; the broadcast timestamps t (10 dims), which allow a time-conditioned operator mapping even when sampling times differ; a surface indicator (1 dim) derived from the airfoil indices, since near-wall physics differs sharply from the bulk (no-slip, boundary layers); a learned neighbour-attention-pooled vector, which injects locality that a purely pointwise MLP cannot see; the per-input-timestep neighbour mean velocity (5×3 dims), a simple and stable statistic for neighbourhood dynamics across the input window; and distance-to-surface features $[d, \log(1 + d)]$ (2 dims), where the $\log(1 + d)$ term improves dynamic range because wall influence decays with distance.

k-NN graph. The data is a point cloud, not a mesh, so there is no stencil to inherit; a Euclidean k-nearest-neighbour graph is used as a substitute. The implementation uses a row-chunked brute-force search, chosen for robustness over speed.

Neighbour attention pooling. For each point i, neighbours j are scored by dot-product attention (Vaswani et al., 2017b): the query comes from $[x_i, \bar{v}_i, s_i]$, where s_i is the surface flag, and the keys and values from $[\bar{v}_j, (x_j - x_i)]$. This yields both a pooled neighbour vector and a set of attention weights, which are reused for message aggregation below. The motivation is that flow dependence is local but anisotropic (wake directionality, shear layers), so which neighbours matter is better learned than assumed.

Edge features. Each edge carries the relative position Δx, the relative mean velocity $\Delta \bar{v}$, and the inverse distance $1/(\|\Delta x\| + \epsilon)$. Relative signals are a natural inductive bias for a PDE-like field, whose evolution is driven by local gradients and geometry.

Residual message-passing blocks. Two blocks (Brandstetter et al., 2022), each of which builds edge messages with an MLP, aggregates them using the attention weights from the pooling stage, updates the node state with a second MLP, and applies a residual connection with LayerNorm (Ba et al., 2016). Two blocks buy expressive power while keeping all interaction local.

Trunk and heads. The MLP trunk sees the raw features concatenated with *both* message-passing states. Keeping the raw signals available guards against oversmoothing while still allowing deeper nonlinear mixing. The final per-point latent is mapped through five separate linear heads, one per output timestep, on the grounds that each horizon step has a different error structure and benefits from timestep-specific capacity.

O.2 Training details

The name records the training recipe rather than the architecture. The submitted checkpoint is the v2 backbone trained with a recipe that raises optimisation pressure on the later prediction horizons (the "tail") while retaining stability levers such as EMA weight averaging and a weighted loss. The author is explicit that this is a training-time choice only: the submitted model is purely the learned neural mapping, and no loss weighting survives into inference.

[15] github.com/gram-competition/iclr-2026/models/

P IMPROVEDMLP

ImprovedMLP is a memory-efficient point-wise MLP for 100k spatial points with 39 input channels (position, velocity, time, pressure, geometry). Four residual blocks (256→512→512→256) with GELU, LayerNorm, and velocity skip connection encode the prior that predictions refine observed velocities. Subsampling to 4,096 points during training while chunking 8,192 points at inference enables GPU feasibility. Training on free Colab T4 (6 GPU-hours)over 50 epochs (816k parameters).

Design choices included pressure fields and geometric features; velocity skip connection as strong inductive bias. GNNs (memory prohibitive for 100k nodes), transformers (O(N²) complexity), and batch normalization (geometry variance) were considered but rejected. This demonstrates competitive prediction through careful feature engineering on consumer hardware.

P.1 ARCHITECTURE SPECIFICATION

The model processes each of 100,000 spatial points independently through a feedforward MLP. Input per point comprises 39 channels:

- Spatial position (3): $\mathbf{p} \in \mathbb{R}^3$
- Input velocity history (15): $[\mathbf{v}_{\text{in},1}, \ldots, \mathbf{v}_{\text{in},5}]$ flattened
- Time encoding (10): broadcasted time vector $\mathbf{t} \in \mathbb{R}^{10}$
- Pressure history (10): per-point pressure across timesteps $[p_1, \ldots, p_{10}]$
- Geometric feature (1): normalized distance to airfoil centroid $d_{\text{airfoil}} \in [0, 1]$

The backbone consists of four residual blocks:

$$\text{ResBlock}_i(\mathbf{x}) = \text{Dropout}\big(\text{GELU}(\text{LayerNorm}(\mathbf{W}_i\mathbf{x}))\big) + \mathbf{x}_{\text{skip}} \tag{20}$$

with channel dimensions 39→256→512→512→256. The final prediction head outputs 15 channels per point, which are reshaped to $(5, 3)$ for the 5 output velocity timesteps.

P.2 DESIGN RATIONALE

Why point-wise processing? Point-wise MLPs avoid the memory overhead of graph neural networks (which require edge computations for 100k-node graphs) and attention mechanisms ($O(N^2)$ complexity). This enables batch processing on consumer GPUs.

Why include pressure as input? During architecture design, we included pressure fields from the dataset as additional signal. Pressure is physically coupled to velocity through Bernoulli's equation and the incompressibility constraint, making it a natural feature for velocity prediction.

Why the velocity skip connection? The skip connection from input velocities to output implements the inductive bias that output velocities are refinements or perturbations of input velocities, rather than completely independent predictions. This is motivated by the incremental nature of time-stepping in fluid dynamics.

Why GELU over ReLU? We selected GELU activations during architecture exploration, as the smooth nature of fluid dynamics suggests activation functions without sharp discontinuities may be beneficial.

P.3 MEMORY AND COMPUTATIONAL CONSIDERATIONS

P.3.1 TRAINING STRATEGY

Point subsampling to 4,096 points (4% of the full 100k point cloud) reduces memory requirements by $\sim$16$\times$ while maintaining sufficient spatial resolution for training convergence. The subsampling is performed via random permutation with an inverse mapping to correctly remap airfoil boundary indices. This strategy allows batch size 4 on T4 GPU (16GB VRAM), achieving practical training speeds.

P.3.2 INFERENCE STRATEGY

At inference, the model processes full 100k-point clouds by chunking them into 8,192-point sections. Each chunk is processed independently through the network, with outputs concatenated. This chunking incurs no accuracy loss since the model is fully point-wise (no cross-point dependencies).

P.3.3 COMPUTATIONAL COST

- Model parameters: 816,127
- Training time: approximately 6 GPU-hours on NVIDIA T4
- Peak training memory: 14GB (batch size 4, 4,096 subsampled points)
- Peak inference memory: 8GB (100k points, chunk size 8,192)
- Inference latency: $\approx$2.5 seconds per 100k-point cloud
- Cost: $0 (Google Colab free tier)

P.4 TRAINING PROGRESSION AND LOSS CURVES

The model was trained for 50 epochs using AdamW optimizer with cosine annealing learning rate schedule.

The training curve exhibits typical learning dynamics: rapid initial improvement (epochs 1-10), steady refinement (epochs 10-30), and fine-tuning as learning rate decays (epochs 30-50). No plateau or divergence was observed, suggesting the model could benefit from extended training.

P.5 DESIGN DECISIONS WE CONSIDERED BUT DID NOT PURSUE

During architecture exploration, we considered but did not implement the following approaches:

Graph Neural Networks. A point cloud naturally suggests graph-based representations where edges connect neighboring points. However, with 100k points, a k-NN graph with modest k (e.g., $k = 16$) generates millions of edges, creating prohibitive memory requirements (estimated 25GB+ per sample). Standard GNN frameworks on consumer hardware cannot feasibly process point clouds of this scale, making this approach impractical despite its theoretical advantages.

Transformer Architectures with Global Attention. Self-attention enabling all-to-all point interactions would theoretically capture long-range flow dependencies. However, $O(N^2)$ complexity yields 10 billion pairwise comparisons per 100k-point cloud, making full attention infeasible. While sparse or local attention variants exist, they would fragment the receptive field across disconnected point neighborhoods, potentially missing critical global flow structures.

Advanced Normalization Schemes. We selected LayerNorm for its stability across point clouds of varying geometry. Batch normalization, while commonly used in computer vision, introduces statistical dependencies across samples in a batch. Since our batches contain diverse airfoil geometries with different velocity field distributions, batch normalization could introduce unwanted variance in training dynamics—a concern we did not empirically validate but identified as a design consideration.

P.6 HYPERPARAMETER CHOICES

Key hyperparameter decisions made during model development:

- **Learning rate (1e-3):** Selected as standard value for transformer-style architectures. Higher rates caused gradient-based instabilities; lower rates slowed convergence unnecessarily.
- **Weight decay (1e-4):** Typical regularization strength for neural networks. Prevented divergence while avoiding excessive regularization.

- **Dropout (0.05):** Minimal dropout applied to avoid disrupting training; set conservatively given the already-small batch size.
- **Batch size (4):** Limited by T4 VRAM; increasing to 8 caused out-of-memory errors. Batch size 2 was feasible but underutilized the GPU.
- **Point subsampling (4,096):** Balances memory constraints with spatial information. Smaller values (2,048) risked losing fine-scale flow structure; larger values (8,192) doubled training time with marginal gains.
- **Chunk size at inference (8,192):** Efficient GPU utilization without exceeding memory.
- **Number of epochs (50):** Arbitrary choice; training showed no signs of convergence plateau and could likely improve with extended training (100+ epochs).

P.7 Future Research Directions

With additional computational resources and time, the following improvements appear promising:

1. **Extended training:** The loss curve shows no plateau at epoch 50, suggesting training to 100+ epochs could yield further improvements.
2. **Fourier positional encodings:** Rather than using raw coordinates, sinusoidal position encodings could help the network learn periodic spatial patterns in flow fields.
3. **Train/validation split:** Current training uses all 810 samples; a proper 80/20 split with early stopping could reduce overfitting risk.
4. **Ensemble methods:** Averaging predictions from multiple independently trained models could improve robustness.
5. **Curriculum learning:** Beginning with coarser point subsampling (1,024 points) and gradually increasing resolution might improve convergence.
6. **Physics-informed constraints:** Incorporating soft penalties for violating incompressibility ($\nabla \cdot \mathbf{v} = 0$) or boundary conditions could bias the model toward physically plausible predictions.
7. **Multi-scale architectures:** Processing the point cloud at multiple resolution levels could help capture both local and global flow structures.

P.8 Code and Reproducibility

All code, trained weights, and training logs are available at:

```
https://github.com/harshitsinghsnu/GRAM_iclr-2026/tree/submission/
                    harshitsinghsnu/models
```

The model can be instantiated and used as follows:

```
from models import ImprovedMLP

model = ImprovedMLP()  # Loads trained weights
output = model(t, pos, idcs_airfoil, velocity_in)
# Output shape: (batch_size, 5, 100000, 3)
```

The implementation requires only PyTorch 2.0+ with no external dependencies. Training can be reproduced on free Google Colab T4 GPU in approximately 6 hours using the provided notebook: `GRaM_Competition_Colab.ipynb`.

P.9 Team and Contact Information

- Harshit Singh (PhD Candidate) — `hs494@snu.edu.in`
- Rajeev Kumar Singh (Professor) — `rajeev.kumar@snu.edu.in`

All affiliated with Shiv Nadar Institution of Eminence, Delhi-NCR, India.

Q FNO3DTIMERES

FNO3DTimeRes adapts the Fourier Neural Operator with Direct Spectral Evaluation (FNO-DSE) (Lingsch et al., 2023) to the temporal prediction of 3D velocity fields around airfoil geometries. FNO-DSE replaces FFT-based spectral convolutions with direct evaluation of truncated Fourier matrices, enabling neural operators to process unstructured point clouds without interpolation to a regular grid. While FNO-DSE was originally demonstrated on 2D static prediction tasks, the framework is extended here through: (1) implementation of a full 3D direct spectral evaluation using runtime-constructed Vandermonde transforms; (2) sinusoidal time embeddings, averaged across the input timesteps and broadcast to all spatial points, to provide an explicit temporal signal; (3) a residual velocity skip connection that projects the last input velocity field directly to the output, allowing spectral layers to focus on turbulent residuals; and (4) geometry conditioning through a binary airfoil surface indicator concatenated to the input features.

Q.1 EXTENDED METHOD DESCRIPTION

We adapt the FNO-DSE to a spatiotemporal forecasting setting. The model receives five observed velocity snapshots and predicts the subsequent five snapshots on an unstructured point cloud containing up to $N = 100{,}000$ spatial locations.

Input representation. For each spatial point x_n, the input feature vector is constructed as

$$f_n = \Big[\underbrace{\tilde{x}_n, \tilde{y}_n, \tilde{z}_n}_{3}, \underbrace{u_{1:n}, \ldots, u_{5:n}}_{15}, \underbrace{m_n}_{1}, \underbrace{\tau}_{16}\Big],$$

where $(\tilde{x}_n, \tilde{y}_n, \tilde{z}_n)$ are min–max normalised coordinates, $u_{t:n} \in \mathbb{R}^3$ denotes the velocity vector at timestep t, $m_n \in \{0, 1\}$ indicates whether the point lies on the airfoil surface, and τ is a learned sinusoidal time embedding shared across all points.

The resulting 35-dimensional feature vector is projected to a hidden width of $W = 32$ through a linear lifting layer.

Direct spectral evaluation on 3D point clouds. Following FNO-DSE, spectral convolutions are performed directly on the irregular point cloud without interpolation to a Cartesian grid. Given coordinates

$$\mathbf{p}_n = (x_n, y_n, z_n),$$

we construct a truncated Fourier basis using frequencies

$$k_x, k_y, k_z \in \{0, \ldots, M-1, -M, \ldots, -1\},$$

with $M = 10$ modes retained per spatial dimension.

The forward transform matrix is

$$V_{kn} = \frac{1}{\sqrt{N}} \exp\Big(- i(k_x x_n + k_y y_n + k_z z_n)\Big),$$

producing spectral coefficients through

$$\hat{u} = V u.$$

The inverse transform is implemented using the conjugate transpose V^*.

Each spectral layer applies learnable complex-valued weights independently to the eight octants of Fourier space before transforming back to physical coordinates. Four such spectral blocks are stacked with pointwise residual convolutions and GELU activations.

Temporal conditioning. The original FNO-DSE (Lingsch et al., 2023) formulation operates on static inputs. To enable temporal forecasting, we encode the physical simulation times using sinusoidal embeddings similar to transformer positional encodings. For each input timestep t_i,

$$\phi(t_i) = [\sin(\omega_1 t_i), \ldots, \sin(\omega_d t_i), \cos(\omega_1 t_i), \ldots, \cos(\omega_d t_i)].$$

The embeddings are processed by a two-layer MLP and averaged across the five observed timesteps. The resulting vector is broadcast to every spatial location and concatenated to the pointwise features.

This explicit time representation allows the model to distinguish between different temporal windows that may exhibit similar flow configurations.

Residual forecasting. A key observation for this challenge is that the most recent velocity field already provides a strong estimate of future large-scale flow structures. Instead of predicting the complete future trajectory directly, we predict a residual correction:

$$\hat{Y} = R(u_{t_5}) + F_\theta(X),$$

where R is a learned linear projection of the final observed velocity field and F_θ denotes the spectral operator.

This residual formulation improved optimization stability and accelerated convergence compared to direct prediction. Empirically, removing the residual connection resulted in slower training and reduced accuracy, particularly in smooth laminar regions.

Geometry encoding. Airfoil geometry is represented through the combination of point coordinates and a binary indicator identifying points that belong to the airfoil surface. The binary mask is constructed from the provided `idcs_airfoil` indices and concatenated to the input features, allowing the operator to condition its predictions on the wing geometry without requiring a separate geometric encoder.

Output head. The final pointwise representation is processed through two fully connected layers,

$$32 \to 128 \to 15,$$

producing velocity predictions for five future timesteps and three velocity components per timestep.

The output tensor is reshaped to

$$(B, 5, N, 3),$$

corresponding to five predicted velocity fields on the original point cloud.

Q.2 What did not work and what led to convergence

During development we identified three primary challenges: memory scalability of the DSE transform, temporal modelling, and reconstruction of high-frequency flow structures. A recurring observation throughout prototyping was that the most recent velocity field already contained most of the information required to predict future large-scale flow behaviour. As a result, increasing model capacity alone yielded limited benefits. This insight ultimately guided us toward a lightweight architecture that focuses on temporal conditioning and residual prediction rather than larger spectral representations.

Memory scaling of the spectral transform. The DSE formulation requires constructing a truncated Fourier transform matrix whose size grows with the cube of the retained mode count. Increasing the number of Fourier modes appeared to be the most direct way to improve reconstruction of fine-scale flow features, but memory consumption and runtime increased rapidly. In practice, mode counts above $M = 10$ provided limited gains relative to their computational cost, making larger spectral resolutions impractical for the available hardware.

Temporal attention. One of the first extensions considered was a lightweight self-attention mechanism operating over the five observed timesteps before the spectral layers. The motivation was to allow the model to learn which previous observations were most relevant when predicting future dynamics. However, with only five input snapshots, the attention mechanism provided little additional information beyond what was already available through direct concatenation of the velocity fields, while increasing model complexity.

Autoregressive temporal prediction. We also considered replacing the direct five-step prediction objective with an autoregressive rollout strategy. In this formulation the model would predict the next timestep, feed the prediction back into the input sequence, and recursively generate the full future trajectory. While conceptually appealing because it more closely matches the underlying dynamical system, preliminary experiments suggested substantially longer training times and increased implementation complexity. Given the competition timeline, we instead adopted a single-shot prediction strategy that forecasts all five future timesteps simultaneously.

Alternative temporal conditioning. The model originally ignored the physical simulation times provided in the dataset. We investigated several approaches for incorporating temporal information and ultimately found that simple sinusoidal time embeddings offered the best trade-off between complexity and performance. More elaborate temporal mechanisms did not provide sufficient gains to justify their additional computational cost.

Multi-scale spectral architectures. A major difficulty of the challenge is the prediction of high-frequency turbulent structures. To address this, we explored the idea of using multiple spectral branches operating at different frequency ranges, for example a low-frequency branch with a small number of modes and a second branch dedicated to higher-frequency corrections. Although this approach is physically well motivated, it substantially increased memory usage and implementation complexity within the DSE framework.

Direct prediction without residual forecasting. Early versions of the model predicted future velocity fields directly from the observed history. These models converged more slowly and produced less accurate predictions in smooth flow regions. Introducing a residual connection from the final observed velocity field consistently improved optimization stability and forecast quality.

Taken together, these experiments led us to favour simple and computationally efficient modifications to the original FNO-DSE architecture. The final model combines sinusoidal time embeddings with residual forecasting from the most recent velocity field, using the latter as a strong low-frequency prior while allowing the spectral operator to focus on modelling the remaining corrections. This combination provided the best trade-off between accuracy, memory consumption, and training stability on the full 100k-point dataset.

Q.3 What we would improve with more time

More efficient spectral operators. The main limitation of the current approach is the computational cost of the DSE transform. Hierarchical, sparse, or local spectral operators could reduce memory consumption and allow larger mode budgets.

Autoregressive temporal modelling. The model predicts all five future timesteps simultaneously. An autoregressive formulation that recursively predicts the next state could better capture temporal dynamics, particularly over longer horizons.

Multi-scale turbulence modelling. The current residual formulation provides a strong low-frequency prior, but turbulent structures remain challenging. A multi-scale architecture with separate low- and high-frequency branches could improve reconstruction of fine-scale flow features.

Richer geometry representations. Airfoil geometry is currently encoded through point coordinates and a binary surface mask. More expressive geometry-aware representations, such as graph-based neighbourhood features or learned geometric embeddings, could improve generalisation across configurations.

Data augmentation and larger models. Additional geometric augmentations (e.g., small rotations, translations, or perturbations of airfoil configurations) could improve robustness and reduce overfitting. Given more computational resources, wider networks, larger spectral resolutions, and model ensembling would also likely improve prediction accuracy.

Q.4 Training details

Hyperparameter	Value
Architecture	FNO with DSE
Raw input channels	19
Time embedding dimension	16
Total input dimension	35
Hidden width W	32
Spectral layers	4
Spectral modes M	10
Output channels	15
Batch size	4
Optimizer	AdamW
Learning rate	10^{-3}
Weight decay	10^{-6}
LR scheduler	StepLR ($\gamma = 0.97$, step size = 10)
Epochs	200
Additional data	None

Table 24: Training configuration for FNO3DTimeRes.

R DeltaGraph

The DeltaGraph model predicts the velocity field as a linear extrapolation from the last two timesteps plus a learned residual correction. The predicted residual comes from a multi-scale graph transformer Vaswani et al. (2017b); Veličković et al. (2018); Ying et al. (2021) that encodes the point cloud at full resolution, downsamples to a coarse subset by farthest point sampling, applies k-nearest-neighbor graph-attention with edge-feature-biased scores, and interpolates back through a skip connection Qi et al. (2017). Model inputs are position, velocities, timestamps, and airfoil-relative geometry features. A three-level variant did not outperform this two-level design. Accurately resolving the high-frequency components of the field motivated a composite loss combining a surface-weighted reconstruction, a graph-Laplacian high-frequency, and a temporal-gradient term. After the relative L_2 error was disclosed as the evaluation metric, training directly on it outperformed the more complex composite loss.

R.1 DeltaGraph model

Architecture. The DeltaGraph model is an encoder-decoder graph network that predicts the velocity field as a linear extrapolation baseline plus a learned per-point correction. It embeds every point of the input cloud, refines a downsampled version with graph attention, and decodes the result back to full resolution as that final correction. We use a single coarse level, as a deeper hierarchy did not improve accuracy.

In detail, the per-point input features described below are first projected to a hidden dimension of 256. We keep these full-resolution embeddings for a later skip connection. Farthest point sampling Qi et al. (2017) then selects a coarse subset of about 8% of the points. The embeddings at these points pass through two graph-transformer blocks Vaswani et al. (2017b); Veličković et al. (2018). A block builds a k-nearest-neighbor graph ($k = 16$) over the coarse positions and runs pre-norm Xiong et al. (2020) multi-head attention (4 heads) along its edges. The attention logit for an edge is the standard scaled dot product of query and key, plus a learned per-head scalar bias Ying et al. (2021). It is computed as a learned linear projection of the edge's geometric features: the

relative position $\mathbf{r}$ of its two endpoints and their Euclidean distance $\|\mathbf{r}\|$. Because each head learns its own map, the heads can attend at different spatial scales. The attention weights are normalized over each node's neighborhood via a softmax, and each block closes with a residual feed-forward network of inner dimension 512.

The decoder lifts the coarse features back to full resolution. Each point takes the inverse-distance-weighted average of its three nearest coarse points Qi et al. (2017), and the result is concatenated with the point's own skip embedding. A Linear-LayerNorm Ba et al. (2016)-ReLU-Dropout Srivastava et al. (2014) block fuses the two. A per-point output head—a Linear-LayerNorm-ReLU-Dropout layer followed by a linear projection—maps the fused features to the velocity correction for all five output timesteps. This correction is added to the baseline, and the airfoil points are then masked to zero. The baseline itself extrapolates each point with the slope $(\mathbf{v}_{-1}-\mathbf{v}_{-2})/\Delta t$ measured from the last two input frames. Because the airfoil indices, and hence the farthest point sampling and the masking, differ between samples, the graph stages are evaluated one sample at a time.

Input features. Every point is described by a 35-dimensional input vector concatenating the point position (3), its velocity at the five input timesteps (15), the five input and five output times (10), and seven geometry features that locate the point relative to the airfoil. The geometry features are the distance to the nearest surface point, a binary indicator that is set within a thin band around the surface (radius 0.02) or on the airfoil itself, the airfoil mask, the surface normal at the nearest surface point, and the cosine between that normal and the direction from the surface to the point, whose sign distinguishes the two sides of the airfoil. The normals come from a local principal component analysis: the airfoil surface is subsampled to at most 512 points, and at each the normal is taken as the direction of least variance among its 16 nearest neighbors. Nearest-surface distances are evaluated in chunks to keep memory bounded. None of these features depend on the flow, so we compute them once per geometry and cache them to disk.

Composite loss. The submitted model minimizes $\mathcal{L} = \mathcal{L}_{\text{recon}} + 0.5\,\mathcal{L}_{\text{hf}} + 0.1\,\mathcal{L}_{\text{grad}}$. The reconstruction term $\mathcal{L}_{\text{recon}}$ is a Huber Huber (1964) (smooth-L_1) loss between predicted and target velocities, reweighted per point by $\exp(-15\,d)+1$ (d the detached distance to the surface, normalized to unit mean) to emphasize the near-surface region. The high-frequency term $\mathcal{L}_{\text{hf}}$ is a graph-Laplacian penalty on a random subset of $16{,}384$ points: it builds a k-nearest-neighbor graph ($k = 16$) and matches the residual $\mathbf{x} - \text{mean}_{\text{nbr}}(\mathbf{x})$ between prediction and target through a Huber loss, which penalizes over-smoothing of fine spatial structure. The subsampling keeps the memory cost at $O(Nk)$. The temporal-gradient term $\mathcal{L}_{\text{grad}}$ is a Huber loss on the frame-to-frame differences $\mathbf{x}_{t+1} - \mathbf{x}_t$.

Relative L_2 variant. Post-competition, we evaluated replacing our composite objective directly with the relative L_2 evaluation metric, i.e. $\sqrt{\sum(\mathbf{y}-\hat{\mathbf{y}})^2/\sum\mathbf{y}^2}$ averaged over the batch. Training on this metric directly simplifies the objective and slightly improves the average relative L_2 score from 0.12 to 0.11.

Training. We optimize with AdamW Loshchilov & Hutter (2019b) (learning rate 10^{-3}, weight decay 10^{-6}) and a cosine-annealing Loshchilov & Hutter (2017) learning rate schedule. Training uses mixed precision Micikevicius et al. (2017) and gradient clipping at 1.0. The roughly 10^5 points per sample limit training to a batch size of one. The data is split $80/20$ into training and validation with a fixed seed. Checkpoints are selected based on the validation loss. No domain-informed validation split or additional data is used.

S SPATIOTEMPORALMNO

The Multiscale Neural Operator (MNO) Wang et al. (2026) acts directly on the unstructured 3D point cloud. An encoder lifts the per-point inputs, and four MNO blocks each fuse a global dimension-shrinkage attention, a local kNN-graph attention and a micro point-wise re-weighting. The decoder predicts a residual around the persistence baseline, the last observed frame, while per-point geometry features such as an airfoil mask, the log wall-distance and a local surface frame, together with an analytic no-slip constraint, inject the boundary physics. Because the field barely changes across the evenly-spaced steps, explicit temporal schemes such as a latent GRU forecaster Chung et al. (2014) or autoregressive unrolling did not improve accuracy, so the velocity history is flattened and the

dynamics inferred implicitly by the MNO stack. Training uses `warped-ifw` under an 80/10/10 split with 32k-point random crops and a relative-L_2 loss.

Problem setup. The warped airfoil flow is modelled as a field-regression problem defined directly on the raw, unstructured 3D point cloud, avoiding any intermediate meshing or voxelization. The model maps the per-point input features and the flattened velocity history to the target velocity field at the prediction horizon.

Architecture. The approach follows the Multiscale Neural Operator (MNO) of Wang et al. (2026): an Encoder, a stack of four MNO blocks, and a Decoder. The encoder, an MLP, embeds each point, namely its 3D coordinates and its auxiliary per-point attributes, into a latent token, so no separate positional encoding is needed. Each MNO block then runs three complementary modules in parallel and fuses their outputs, decomposing the field across scales:

- A **global dimension-shrinkage attention** that captures long-range, large-scale dependencies across the whole point cloud by attending in a projected, reduced dimension, which keeps the global interaction computationally efficient.
- A **local graph attention** that models mid-scale, neighbourhood-level interactions by attending over each point's k nearest neighbours.
- A **micro point-wise attention** that evolves each point's features independently to retain fine-grained, high-frequency variations.

The decoder predicts the output field as a *residual around the persistence baseline*, the last observed frame: the prediction is $\hat{u} = u_{\text{last}} + \Delta u$, where Δu comes from a zero-initialized head. Zero-initialization makes the model start exactly at the persistence baseline, a strong prior for this dataset, so that only the small departure from it is learned.

Boundary physics. Geometry and boundary information enter through two channels. The first is a set of per-point geometry features: an airfoil/solid mask, the log wall-distance, and a local surface frame. The second is an analytic no-slip constraint that enforces vanishing velocity on the airfoil surface, so the boundary condition is satisfied by construction rather than learned from data alone.

Temporal modelling. Explicit temporal strategies such as autoregressive unrolling did not improve accuracy during prototyping. The field changes only marginally across the constant, evenly-spaced timesteps, which makes an explicit temporal model largely irrelevant. The velocity history is therefore flattened into the per-point feature vector and the dynamics are inferred implicitly by the MNO stack. This proved both simpler and more stable than rollout-based alternatives.

Training. No additional data beyond the official `warped-ifw` dataset is used. Training minimizes a relative-L_2 objective on randomly cropped 32k-point sub-clouds, which keeps memory bounded while exposing the model to diverse local neighbourhoods, under an 80/10/10 train/validation/test split. Data augmentation could have been used by simply flipping the whole dataset vertically.

Limitations and future work. The dominant source of error is the turbulence itself. The high-frequency near-wall structure and the disordered wake downstream of the airfoil are the hardest regions to fit, and they are where the residual prediction departs most from the ground truth. A natural first step is to strengthen the local kNN-graph attention so that it better resolves this fine, high-frequency structure, in particular through higher-frequency relative-position encodings such as Fourier features on the neighbour offsets Tancik et al. (2020b).

The other main limitation was practical rather than methodological, namely compute. Although the architecture is parameter-efficient, with roughly 2.15M parameters, it was costly to train, and the cost is driven by the multiscale attention acting on the full point cloud rather than by the parameter count. In particular, the local kNN-graph attention materializes a neighbour tensor for every point, so its memory and floating-point cost grow with both the number of points N and the neighbourhood size k. On a large cloud this is what pushes GPU memory toward its limit. This is precisely why we train on randomly cropped 32k-point sub-clouds and keep k modest, and it in turn leaves little headroom

for the larger neighbourhoods or finer resolutions that would most help the turbulent regions noted above. Scaling the model to resolve turbulence more faithfully will therefore require either a more memory-efficient neighbour attention or a sparser, hierarchical treatment of the shared graph.

T WAVELETLATENTOPERATOR

The Wavelet Latent Operator is a geometry-conditioned latent operator that forecasts the five future velocity frames of the unsteady airfoil flow. Wing geometry and velocity history are both voxelised and encoded by second-order solid harmonic wavelet scattering (building on Kymatio); the history is split into a mean field and a residual before scattering, so the module encodes flow *variation* rather than absolute velocity, with the mean recovered at decode time. The resulting geometry descriptor conditions the operator throughout, gating history channels by their relevance to the wing shape and modulating four 3D-convolutional residual blocks via FiLM (Perez et al., 2018). A point decoder then trilinearly samples the latent volume and predicts seven bounded coefficients over a local basis: the five history velocities, their mean, and the last temporal delta. Future velocity is thus a small, physically plausible correction around observed flow states, added to a repeat-last baseline that enforces zero velocity on airfoil surface points.

Overview. The Wavelet Latent Operator encodes the 3D airflow history via solid harmonic wavelet scattering, processes it in a 3D convolutional latent space modulated by a geometry context vector, and decodes future velocities at each point as a linear combination of local history basis vectors.

Geometry encoding. Airfoil surface points are voxelised onto a 48^3 occupancy grid via trilinear splatting. Second-order solid harmonic wavelet scattering (J=3, L=3, building on Kymatio) is applied to the occupancy volume, producing a 164-dimensional L_p-pooled geometry descriptor per sample.

History encoding. The 5-step velocity history is decomposed into a mean field and a residual. Subtracting the mean field before scattering means the module encodes flow variation rather than absolute velocity, with the mean recovered at decode time as part of the local basis. The residual $(v_x, v_y, v_z, \text{speed})$ is voxelised per timestep and passed through second-order solid harmonic scattering, but this time preserving spatial responses and pooling to 12^3. This gives a $(5 \times C_{\text{maps}} \times 12^3)$ history tensor per sample, where $C_{\text{maps}} = 160$ (4 channels $\times$ 40 scattering maps each), precomputed and cached before training.

Latent operator. Geometry influences the operator in three ways. First, the 164-dim scattering descriptor produces 160 sigmoid gates, one per history map channel, which weight the scattering channels by their relevance to the wing geometry: given this wing shape, which velocity features matter. Second, the 48^3 occupancy volume is average-pooled to 12^3 and concatenated with the gated history maps and a $[-1, 1]^3$ coordinate grid, giving the network direct spatial access to the wing geometry at each voxel. Third, the scattering descriptor is projected to a 64-dim context vector that modulates every residual block via FiLM (Perez et al., 2018). The concatenated input is lifted to 64 channels by a 1×1 `Conv3d`, then processed by four residual blocks (GroupNorm + `Conv3d` $\times2$, GELU, tanh-clamped FiLM), with the context vector injected at each block. A final 1×1 head expands to 5×64 channels, reshaped to give one 64-channel latent volume per output step.

Point decoder. For each output step, per-point latent features are sampled from the latent volume via trilinear interpolation. A 2-layer MLP with LayerNorm decodes 7 scalar coefficients which linearly combine a local 7-vector basis: the 5 history velocity vectors, their mean, and the last temporal delta. Coefficients are bounded via $\tanh \times 0.35$, constraining each output step to a small correction around the local basis rather than an unconstrained prediction. This basis formulation constrains the model to predict physically plausible corrections, since future velocity is expressed as a weighted combination of observed flow states rather than an unconstrained extrapolation. The final prediction adds the repeat-last baseline, which carries much of the low-frequency signal and enforces zero velocity on airfoil surface points.

Training. The loss is MSE $+ 0.2\times$ per-sample relative L_2 $+0.05\times$ temporal-difference MSE. Optimisation uses AdamW (lr $= 3\text{e}{-4}$, weight decay $= 1\text{e}{-5}$) with ReduceLROnPlateau scheduling

(patience 5, factor 0.5, min lr $= 1\mathrm{e}{-6}$), over 75 epochs at batch size 2 with gradient clipping at 1.0; the best checkpoint is saved by validation relative L_2. The data is split $85/15$, stratified by simulation ID, and residual velocities are normalised to zero mean and unit standard deviation computed over a random subsample of training points. Wavelet feature caches (geometry volumes, scattering descriptors, and history maps) were precomputed once on GPU before training and loaded from disk each epoch.

Limitations and future work. Due to time and compute constraints, little exploration of the hyperparameter space was undertaken, and it is reasonable to expect that modest empirical tuning could improve the results substantially. Voxelisation also moves away from the native point cloud structure of the problem; a mesh-native or point-based operator would be a more principled fit for this data.

U PERCEIVERFLOW

PerceiverFlow is a geometry-conditioned spatiotemporal predictor assembled entirely from Perceiver IO parts (Jaegle et al., 2022). A spatial encoder cross-attends from the ∼100k points into 256 latent queries, compressing the cloud to a fixed-size latent set whose cost is independent of the number of points; three transformer layers (Vaswani et al., 2017b) then evolve that set, conditioned on a geometry embedding so the wing shape modulates the dynamics rather than merely entering as another input channel; and a spatial decoder reads the evolved latents back out at the query points to recover the field at full resolution. Unlike almost every other entry, the model predicts the freestream-normalised velocity *directly* rather than as a residual on the last observed frame, so it forgoes the persistence baseline that supplies most of the signal for free elsewhere in this report. Training used 810 samples for 50 epochs with AdamW (Loshchilov & Hutter, 2019b) and a physics-aware loss that upweights points near the surface.

Provenance. No write-up was supplied with this submission. The summary above and the details below were reconstructed from the model code and README shipped with the submission in the competition repository.[16] That README is brief, so this section is correspondingly shorter than the others: it records the architecture and training recipe as documented, and we have not inferred detail beyond it.

U.1 ARCHITECTURE

PerceiverFlow is a geometry-conditioned spatiotemporal predictor built from three stages, all of them Perceiver IO components (Jaegle et al., 2022):

- **Spatial encoder.** A Perceiver IO encoder cross-attends from the ∼100k input points into 256 latent queries, compressing the point cloud to a fixed-size latent set independent of the number of points.
- **Temporal transformer.** Three transformer layers (Vaswani et al., 2017b) evolve the latent set, conditioned on a geometry embedding, so the wing shape modulates the dynamics rather than merely being another input channel.
- **Spatial decoder.** A Perceiver IO decoder reads the evolved latents back out at the query points, recovering a per-point field at full resolution.

U.2 TRAINING DETAILS

Training used 810 samples for 50 epochs with AdamW (Loshchilov & Hutter, 2019b) at learning rate 3×10^{-4}. The loss is physics-aware, upweighting points near the surface where the field varies most sharply. The trained weights are hosted on Hugging Face (`hd-hg/perceiver-flow-weights`) and fetched by the model constructor rather than shipped in the repository.

[16] github.com/gram-competition/iclr-2026/models/

The submission includes a contract check: a zero-argument constructor loads the weights, and a forward pass on the competition signature (timestamps $(B, 10)$, positions $(B, N, 3)$, airfoil indices, and input velocities $(B, 5, N, 3)$) returns $(B, 5, N, 3)$, run at N=1000 rather than the full cloud to keep the check fast.

V ZONALMOE

ZonalMoE is a graph-based mixture of experts model that routes the turbulent and laminar airflow points to separate experts (Shazeer et al., 2017). Each node encodes the recent velocity history, geometry, wall distance, and motion features. The common GNN builds the overall airflow features and the routing gate mixes the laminar expert with a deeper layered turbulent expert to capture future velocity residuals (Veličković et al., 2018). In the initial prototyping phase, we tested the GNN family models (Kipf & Welling, 2017) and a relevant insight revealed the common pattern of ease of prediction over smooth regions but struggle near walls and high-change regions, leading us instead to the hypothesis that these regions should not share the same predictor. This motivated a follow up experiment of using separate GNN branches for different flow zones and aggregate their outputs into a final prediction. However, this simple aggregation was too sensitive to zone classification and worsened the weighted residual MSE during training, motivating a softer routing mechanism instead of the fixed aggregation. This led to the MoE idea formulation which leads to a cleaner version of the same intuition. However, due to time constraints, the baselines and routing gate were not fully calibrated and tested, which may have increased variance and caused suboptimal expert usage and loss of information. Tuning of expert sizes and overall model capacity were planned and are explored further in the Appendix with more details.

W APPENDIX: TRAINING DETAILS AND METHOD EVOLUTION

W.1 ORIGINAL ZONALMOE METHODOLOGY

The original model uses a temporal encoder over the five input frames and has a shared graph attention backbone with two experts specialized for laminar and turbulent regions. In general the regions near the wall are more likely to be turbulent in nature and the routing gate therefore efficiently navigates them to the turbulent expert through a normalized wall distance metric as a simplified replacement for Law of The Wall principle (Pope, 2000) coupled with the vorticity computation as a measure for including the rotational changes at points. We describe more about the details of ZonalMOE specifically focusing on the training details and the intuition behind the design decisions made further.

The model was originally trained as a residual predictor for the next time step and during the competition we came across another relevant submission which focused on fitting a low order polynomial to the given temporal velocity history if it followed a smooth linear like trend and used this extrapolation as a prior for future state prediction. We adopted this in our method by aggregating the routed expert residual to the polynomial extrapolated baseline. In the smaller initial runs this formulation appeared promising and was used in the submitted model under the constraint of competition deadline. However, the final hidden set evaluation revealed poor generalisation of the method therefore prompting us to investigate further on the cause, of which more details are mentioned in the next section.

The training pipeline used a geometry-aware 80/20 train-validation split , AdamW optimization, cosine learning-rate decay, gradient accumulation for the large graph batches, cached geometry preprocessing, and mixed precision when available. Training was performed on NVIDIA H100 80GB HBM3 GPUs and completed in approximately 7 hours. A significant portion of which was utilized in the pre-processing of initial KNN graph construction after which the geometry features are cached and reduce the repeated data fetch overhead substantially.

W.2 ZONALMOE-V2 METHODOLOGY

In practice, the polynomial baseline became the major weakness of our original approach. We describe the failure mode below and also introduce ZonalMOE-v2 as a revised model that addresses

the identified gaps. The extrapolation is not a reliable prior and often produces even worse trajectories than the trajectories obtained by simply repeating the last observed frame. As a result the original model spent a significant portion of its capacity correcting the errors induced by the prior.

Our revised model inherits the same idea of graph MOE on different regions of the airfoil but removes the polynomial baseline and does residual prediction with last-frame persistence instead. This can be described as:

$$\hat{u}_{t+1:t+5} = u_t + \Delta_\theta,$$

where u_t is the last observed input velocity frame repeated across the five output steps, and Δ_θ is the learned residual. This makes the initial model behave in a constrained way: when the learned residual is zero, the model would replicate the value of persistence and avoid unstable explorations.

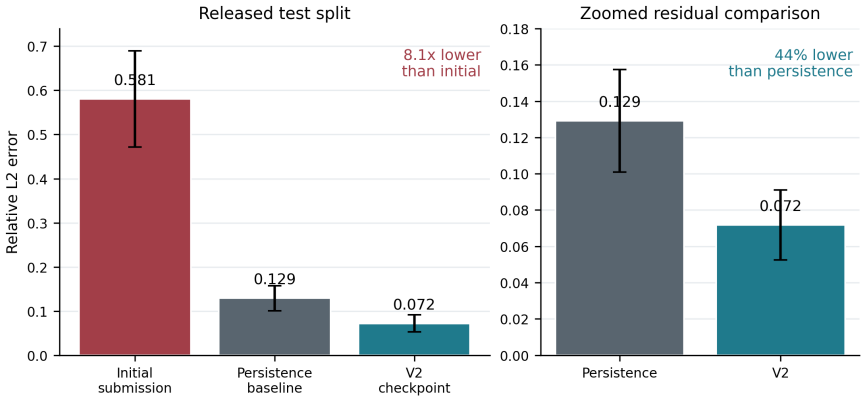

Figure 11: Comparison of the submitted model and ZonalMOE-v2 on the released test split

The architecture was scaled to roughly 6.5M trainable parameters. The router also needed to be redesigned as another analysis from last iteration revealed the poor gating allowed a lot of turbulent points to the laminar expert. We include normalized wall distance, an exponential near-wall feature, vorticity magnitude, speed, an airfoil-surface indicator, and a near-wall/vorticity interaction in this version. The gate was regularized with a small balance loss to ensure that the routing adapts gradually with the different observed geometries.

The main training objective remained weighted residual MSE, with near-wall points upweighted by a factor of 2.0 to motivate efficient capturing of turbulent dynamics. We trained for 40 epochs with AdamW, learning rate 3×10^{-4}, weight decay 10^{-5}, cosine decay to 10^{-6}, gradient accumulation over four samples, and automatic mixed precision on an NVIDIA RTX PRO 6000 Blackwell Server Edition GPU for about 8 hours.

W.3 Results

On the same 80-sample geometry-held-out validation split, V2 substantially improved over both the original submitted wrapper and the persistence baseline:

Method	**Mean relative L2**	**Std.**	**Global relative L2**
Submitted ZonalMoE wrapper	0.4977	0.1353	0.5082
Last-frame persistence	0.1105	0.0306	0.1127
V2 persistence-residual model	0.0602	0.0189	0.0620

The improvements can be inferred clearly from the above. The removal of polynomial extrapolation reduces the global relative L2 from roughly 0.50 to 0.11 for the persistence baseline and a further addition of scaling the parameters and selective soft gating reduced the final error to 0.062 corresponding to 7th in the leaderboard for ZonalMOEv2.

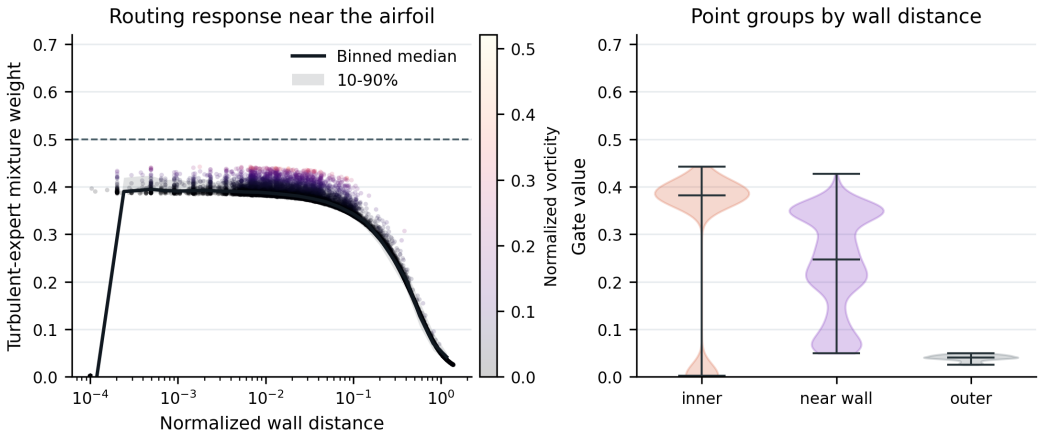

Figure 12: Effect of routing in near wall scenario

The routing efficiency can be observed in Figure 12, for the points closer to the airfoil surface the gating value remains high and as wall distance increases the value decreases gradually. The point group distribution shows a similar trend where the inner and near wall points have higher turbulent expert weights whereas the outer field points are restricted to lower gate values. This indicates that the model learns a soft gating mechanism rather than a hard assignment.

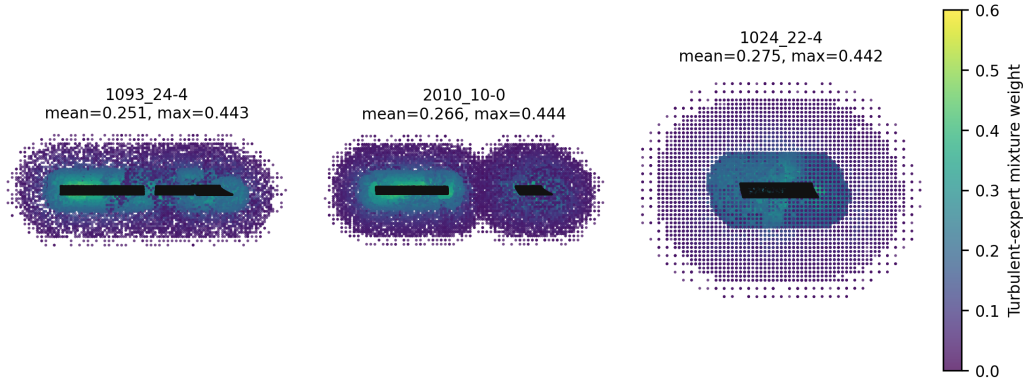

Figure 13: Spatial Distribution and Routing Effect

Figure 13 supports the same observation over three representative test samples selected by their mean gate value, the black points mark airfoil surface and have a higher associated gate values whereas points farther away from the airfoil have significantly lower weights.

These changes suggest that a mixture of experts formulation is suitable for this problem and when paired with a stable residual baseline and calibrated routing can lead to competitive results. Future work could further explore expert size tuning, scaling number of experts, topological neural operators (Bastian et al., 2026) and higher dimension structures for richer encoding as ideas to improve the current model.

X AIRLIQUID

Reported competition result: relative $L^2 = 0.1613 \pm 0.0204$ on the official test split ($n = 95$).

X.1 GENERAL METHOD

AIRliquid (SPATIOTEMPORALGNN) is a geometry-conditioned neural operator that couples a deep residual MeshGraphNet-style spatial encoder (Pfaff et al., 2021b) with a shallow, non-autoregressive temporal Transformer. The spatial encoder applies twelve message-passing blocks of width 128 on a k-nearest-neighbour graph ($k = 16$); the temporal head is a two-layer, four-head Transformer that decodes all five output frames per node in a single pass. Node features combine Fourier-encoded position, a binary surface indicator, normalised wall distance, and standardised pressure over the five input frames. Airfoil velocities are zeroed by a hard no-slip mask both before encoding and after decoding. The decoder predicts a residual correction added to the last observed frame, $u_{\text{out}} = u^{(t-1)} + \Delta v$, and is trained in scaled space on Δv using relative velocity inputs $v_t - v_{\text{last}}$ so that the loss is not dominated by bulk advection. The overall pipeline is shown in Figure 14.

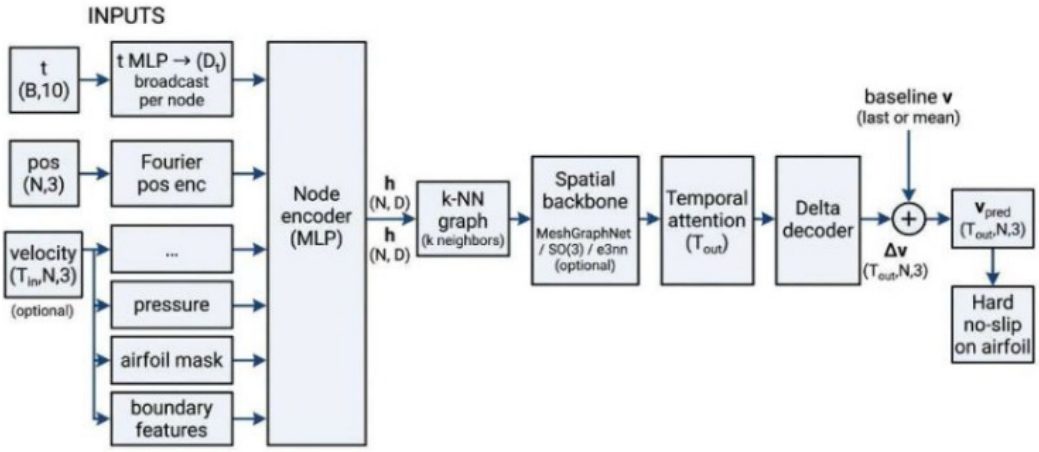

Figure 14: **AIRliquid (SpatioTemporalGNN) architecture.** Per-node inputs (time embedding, Fourier position encoding, velocity history, pressure, airfoil mask, boundary features) are encoded by an MLP, passed through a k-NN MeshGraphNet spatial backbone and a temporal attention head, and decoded to a residual Δv that is added to a persistence baseline before hard no-slip enforcement on the airfoil.

The submitted checkpoint uses the MeshGraphNet backbone (`use_so3_backbone: false`); optional SO(3)-equivariant and `e3nn` steerable encoders remain in the repository as ablation code paths but are not part of the exported weights.

X.2 WHAT DID NOT WORK, AND HOW THE DESIGN CONVERGED

Several families were explored before converging on the submitted configuration. Global architectures—MLPs and Perceiver-style models (Jaegle et al., 2022) that pool or flatten the full point cloud—consistently collapsed toward repeating the last input frame and failed to recover shear layers, separation, and wake detail; the flow structure at this resolution is locally organised near the wall and in advected regions, which motivated full-resolution k-NN message passing instead of a global bottleneck. Applying a Transformer directly over all 100,000 tokens without spatial locality was impractical (memory, unstable training) and gave weak turbulent residuals. Shallow graph encoders underfit the high-frequency component until depth was increased to twelve residual blocks. Training on absolute velocity in scaled space let the loss be dominated by bulk advection; relative-velocity inputs and supervision on Δv against a persistence baseline addressed this. Soft wall-slip penalties left non-zero velocity on airfoil nodes, so hard no-slip masking at both input and output was required. Spatial-only variants without a temporal head produced inconsistent dynamics across the five outputs, while long autoregressive rollouts accumulated error; a single non-autoregressive Transformer pass over the five future frames was sufficient. The optional SO(3)-equivariant and `e3nn` backbones did not outperform the MeshGraphNet path within the 100-epoch budget once pressure and residual training were included. Checkpointing on training loss alone selected models

that looked strong in scaled space but underperformed persistence on validation; checkpoints were therefore selected by the validation relative-L^2 gap versus persistence (`val_rl2_gap`).

X.3 Training details

Only the official Hugging Face dataset `gram-competition/warped-ifw` was used (no external data). Splits are made at the trajectory-group (geometry) level rather than per file: approximately 90% train ($\sim$730 samples), 5% validation ($\sim$40), and 5% internal dev-test ($\sim$40); the official 95-file test split is used only for the final competition metric. Pressure uses the first five input frames, per-sample standardised; velocities are standardised for training and kept unscaled for evaluation.

The submitted model was trained for 100 epochs on a single eight-GPU node of AMD Instinct MI300X accelerators (192 GB HBM3 each) with PyTorch DDP (`torchrun --nproc_per_node=8`), bf16 mixed precision, AdamW (weight decay 10^{-4}), a OneCycle schedule with peak learning rate 10^{-4}, batch size 1 per GPU and gradient accumulation over 4 steps (effective batch size 32). Peak memory was $\approx$48 GiB per GPU. Architecture flags: 12 MeshGraphNet blocks, latent dimension 128, $k = 16$; temporal Transformer with 2 layers and 4 heads; pressure, relative-velocity inputs, Δv loss and time embedding all enabled.

Figure 15 reports the training run. Training and validation relative L^2 are computed on different scales/aggregations (the training Δ-loss is large early on and is not directly comparable to full-field validation error), so the physical HINT error is the more interpretable curve: validation HINT falls to a best of 4.55 near epoch 90, with a held-out test HINT of 5.21. The competition relative L^2 of 0.1613 ± 0.0204 is computed by the official `gram_competition_relative_l2.py` on physical velocities over the 95 official test files—a different protocol from the internal dev metric.

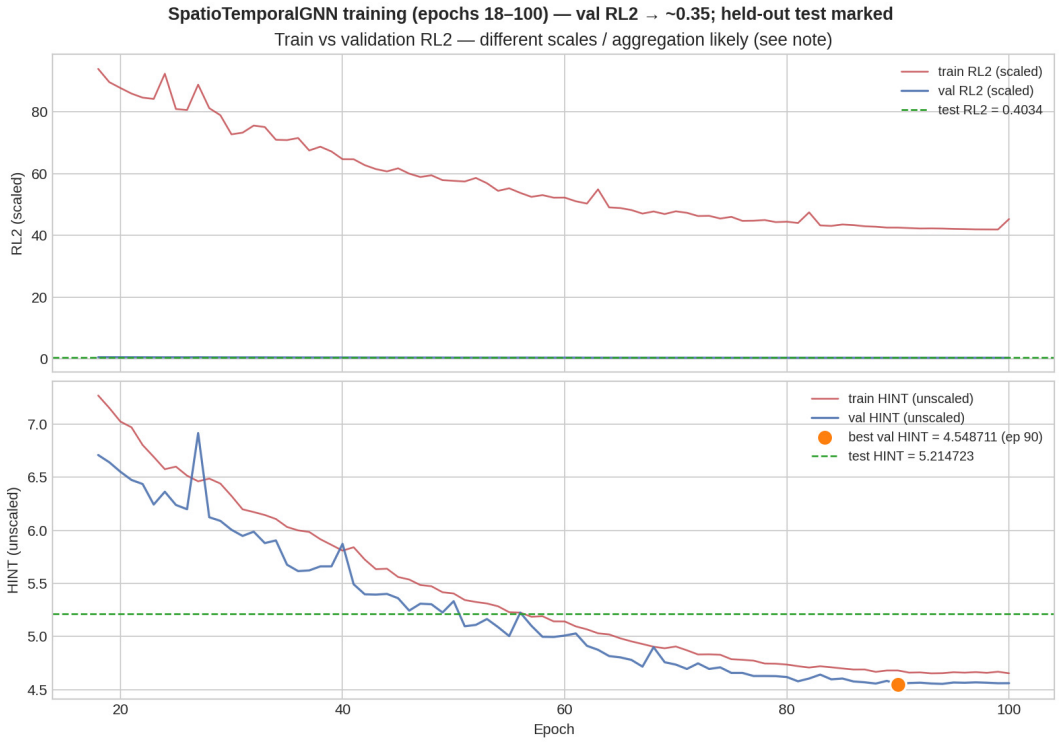

Figure 15: **AIRliquid training curves (epochs 18–100).** Top: scaled relative L^2 for train and validation, with the held-out test value marked. Bottom: unscaled physical (HINT) error, with best validation HINT $= 4.55$ at epoch 90 and held-out test HINT $= 5.21$. Checkpoints were selected by `val_rl2_gap`, not by training loss.

Directions for improvement. The author notes longer training and learning-rate tuning, an optional divergence penalty with a correct discrete operator, capacity ablations, and a fair 100-epoch SO(3)-equivariant ablation under the same pressure, Δ-loss and no-slip recipe as the most promising next steps, together with replacing the temporal Transformer by a surrogate-gradient spiking (SNN) head as a longer-term neuromorphic direction.

Y PT-POINTNET

PT-PointNet is a deliberately lightweight baseline from the author of the AB-UPT entry, submitted out of competition to see how far a compact point model gets. Points are embedded from position, Fourier features of position, the five normalised velocity frames, and the start time. Two Point-Transformer blocks over a fixed k=16 neighbourhood supply local structure; a single max-pooled vector broadcast onto every point supplies global structure. That is the model's entire long-range communication. A decoder maps the concatenated features to five velocity deltas, added to the last input frame and hard-masked to zero on the airfoil. Trained for 150 epochs on one 24,GB GPU, the shipped weights occupy only $\sim$1.8,MB.

Provenance. No write-up was supplied with this out-of-competition submission. The summary above and the details below were reconstructed from the model code and README shipped with the submission in the competition repository.[17]

PT-PointNet comes from the same author as the AB-UPT entry (Appendix K) and is deliberately simpler and smaller than it. Its k-NN attention branch was later reused by the organisers as a point branch in the post-competition ablation ladder (Section 5.2).

Y.1 ARCHITECTURE

The forward pass, per batch sample, has six stages.

Point embedding. Position, Fourier features of position (Tancik et al., 2020a) (frequencies $1, 2, 4, 8, 16, 32, 64, 128$), the five normalised input velocity fields, and the start time are concatenated and projected to a hidden representation.

k-NN neighbourhood. The k=16 nearest neighbours are computed once on positions and reused across all blocks, so the graph construction cost is paid a single time.

Point-Transformer blocks ($\times 2$). Subtraction-form attention over the k neighbours, with a learned relative-position encoding δ added to both keys and values. Attention weights come from an MLP $\gamma(q - k + \delta)$ producing per-head softmax weights over the neighbourhood, in place of the usual dot product. Each sub-block is wrapped in a feed-forward network with pre-LayerNorm (Xiong et al., 2020) and a residual connection.

Global pooling. A per-sample max over points gives one global feature vector, broadcast back onto every point. This is the whole of the model's long-range communication: two hops of k-NN attention, plus one global summary.

Decoder and head. The concatenation of the point embedding, the neighbourhood feature, and the global feature is projected to per-point five-step velocity deltas. The deltas are added to the last input frame, denormalised, and hard-masked to zero at airfoil points, so no-slip holds exactly. Inputs are normalised with per-component mean and standard deviation stored alongside the weights.

Y.2 TRAINING DETAILS

Data. The competition training split, doubled by y-axis mirror augmentation, which exploits the spanwise symmetry of the front-wing flow.

[17] github.com/gram-competition/iclr-2026/models/

Optimisation. Adam on a cosine schedule (peak 5×10^{-4}, decaying to 0) for 150 epochs at batch size 4, with bf16 autocast (Micikevicius et al., 2017) and activation checkpointing during training so the run fits on a single 24 GB GPU. An exponential moving average of the weights (decay 0.999, linear warmup over 100 steps) is tracked throughout, and the best EMA checkpoint on the held-out split is the one shipped.

Footprint. The trained EMA weights occupy $\sim$1.8 MB. The model is zero-argument constructible: it loads its weights and normalisation statistics at construction, moves to CUDA if available, and runs its forward pass under bf16 autocast and `inference_mode` by default.