FLARE++: LOW-RANK ATTENTION WITH ATTENTION-SYNTHESIZED ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Full self-attention is a strong token mixer for PDE surrogates on irregular domains, but its quadratic cost limits its use on large problems. Latent-space attention methods such as PerceiverIO, Transolver, and FLARE (Fast Low-rank Attention Routing Engine) avoid that cost by routing attention among N tokens through $M \ll N$ learned latents. They compress by dot-product matching of the input tokens against M learned query tokens or projection weights: once trained, the same learned templates serve every input. We remove this restriction with FLARE++, a low-rank attention architecture with input-conditioned routing queries. FLARE++ uses FLARE's own encoder to map the N input tokens to M query tokens, which correct the learned queries. The adapted queries then determine how that same input is compressed and redistributed. This preserves FLARE's explicit low-rank factorization and linear $\mathcal{O}(NM)$ complexity, and expresses the complete routing operation with standard scaled dot-product attention (SDPA) calls alone. We also provide a multi-GPU context-parallel implementation that shards input tokens across devices without ever gathering the full token sequence on one of them. FLARE++ reduces FLARE's error by 25% on average across five standard PDE benchmarks, achieving the lowest errors among the efficient models compared. The gains persist on industrial-scale DrivAerML aerodynamics and on Long Range Arena, where average accuracy rises by 3.5% over fixed-query FLARE.

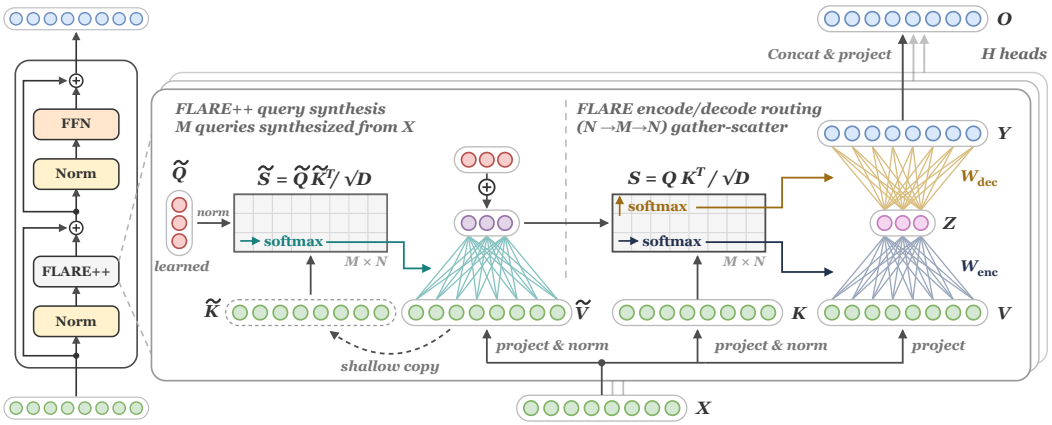
1 INTRODUCTION

PDE surrogates must connect local physics to global field behavior. Self-attention provides this global communication directly: each of the N discretization points can attend to every other. Its quadratic cost, however, makes this approach impractical on high-resolution engineering meshes (Vaswani et al., 2017). A range of efficient token mixers communicate instead through $M \ll N$ latent tokens. This makes global interaction affordable, but introduces a bottleneck in global communication that determines what information to collect and disseminate.

Latent-space attention models, including Perceiver IO (Jaegle et al., 2021a), the Transolver family (Wu et al., 2024; Luo et al., 2025; Zhou et al., 2026), and Latent Neural Operator (LNO) (Wang & Wang, 2024), gather input features into a latent subsequence, process information in that space, and project it back to the full input domain. FLARE (Fast Low-rank Attention Routing Engine) (Puri et al., 2026), which is another latent attention architecture, instead uses latent space to effect low-rank self-attention. Its $M \ll N$ learned queries gather N point tokens into M latent tokens with a cross-attention call, and a second cross-attention call redistributes those tokens to the points, without an intervening latent self-attention stage. The two calls form an encode–decode operator of rank at most M per head, implemented entirely with standard fused scaled dot-product attention (SDPA). By isolating compression and decompression as routing operations, FLARE is a natural starting point.

These latent-space attention architectures compress the input field using dot-product scores against learned weights or latent tokens; in FLARE, these are the M latent query tokens. We call the addressing vectors that define this compression a *compression template*. These methods differ in how they construct this template: FLARE's queries are learned parameters, while other constructions use a learned pointwise map or a prescribed field-dependent rule. In FLARE, training learns M query templates per layer and head, and those same templates then serve every geometry and boundary

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



Operation	Time Complexity	Space Complexity
`K,V,K̃ = project(X)`	$\mathcal{O}(3NC^2)$	$\mathcal{O}(3NC)$
`Q = Qf + SDPA(Q̃,K̃,K̃)`	$\mathcal{O}(NMC)$	$\mathcal{O}(MC)$
`Z = SDPA(Q, K, V)`	$\mathcal{O}(NMC)$	$\mathcal{O}(MC)$
`Y = SDPA(K, Q, Z)`	$\mathcal{O}(NMC)$	$\mathcal{O}(NC)$
`O = merge(Y) @ Wo`	$\mathcal{O}(NC^2)$	$\mathcal{O}(NC)$
FLARE++ layer	$\mathcal{O}(N(4C^2+3MC))$	$\mathcal{O}(5NC+2MC)$

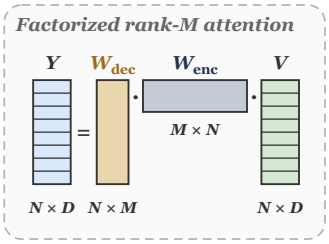


Figure 1: The FLARE++ mixer. Learned seeds $\widetilde{Q}_h$ attend to the input to synthesize M corrections, gated onto learned queries Q_h^{f} to form the routing queries $Q_h(X)$ (left), which gather the N tokens into M latent values and redistribute them (right). The synthesis call shares its keys and values. Arrows mark the softmax axis of each score matrix: $W_{\mathrm{enc},h}$ over the N tokens, $W_{\mathrm{dec},h}$ over the M routes. Below: the mixer over all H heads with its cost, and the rank-M operator its two routing calls compose to. `SDPA` is fused, so S_h and $\widetilde{S}_h$ are never materialized.

condition. The routing weights respond to the current keys, but the queries defining the template do not see the field they compress. This fixed template determines how the rank-M bottleneck is spent.

We present **FLARE++**, which constructs the compression template from the input tokens instead of fixing it after training. This dynamic query construction reuses FLARE's own encode mechanism: learned latent seeds act as the queries of one extra encode call, gathering the N input tokens into M tokens. These tokens define the routing queries of the encode–decode pair that compresses and redistributes the field (Figure 1). Because query construction uses the same encode operation as FLARE, it inherits the efficient fused SDPA implementation and requires no new kernel. One attention call constructs the routes; two more gather and dispatch values along them.

The change is confined to the mixer, and within the mixer to the construction of the compression template. The induced routing matrix remains rank at most M per head for each fixed input, and the complexity remains $\mathcal{O}(NM)$ for fixed channel width. The residual backbone retains the same number of blocks, channel width, and residual update structure, so the comparison with FLARE isolates the routing change without making the network deeper or wider. The mixer itself gains one projection and one attention call, costing 1.1–1.2× FLARE's time per full training run (Section 4.2).

We also make this global routing practical on meshes that exceed one accelerator's memory. Sharding tokens across GPUs needs just the standard log-sum-exp merge of blockwise and Ring Attention (Liu & Abbeel, 2023; Liu et al., 2024), exchanging latent outputs and softmax statistics. FLARE++ invokes it for query synthesis and again for value gathering, while dispatch remains local. This supports full-surface DrivAerML evaluation on 8.2M cells.

Our experiments compare FLARE++ with PDE-specific and general-purpose attention models. Across five standard PDE benchmarks, FLARE++ reduces FLARE's error by 25% on average and achieves the lowest errors among the efficient models compared. Depth and latent-budget sweeps show that the gain persists as either resource grows, and that at a fixed latent budget, synthesized routing

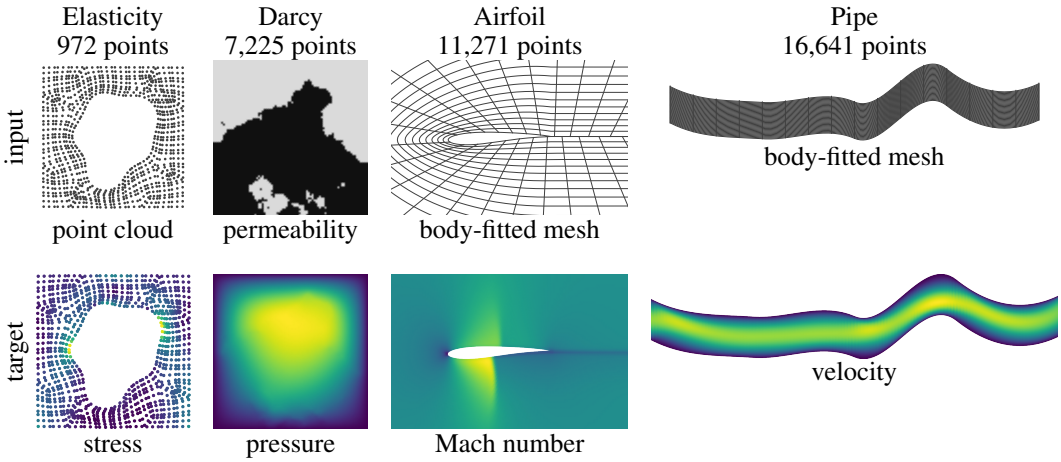


Figure 2: The two-dimensional benchmarks. Top: the input each surrogate receives, namely a point cloud on Elasticity, the permeability field on Darcy, and body-fitted meshes on Airfoil and Pipe. Bottom: the target field. One case each at true aspect ratio; Airfoil is cropped from its far-field mesh.

queries reach a given accuracy with a shallower model. On full-surface DrivAerML, FLARE++ improves on FLARE in all 18 matched depth and latent-budget configurations, by up to 33% in shallow models, and Long Range Arena shows the gains extend beyond physical fields. Our contributions are:

- **Low-rank attention with synthesized routing queries.** FLARE++ uses one extra attention call to construct routing queries from the current input, then uses them to gather and redistribute its tokens, yielding efficient rank-$\leq M$ attention with linear cost in token count.
- **Exact context parallelism for large meshes.** Because tokens interact only through M latents, FLARE++ shards tokens across GPUs and exchanges only $\mathcal{O}(M)$ latent outputs and softmax statistics. This enables single-pass inference on the 8.2M-cell DrivAerML surface.
- **Improved accuracy.** FLARE++ improves on FLARE across benchmarks and is the strongest efficient attention model compared; ablations show synthesized routing queries reach a given accuracy with a shallower model than fixed ones.

2 Related work

Neural operators on irregular domains. Neural operators learn maps between function spaces and extend from regular grids to irregular point sets and complex geometries (Li et al., 2020; Lu et al., 2021; Kovachki et al., 2023; Pfaff et al., 2020; Mousavi et al., 2025). Closest to ours are transformer operators on unstructured meshes, such as GNOT (Hao et al., 2023), and the geometry encoders GINO and GINOT (Li et al., 2023b; Liu et al., 2025). We study the token mixer inside such an operator, and classify latent mixers by where their compression template comes from.

Learned templates. Perceiver IO (Jaegle et al., 2021a), LNO (Wang & Wang, 2024), Set Transformer (Lee et al., 2019), the Transolver family (Wu et al., 2024; Luo et al., 2025; Zhou et al., 2026), FLARE (Puri et al., 2026), Luna (Ma et al., 2021), and TokenLearner (Ryoo et al., 2021) compress against a learned template that is fixed at inference. In each, latent m keeps its identity across inputs. Perceiver, LNO, and Transolver additionally apply nonlinear operations in the latent space.

Data-derived templates. Other methods use the data itself as the template, selected by a prescribed rule. Nyströmformer uses segment means of the queries and keys as landmarks (Xiong et al., 2021), Agent Attention pools the query tokens into agent tokens (Han et al., 2024), UPT aggregates the input onto supernodes sampled from the input points (Alkin et al., 2024), and AB-UPT attends through sampled anchor points (Alkin et al., 2025); hashed and clustered attention group tokens by similarity (Kitaev et al., 2020; Vyas et al., 2020). These templates change with every input, but their

Table 1: Test relative L^2 error (in %) on standard PDE benchmarks; bold and underline mark best and second-best among efficient models. Full self-attention is shown separately as a quadratic-cost reference, and $\sim$ marks runs prohibitively slow. *Mean over ten random seeds (Section D.1).

Model	Elasticity (1K points)	Darcy (7K points)	Airfoil (11K points)	Pipe (16K points)	DrivAerML-40K (40K points)
Full self-attention	0.41	0.43	0.58	$\sim$	$\sim$
PerceiverIO	2.80	2.06	0.77	0.69	24.80
Set Transformer	<u>0.57</u>	1.18	0.67	0.39	<u>6.56</u>
GNOT	1.33	1.69	10.30	0.59	11.50
LNO	0.93	0.76	1.78	0.81	14.60
Luna	0.59	0.84	0.60	<u>0.38</u>	6.90
Transolver	0.72	1.04	0.60	<u>0.38</u>	7.39
Transolver++	1.02	2.76	1.27	0.76	9.39
Transolver-3	0.68	1.03	0.73	0.44	7.36
AB-UPT	1.34	2.70	2.93	4.12	10.0
FLARE	0.71*	<u>0.76</u>	<u>0.57</u>	0.51	7.22
FLARE++ (ours)	**0.39***	**0.62**	**0.50**	**0.35**	**5.80**

latents carry no identity across inputs: anchor m on one geometry bears no relation to anchor m on another, and segment means and pooling rely on token order or grids absent on unstructured meshes.

Recurrent latent state. In Slot Attention (Locatello et al., 2020), the original Perceiver's iterated cross-attention (Jaegle et al., 2021b), Q-Former (Li et al., 2023a), and Luna (Ma et al., 2021), the latent content produced by one cross-attention becomes the next one's query. FLARE++ carries no latent state across blocks: each block rebuilds its routing queries from a learned template, and the path that constructs them is separate from the path that gathers values.

Other efficient attention. Kernel feature maps linearize attention without a latent bottleneck (Katharopoulos et al., 2020; Choromanski et al., 2020; Zhang et al., 2024); Section D.5 compares several such mixers. Unlike hypernetwork and FiLM conditioning, which generate weights or feature modulations (Perez et al., 2018; Cho et al., 2023), FLARE++ conditions only the attention queries, as a bounded deformation of a learned basis.

Existing latent mixers thus use either a learned template fixed at inference or a data-derived template without persistent latent identities. FLARE++ combines the two: it keeps a learned template and adapts it to each input. Because this adaptation is built by attention rather than by sampling or pooling, it is permutation-invariant and defined for any discretization.

3 METHOD

3.1 PRELIMINARIES

We consider PDE surrogates that operate on fields discretized at N spatial points, each carrying problem-dependent inputs such as coordinates, boundary conditions, material parameters, or an initial state. A pointwise input projection embeds these into N tokens, $X = [x_1, \dots, x_N]^\top \in \mathbb{R}^{N \times C}$, with hidden width C. Residual token-mixing and feedforward (FFN) blocks communicate information between tokens, and a pointwise output projection decodes the solution field. We review full self-attention and FLARE's fixed-query operator, treating heads independently as in Puri et al. (2026).

Full self-attention. Learned projections construct $Q = XW_Q$, $K = XW_K$, $V = XW_V$, split into H heads of width $D = C/H$, and head h returns $Y_h = \mathrm{softmax}(Q_h K_h^\top / \sqrt{D}) V_h$; the head outputs are concatenated and projected by W_O. Both sides of the $N \times N$ score matrix depend on the input, and each token can attend directly to every other token, at $\mathcal{O}(N^2 D)$ work per head. We write SDPA

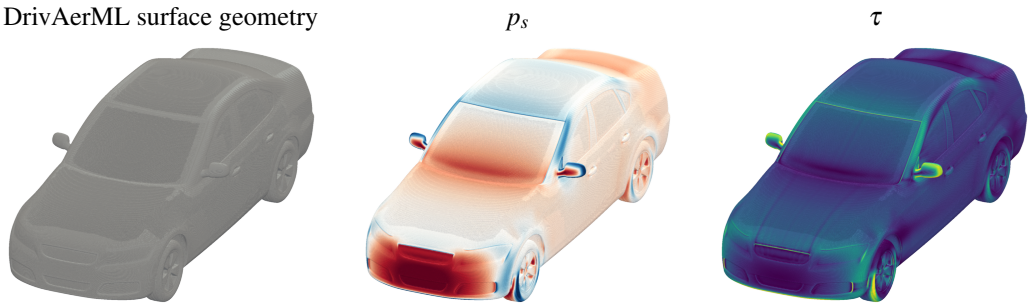


Figure 3: The DrivAerML benchmark: surface geometry, surface pressure p_s, and wall-shear magnitude τ on the full surface mesh of 8.2M cells (colour scales in Appendix E). This is the full-surface task of Section 4.2, distinct from the 40K-point DrivAerML-40K of Table 1.

for this primitive as a function of its three arguments,

$$\mathrm{SDPA}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{D}}\right)V \in \mathbb{R}^{N_q\times D}, \qquad Q\in\mathbb{R}^{N_q\times D}, \quad K,V\in\mathbb{R}^{N_k\times D}, \tag{1}$$

where the softmax normalizes over the N_k keys. Self-attention is the case $N_q = N_k = N$; the operators below take $N_q \neq N_k$ to move information between N tokens and M latent routes.

FLARE: fixed-query latent routing. FLARE (Puri et al., 2026) replaces the dense matrix with an encode–decode factorization through $M \ll N$ latent routes. Only keys and values are projected from the input; each head owns an independent learned query set:

$$K_h = XW_{K,h}\in\mathbb{R}^{N\times D}, \qquad V_h = XW_{V,h}\in\mathbb{R}^{N\times D}, \qquad Q_h\in\mathbb{R}^{M\times D} \text{ is learned.} \tag{2}$$

The encoder and decoder are standard SDPA calls,

$$Z_h = \mathrm{SDPA}(Q_h,K_h,V_h), \qquad Y_h = \mathrm{SDPA}(K_h,Q_h,Z_h), \tag{3}$$

which, writing $S_h = Q_hK_h^\top/\sqrt{D}\in\mathbb{R}^{M\times N}$, $W_{\mathrm{enc},h} = \mathrm{softmax}(S_h)$ and $W_{\mathrm{dec},h} = \mathrm{softmax}(S_h^\top)$, gather the N input values into M latent values and scatter them back:

$$Y_h = W_{\mathrm{eff},h}V_h, \qquad W_{\mathrm{eff},h} = W_{\mathrm{dec},h}W_{\mathrm{enc},h}\in\mathbb{R}^{N\times N}. \tag{4}$$

Because $W_{\mathrm{eff},h}$ factors through M latent routes, its rank is at most M, and it is never materialized: the two SDPA calls apply its factors sequentially at $\mathcal{O}(NMD)$ cost, but the queries Q_h that define those routes are fixed across samples.

3.2 FLARE++: ROUTING-QUERY SYNTHESIS

The queries Q_h define how the M latent routes score the input tokens. We retain the M-route bottleneck and its learned queries, and let the current input adapt them. Because the output of the synthesis call becomes the query of the routing calls, the routing vectors move while the encoder is still learning to use them, which makes early training unstable (Section A.2). We control this by normalizing every tensor that enters an attention score per head with RMSNorm (Zhang & Sennrich, 2019; Henry et al., 2020; Dehghani et al., 2023), and by anchoring the synthesized queries to a fixed learned reference, as described next (Section A.3, Section A.4). We write $\mathcal{N}_a$ for RMSNorm with a learned scale and $\mathcal{N}_0$ for RMSNorm without one, so that every row of $\mathcal{N}_0(\cdot)$ has unit root-mean-square (RMS) norm. To construct the adaptation, we apply FLARE's encoder a second time, with learned seeds $\widetilde{Q}_h\in\mathbb{R}^{M\times D}$ as its queries and normalized synthesis keys, shared with the values,

$$\widetilde{K}_h = \mathcal{N}_0(X\widetilde{W}_{K,h}), \qquad \widetilde{V}_h = \widetilde{K}_h. \tag{5}$$

Sharing them costs no substantial accuracy (Section A.6). The encoder's M outputs are sample-specific query corrections,

$$Q_h^{\mathrm{d}}(X) = \mathrm{SDPA}\left(\mathcal{N}_a(\widetilde{Q}_h),\widetilde{K}_h,\widetilde{V}_h\right) = \mathrm{softmax}\left(\frac{\mathcal{N}_a(\widetilde{Q}_h)\,\widetilde{K}_h^\top}{\sqrt{D}}\right)\widetilde{V}_h\in\mathbb{R}^{M\times D}. \tag{6}$$

Each row of $Q_h^{\mathrm{d}}(X)$ is a convex combination of unit-RMS value rows, so the triangle inequality gives $\mathrm{RMS}(Q_{h,m}^{\mathrm{d}}) \le 1$. The correction is thus bounded without being normalized, and its magnitude records how coherently the attended features agree (Section A.4). The routing queries combine a learned reference set $Q_h^{\mathrm{f}} \in \mathbb{R}^{M\times D}$, as in FLARE, with this correction through a learned per-head gate $g_h = \sigma(\gamma_h) \in (0,1)$:

$$Q_h(X) = \mathcal{N}_0(Q_h^{\mathrm{f}}) + g_h\, Q_h^{\mathrm{d}}(X). \tag{7}$$

The fixed term has unit RMS and provides a common routing basis; the synthesized term, of RMS at most $g_h < 1$, adapts it to the current field, so every routing query has RMS between $1 - g_h$ and $1 + g_h$. As $g_h \to 0$, the construction approaches fixed-query routing. Routing with $Q_h^{\mathrm{d}}(X)$ alone gives essentially the same FP32 accuracy as the gated form, reducing FLARE's error by 24% without the routing-key and fixed-query normalizations, so those normalizations do not explain the gain; adding them to FLARE did not improve it (Section A.8, Section A.4). The gate matters in the mixed-precision study of Section D.2: FLARE++ holds its FP32 accuracy in FP16 and trains without difficulty, whereas the ungated variant degrades and becomes hard to train (Section A.2, Section A.4). Each synthesized correction aggregates features across the current field, so changing one part of the field can change the queries that address every token in the block. With routing keys $K_h = \mathcal{N}_a(XW_{K,h})$, whose learned scale sets the temperature of the routing logits, and values $V_h = XW_{V,h}$, the routing queries enter both factors of the FLARE operator:

$$Z_h = \mathrm{SDPA}(Q_h(X), K_h, V_h), \qquad Y_h = \mathrm{SDPA}(K_h, Q_h(X), Z_h). \tag{8}$$

Writing the resulting routing factors as $W_{\mathrm{enc},h}(X)$ and $W_{\mathrm{dec},h}(X)$ gives

$$Y_h = W_{\mathrm{dec},h} W_{\mathrm{enc},h} V_h, \qquad \mathrm{rank}(W_{\mathrm{dec},h} W_{\mathrm{enc},h}) \le M, \tag{9}$$

so the current field determines the queries used for both gathering and dispatch. The bound holds for each fixed input; the map remains nonlinear because the routing weights depend on X. The full mixer is three fused SDPA calls (Figure 1), with no custom kernel or sequence-projection matrix. Every added component (synthesis projection, seeds, gate, per-head normalizations) acts only through the routing scores $Q_h K_h^{\top}$; the value, output, and feed-forward paths are FLARE's (Table 2).

Computational complexity. Both mixers are built from two primitives: a pointwise projection, at $\mathcal{O}(NC^2)$ time and $\mathcal{O}(NC)$ space, and a token–latent SDPA call, at $\mathcal{O}(NMC)$ time and $\mathcal{O}(NC)$ space, since a fused kernel never materializes the $N \times M$ scores. FLARE uses three projections (K, V, output) and two SDPA calls, $\mathcal{O}(N(3C^2 + 2MC))$ per block; FLARE++ adds $\widetilde{K}$ and one call, $\mathcal{O}(N(4C^2 + 3MC))$, and the gated sum adds $\mathcal{O}(MC)$. For fixed M and C, both are linear in N, against $\mathcal{O}(N^2C)$ for full self-attention. The additional projection and attention call increase mixer arithmetic by a factor between $4/3$ and $3/2$, depending on M/C. Section 5.2 measures the corresponding time and memory cost and relates it to the accuracy gained at a given depth.

3.3 Multi-GPU context parallelism

Linear complexity alone does not make a high-resolution mesh fit on one accelerator, because pointwise activations still grow with N. We therefore shard the token dimension across R ranks, rank r holding $X_r \in \mathbb{R}^{N_r\times C}$ with $\sum_r N_r = N$, which leaves projections, normalization, FFN layers, and residual updates local. The only globally coupled primitive is the encoder gathering sharded tokens into M replicated latents. Each rank runs a fused SDPA call on its local keys and values, returning a local output Z_r and rowwise log-normalizer L_r; the exact global result is the standard log-sum-exp merge of blockwise attention (Liu & Abbeel, 2023; Liu et al., 2024; Dao et al., 2023),

$$L = \log \sum_{r=0}^{R-1} \exp(L_r), \qquad Z = \sum_{r=0}^{R-1} \exp(L_r - L)\, Z_r, \tag{10}$$

which equals the encoder applied to the concatenated sequence in exact arithmetic. With M replicated latent queries, the merge is one all-reduce of latent outputs and softmax statistics, $\mathcal{O}(HM(D + 2))$ values per rank independent of N; reduction order can change rounding. Decoding, $Y_r = \mathrm{SDPA}(K_r, Q, Z)$, is local once Q and Z are replicated, so no all-gather over tokens is ever required. FLARE invokes the encoder once per mixer; FLARE++ invokes it twice, for $Q(X)$ and for Z. The backward pass follows the same decomposition into local attention and global reductions (Appendix B); Section 5.2 evaluates the scaling.

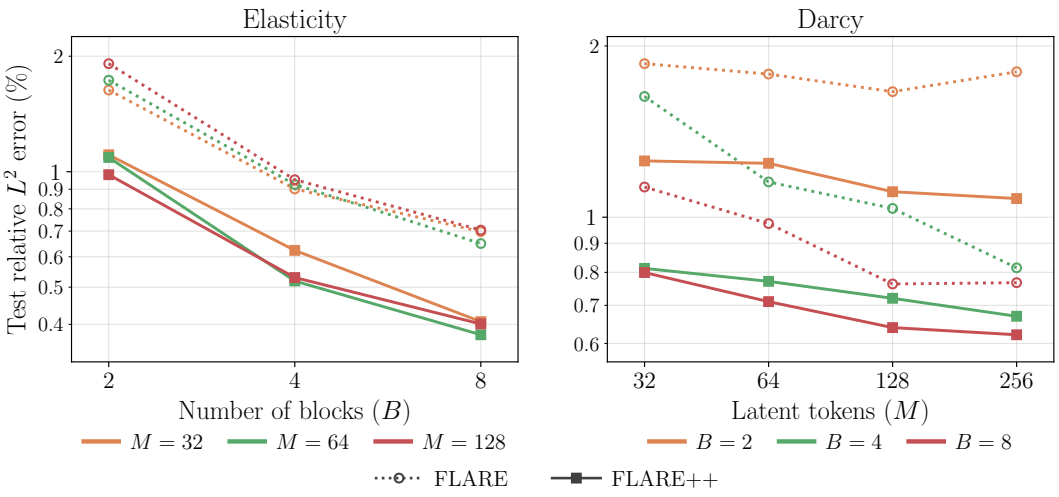


Figure 4: Test error of FLARE and FLARE++ on Elasticity and Darcy over the joint latent-budget and depth grid (M, B). FLARE++ is more accurate in every cell; on Darcy both mixers benefit from additional latents, with diminishing returns beyond $M = 128$ at $B = 8$.

4 Experiments

4.1 Standard PDE surrogate benchmarks

Benchmark problems. We evaluate on the Elasticity, Darcy, Airfoil, and Pipe benchmarks of the Transolver and LNO papers (Wu et al., 2024; Wang & Wang, 2024) and on the DrivAerML-40K benchmark of the FLARE study (Puri et al., 2026), spanning structured and unstructured discretizations of 1K–40K points (Figure 2, Section C.1).

Comparison protocol. We compare FLARE++ with FLARE, the Transolver family, Luna, AB-UPT, and full self-attention as token mixers in a shared residual backbone (Puri et al., 2026; Wu et al., 2024; Luo et al., 2025; Zhou et al., 2026; Ma et al., 2021; Alkin et al., 2025; Vaswani et al., 2017): channel width, head count, depth, latent budget, input and output projections, feed-forward blocks, and training settings are fixed within each dataset, so the comparison isolates the token mixer (Section C.3). AB-UPT uses 512 anchors throughout, since at the 64–128 latent budgets of the other mixers its error is markedly worse. Parameter counts therefore differ by mixer (Table 10), which is also why our FLARE entries are not those of the parameter-matched table in the FLARE study (Section C.3); FLARE++ carries 14–23% more parameters than FLARE, and we account for the mixer's cost in step time and memory at a given token count (Section 5.2) and through the joint depth and latent-budget sweeps (Section 5.1). Set Transformer is evaluated as a complete architecture; PerceiverIO, GNOT, and LNO results are reproduced from the FLARE study (Lee et al., 2019; Jaegle et al., 2021a; Hao et al., 2023; Wang & Wang, 2024). All runs use FP32 and report test relative L^2 error; full self-attention provides a quadratic-cost reference on the three smallest problems. Since Section 4.2 uses FP16, Section 5.3 checks that precision does not alter the mixer ranking.

Results. FLARE++ achieves the lowest error among the efficient models compared on all five benchmarks (Table 1). It reduces FLARE's error by 12–45% (25% on average) and Transolver-3's by 20–43% (31% on average). Averages are over per-task relative reductions. The improvement spans solid mechanics, porous flow, and aerodynamics. The gains extend beyond comparisons with PDE-specific architectures. FLARE++ improves on Set Transformer and on Luna on all five tasks. The results favor adapting a compact set of routes to the field. Full self-attention provides a complementary reference for the cost of compression. FLARE++ has lower error on Elasticity and Airfoil, while full self-attention retains the lowest Darcy error. A low-rank routing bottleneck can therefore support accurate predictions without reproducing unrestricted token-to-token attention.

4.2 Industrial-scale aerodynamics: DrivAerML

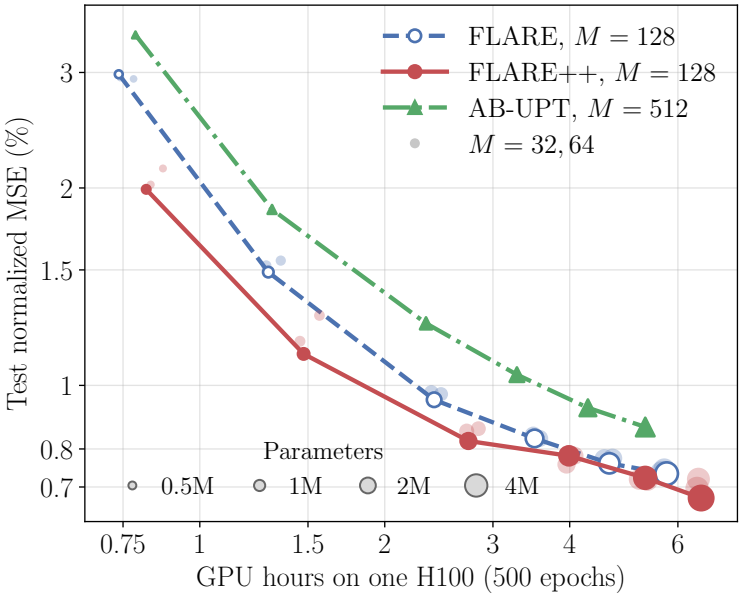


Figure 5: Test normalized MSE on full-surface DrivAerML against GPU hours per training run on one NVIDIA H100. Lines trace depth at $M = 128$ for FLARE and FLARE++ and 512 anchors for AB-UPT. Marker area is proportional to the parameter count, 0.37–4.72M over $B \in \{2, 4, 8, 12, 16, 20\}$. Figure 8 plots the runs against depth.

Setup. DrivAerML (Ashton et al., 2024) tests whether the routing gains persist in industrial-scale automotive aerodynamics (Figure 3). Models predict surface pressure and three wall-shear components, training on 100K sampled cells per step and evaluating on the full 8.2M-cell surface under context parallelism. We compare FLARE and FLARE++ across three latent budgets and six depths with the same backbone and training settings, and include the surface-only AB-UPT mixer with 512 anchors. We also attempted to include Transolver-3, but its official recipe is FP32 only and every 16-bit run we tried on DrivAerML diverged (Section C.3). The metric is normalized four-channel MSE (in %), distinct from the pressure-only relative L^2 error of DrivAerML-40K. We report it against training cost, measured as the GPU hours to train each run for 500 epochs on one NVIDIA H100 (Section D.4). Section C.1 gives the sampling, normalization, and model details.

Results. FLARE++ improves on FLARE in all 18 matched configurations of depth and latent budget, by 31% at two blocks, 22% at four, and 13% at eight when averaged over latent budgets, and by 6–7% in the deepest models tested (Section D.4). A full FLARE++ training run costs 1.1–1.2× the GPU hours of the matching FLARE run (Figure 5). Above 3.4 GPU hours, every FLARE run is dominated by a FLARE++ run that is both cheaper and more accurate, and FLARE's lowest error, 0.734% at 5.75 GPU hours, is undercut by FLARE++ at 5.19 GPU hours. Below 2.5 GPU hours the two models alternate, although at the smallest budget FLARE++ lowers the error from 2.93–2.98% to 1.99% for under 0.1 additional GPU hours. AB-UPT has higher error than FLARE and FLARE++ at every depth despite its larger anchor budget, and every AB-UPT run is dominated by a FLARE or FLARE++ run of equal or lower cost. Together with the standard benchmarks, these results show that query synthesis improves prediction across a wide range of spatial resolutions.

4.3 Long Range Arena benchmark

We evaluate FLARE++ on Long Range Arena (LRA) (Tay et al., 2021b) with FLARE's backbone and task configurations (Puri et al., 2026), to test whether the routing improvement transfers to sequence and image tasks. FLARE++ improves on FLARE on all five tasks, raising average accuracy from 58.08% to 61.60%, second only to the published Luna result (61.95%), and leads on two tasks. Section D.5 reports full results and protocol.

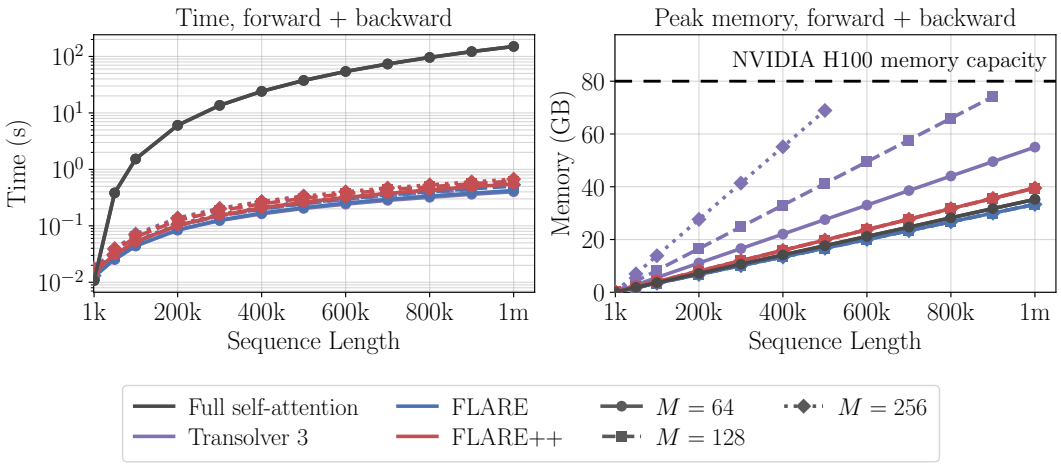


Figure 6: Forward-plus-backward time and peak memory against input size at $B = 8$, $C = 128$, $H = 8$, FP16, on one NVIDIA H100; Section D.3 gives the measurement details.

5 Scaling, Efficiency, and Ablations

5.1 Latent Budget and Residual Depth

We jointly vary the latent budget M and residual depth B of FLARE and FLARE++ on Elasticity and Darcy, with other settings as in Section C.2 (Figure 4). FLARE++ is more accurate than FLARE in all 21 matched configurations, by 16–50%, so the gain persists across communication budgets and backbone depths. At a fixed latent budget, FLARE++ reaches a given error at a shallower depth than FLARE on both tasks. Increasing the latent budget alone does not recover this accuracy: on Elasticity, raising M from 32 to 128 leaves FLARE unchanged at every depth while FLARE++ improves (marginally at $B = 8$), so the same rank budget predicts better with adaptive routing queries.

5.2 Efficiency and Scaling

On a single H100, from 1K to 1M tokens (Figure 6, Section D.3), fused SDPA keeps memory linear in N for every attention model tested, but full self-attention retains quadratic work and at 500K tokens is two orders of magnitude slower than FLARE++. FLARE++ costs 1.2–1.3× FLARE's step time and 1.18× its peak memory, stable beyond 100K tokens and below the $4/3$–$3/2$ mixer-arithmetic increase of Section 3.2 because the feed-forward network dilutes it. On up to four H100 GPUs, context parallelism keeps time and memory efficiencies at or above about 0.91 and 0.95 (Section B.6).

5.3 Mixed-Precision Sensitivity

Section D.2 repeats the depth sweep in FP32 and FP16 for full self-attention, FLARE, and FLARE++ on Elasticity, Darcy, and Airfoil. Precision changes full self-attention's error by up to 82%, but FLARE++ remains below FLARE in every comparison and is the least sensitive mixer: its FP16 error stays within 8% of its FP32 error at every depth, against 0.88–1.33× for FLARE and 0.70–1.82× for full self-attention. Precision thus confounds comparisons against full self-attention but not between latent mixers; FLARE++'s stability comes from its fixed reference and gate (Section A.4).

6 Conclusion

FLARE++ lets a low-rank attention operator decide from the input itself how to compress and redistribute it, rather than reusing queries fixed at training time, with FLARE's attention primitive, at linear cost, on meshes too large for one GPU. It improves on fixed-query FLARE across PDE benchmarks, full-surface DrivAerML, and Long Range Arena at 1.1–1.2× FLARE's time per full training run. The depth and latent-budget sweeps indicate that improved routing, not added depth or latents, is what reaches a given accuracy with a shallower model.

AI USE STATEMENT

We used generative AI tools (large language model assistants) throughout this work. They were used to brainstorm and design the method, to formulate and check the mathematical claims, to write and run the experiment, analysis, and plotting code, to analyze the results, and to draft and edit the text of this paper and format its references. We did not use generative AI tools to generate data; all datasets are the published benchmarks cited in the paper. All AI-assisted work was reviewed by the authors: the derivations were checked by hand, the context-parallel implementation was tested numerically against its single-device counterpart, every reported number was traced to its experiment logs, and all text was revised by the authors. We take responsibility for the final content of this work, including the method, claims, code, and text produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix A specifies the mixer operations, and Appendix B specifies the context-parallel implementation and its backward pass. The remaining appendices document experiment configurations, extended results, and qualitative predictions. The supplementary material contains the training code and runnable scripts for the key results.

REFERENCES

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

Benedikt Alkin, Maurits Bleeker, Richard Kurle, Tobias Kronlachner, Reinhard Sonnleitner, Matthias Dorfer, and Johannes Brandstetter. AB-UPT: Scaling neural CFD surrogates for high-fidelity automotive aerodynamics simulations via anchored-branched universal physics transformers, 2025. URL https://arxiv.org/abs/2502.09692.

Neil Ashton, Charles Mockett, Marian Fuchs, Louis Fliessbach, Hendrik Hetmann, Thilo Knacke, Norbert Schonwald, Vangelis Skaperdas, Grigoris Fotiadis, Astrid Walle, et al. DrivAerML: High-fidelity computational fluid dynamics dataset for road-car external aerodynamics. *arXiv preprint arXiv:2408.11969*, 2024.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Yifan Chen, Qi Zeng, Heng Ji, and Yun Yang. Skyformer: Remodel self-attention with Gaussian kernel and Nyström method. *Advances in Neural Information Processing Systems Workshop on Efficient Natural Language and Speech Processing*, 2021.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Woojin Cho, Kookjin Lee, Donsub Rim, and Noseong Park. Hypernetwork-based meta-learning for low-rank physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11219–11231, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/24f8dd1b8f154f1ee0d7a59e368eccf3-Paper-Conference.pdf.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *arXiv preprint arXiv:2006.03236*, 2020.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. https://pytorch.org/blog/flash-decoding/, 2023. PyTorch blog, October 2023.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2302.05442.

Dongchen Han, Tianzhu Ye, Yizeng Han, Zhuofan Xia, Siyuan Pan, Pengfei Wan, Shiji Song, and Gao Huang. Agent attention: On the integration of softmax and linear attention. In *European Conference on Computer Vision (ECCV)*, 2024.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pp. 12556–12569. PMLR, 2023.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4246–4253, 2020. URL https://arxiv.org/abs/2010.04245.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver IO: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021a.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pp. 4651–4664. PMLR, 2021b.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 3744–3753, 2019.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*, 2023a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, et al. Geometry-informed neural operator for large-scale 3D PDEs. *Advances in Neural Information Processing Systems*, 36: 35836–35854, 2023b.

Hao Liu and Pieter Abbeel. Blockwise parallel transformers for large context models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. In *International Conference on Learning Representations*, 2024.

Qibang Liu, Weiheng Zhong, Hadi Meidani, Diab Abueidda, Seid Koric, and Philippe Geubelle. Geometry-informed neural operator transformer, 2025. URL https://arxiv.org/abs/2504.19452.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(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 *Forty-second International Conference on Machine Learning*, 2025.

Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. Luna: Linear unified nested attention. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

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. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1710.03740.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Pablo Miralles-González, Javier Huertas-Tato, Alejandro Martín, and David Camacho. On the locality bias and results in the long range arena. *arXiv preprint arXiv:2501.14850*, 2025.

Sepehr Mousavi, Shizheng Wen, Levi Lingsch, Maximilian Herde, Bogdan Raonić, and Siddhartha Mishra. RIGNO: A graph-based framework for robust and accurate operator learning for PDEs on arbitrary domains, 2025. URL https://arxiv.org/abs/2501.19205.

NVIDIA PhysicsNeMo Team. Transformer models for external aerodynamics on irregular meshes, 2026. URL https://docs.nvidia.com/physicsnemo/26.03/physicsnemo/examples/cfd/external_aerodynamics/transformer_models/README.html. NVIDIA PhysicsNeMo Framework documentation. Accessed April 30, 2026.

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 Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

Vedant Puri, Yongjie Jessica Zhang, and Levent Burak Kara. FLARE: Fast low-rank attention routing engine, 2026. URL https://arxiv.org/abs/2508.12594.

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022a.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosFormer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*, 2022b.

Markus N. Rabe and Charles Staats. Self-attention does not need $O(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

Michael S. Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. TokenLearner: Adaptive space-time tokenization for videos. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. *arXiv preprint arXiv:2002.11296*, 2020.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models. In *International Conference on Machine Learning*, pp. 10183–10192. PMLR, 2021a.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse PDE problems. *arXiv preprint arXiv:2406.03923*, 2024.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. *arXiv preprint arXiv:2402.02366*, 2024.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 14138–14148, 2021.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL `https://arxiv.org/abs/1910.07467`.

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. *arXiv preprint arXiv:2402.04347*, 2024.

Hang Zhou, Haixu Wu, Haonan Shangguan, Yuezhou Ma, Huikun Weng, Jianmin Wang, and Mingsheng Long. Transolver-3: Scaling up transformer solvers to industrial-scale geometries. *arXiv preprint arXiv:2602.04940*, 2026.

A FLARE++ MIXER ARCHITECTURE

This appendix records the design of the FLARE++ mixer in more detail than Section 3.2: why synthesized routing queries are hard to optimize, how normalization and the fixed reference stabilize them, the near-identity initialization of the feature projections, and the shared synthesis keys and values. Table 2 lists every operation of FLARE, FLARE++, and the simplified variant in execution order. The simplified variant drops the fixed reference and gate and routes with $Q_h^{\mathrm{d}}(X)$ alone; Section A.8 defines it and reports the ablation. Throughout, heads are treated independently and $D = C/H$ is the head width.

A.1 Query synthesis

FLARE routes every token through $M \ll N$ latent queries that are learned once and held fixed across samples. FLARE++ instead adapts the routing queries to the current input: with learned seeds $\widetilde{Q}_h \in \mathbb{R}^{M\times D}$, learned fixed queries $Q_h^{\mathrm{f}} \in \mathbb{R}^{M\times D}$, a per-head gate g_h, and projections $\widetilde{K}_h, \widetilde{V}_h, K_h, V_h$ of X, one block computes

$$\begin{aligned} Q_h^{\mathrm{d}} &= \mathrm{SDPA}(\widetilde{Q}_h, \widetilde{K}_h, \widetilde{V}_h), & Q_h &= \mathcal{N}_0(Q_h^{\mathrm{f}}) + g_h\, Q_h^{\mathrm{d}}, \\ Z_h &= \mathrm{SDPA}(Q_h, K_h, V_h), & Y_h &= \mathrm{SDPA}(K_h, Q_h, Z_h), \end{aligned} \tag{11}$$

as in equation 6–equation 8, with $\widetilde{V}_h = \widetilde{K}_h$ in the default configuration (Section A.6) and with the per-head normalizations of Section 3.2 introduced below one at a time. The first call synthesizes an input-dependent correction to the M routing vectors; the second gathers the input into M latents through them, and the third scatters the latents back to the N input locations. This appendix keeps the general form with a separate $\widetilde{V}_h$, since the simplified variant of Section A.8 uses it.

A.2 Optimization difficulty of synthesized routing queries

The principal optimization challenge in FLARE++ is that the output of the first attention call is the query of the second. Writing the synthesis attention as

$$Q_h^{\mathrm{d}} = \widetilde{A}_h \widetilde{V}_h, \qquad \widetilde{A}_h = \mathrm{softmax}\left(\frac{\widetilde{Q}_h \widetilde{K}_h^\top}{\sqrt{D}}\right), \tag{12}$$

the gather weights of the encoder are $A_h = \mathrm{softmax}(Q_h K_h^\top/\sqrt{D})$, so updates to the synthesis branch also change the scores used by the value-gathering encoder.

This is a difficult problem at initialization. The seeds $\widetilde{Q}_h$ and the projections producing $\widetilde{K}_h$ and $\widetilde{V}_h$ initially carry little task-specific structure, so $\widetilde{A}_h$ is weakly structured and often close to diffuse, while the values it aggregates have not yet learned a meaningful basis. The synthesized queries

$$Q_{h,m}^{\mathrm{d}} = \sum_{n=1}^{N} \widetilde{A}_{h,mn}\, \widetilde{V}_{h,n} \tag{13}$$

can therefore change substantially from one optimization step to the next early in training. This nonstationarity matters because Q_h^{d} is not merely an intermediate feature: it enters the routing query Q_h, whose direction sets the next attention logits through $Q_h K_h^\top$, so a change in the first call changes the coordinate system in which the second call routes. The encoder must learn useful routing directions while the routing vectors themselves are moving. We found that controlling this early-training instability explicitly is important for optimizing FLARE++ reliably. FLARE++ does so in two ways: it normalizes the inputs of the synthesis call, and it blends the synthesized correction with a fixed learned reference.

A.3 Normalizing the synthesis call

The first stabilization applies query–key normalization (Henry et al., 2020; Dehghani et al., 2023) to the inputs of the synthesis call,

$$Q_h^{\mathrm{d}} = \mathrm{SDPA}\big(\mathcal{N}_a(\widetilde{Q}_h), \mathcal{N}_0(\widetilde{K}_h), \widetilde{V}_h\big), \tag{14}$$

so that the synthesis logits $\widetilde{S}_{h,mn} = \mathcal{N}_a(\widetilde{Q}_h)_m^\top \mathcal{N}_0(\widetilde{K}_h)_n/\sqrt{D}$ no longer inherit uncontrolled variation in the magnitudes of the seeds and keys. The seeds carry a learned scale, which sets the temperature of the synthesis logits; the keys do not, for the reason given in Section A.4. Early changes in the attention distribution are then driven by changes in feature direction rather than by simultaneous changes in direction and scale. We use this normalization in all reported FLARE++ and simplified FLARE++ runs to stabilize the synthesis logits. The main text writes RMSNorm (Zhang & Sennrich, 2019) throughout; in full precision the same placement uses LayerNorm, whose affine-free output also has unit RMS, so the bounds below are unchanged. LayerNorm was markedly more sensitive to FP16 arithmetic in this setting, whereas RMSNorm was reliable, so every mixed-precision run uses RMSNorm. The normalization acts on the inputs to the synthesis call and not on its output Q_h^{d}: as Section A.4 argues, the magnitude of a synthesized query carries information that should not be discarded.

A.4 FIXED REFERENCE AND GATE

The second stabilization keeps FLARE's learned routing basis and lets the synthesis call learn an input-dependent correction to it. FLARE++ combines the synthesized query $Q_h^{\mathrm{d}}(X)$ with an independent learned set of fixed queries $Q_h^{\mathrm{f}} \in \mathbb{R}^{M\times D}$,

$$Q_h(X) = \mathcal{N}_0(Q_h^{\mathrm{f}}) + g_h\, Q_h^{\mathrm{d}}(X), \qquad g_h = \sigma(\gamma_h) \in (0,1), \tag{15}$$

as in equation 7. The gate holds one learned scalar γ_h per head, broadcast over latent queries and channels. In the limit $g_h \to 0$, the queries become input-independent; the normalization choices still differ from FLARE. We also trained FLARE with the same query–key normalization applied to its fixed queries and routing keys and saw no improvement. Increasing g_h admits a larger input-dependent correction. We initialize $\gamma_h = 0.25$, so $g_h = \sigma(0.25) \approx 0.56$: the model starts with a substantial synthesized contribution rather than collapsing to FLARE, while the fixed component provides a stable routing anchor from the first step. The blend is evaluated in FP32 and cast back to the model datatype, which costs $\mathcal{O}(MC)$ per block against the $\mathcal{O}(NC^2)$ projections and $\mathcal{O}(NMC)$ attention calls, and so adds negligible overhead.

Why the synthesized query is not normalized. A natural alternative is to normalize Q_h^{d} separately before blending it with Q_h^{f}. We found this undesirable. By equation 13, each synthesized query is a convex combination of the value rows selected by the synthesis attention. Early in training both $\widetilde{A}_h$ and $\widetilde{V}_h$ evolve rapidly, so both the direction and the magnitude of Q_h^{d} vary considerably, and normalizing the aggregate would remove the magnitude entirely. A small magnitude need not signal a numerical failure: it arises naturally when the attended value vectors point in inconsistent directions and partially cancel, while a large magnitude arises when they agree. The magnitude of Q_h^{d} therefore encodes how coherently the input supports a routing direction, and normalizing it would rescale a weak or poorly aligned correction to the same nominal size as a strongly supported one, converting a weak routing signal into a full-strength update. We instead control the scale of Q_h^{d} before aggregation, by normalizing the individual value vectors.

Bounding the synthesized query through value normalization. FLARE++ applies an affine-free RMS normalization $\mathcal{N}_0$, with no learned scale or bias, to the synthesis values, so that

$$Q_{h,m}^{\mathrm{d}} = \sum_{n=1}^{N} \widetilde{A}_{h,mn}\, \mathcal{N}_0(\widetilde{V}_h)_n, \qquad \mathrm{RMS}\left(\mathcal{N}_0(\widetilde{V}_h)_n\right) = 1. \tag{16}$$

Since every row of $\widetilde{A}_h$ is a probability distribution and $\mathrm{RMS}(\cdot) = \|\cdot\|_2/\sqrt{D}$ is a norm up to a constant, the triangle inequality gives

$$\mathrm{RMS}\left(Q_{h,m}^{\mathrm{d}}\right) \le \sum_{n=1}^{N} \widetilde{A}_{h,mn}\, \mathrm{RMS}\left(\mathcal{N}_0(\widetilde{V}_h)_n\right) = 1, \qquad \text{equivalently} \qquad \left\|Q_{h,m}^{\mathrm{d}}\right\|_2 \le \sqrt{D}. \tag{17}$$

This gives the desired scale control without normalizing the aggregated query. The distinction matters: normalizing Q_h^{d} would force every latent correction to the same magnitude, whereas normalizing the constituent values only imposes an upper bound on the convex aggregate, and Q_h^{d} remains free to be small when the attended directions disagree. The fixed component is normalized by the same affine-free $\mathcal{N}_0$ for the complementary reason: its role is a stable anchor, and without a learned scale its RMS is exactly one throughout training. Combining the two, every row of the routing query in equation 15 satisfies $1 - g_h \le \mathrm{RMS}(Q_{h,m}) \le 1 + g_h$, with $g_h < 1$. The blend thus pairs a controlled fixed reference with a bounded but magnitude-preserving displacement, and can be read as an input-dependent deformation of FLARE's latent routing basis in which each head adapts its routing directions to the current input while staying anchored to a stable learned coordinate system. The routing keys $K_h = \mathcal{N}_a(XW_{K,h})$ are normalized with a learned scale: with the query magnitude controlled as above, that scale sets the temperature of the routing logits $Q_h K_h^\top/\sqrt{D}$, as the seed scale does for the synthesis logits.

A.5 NEAR-IDENTITY FEATURE PROJECTIONS

A second optimization choice concerns the projections producing $\widetilde{K}_h$, $\widetilde{V}_h$, K_h, and V_h. Rather than arbitrary random linear maps, we initialize them close to the identity,

$$W = I + \Delta W, \qquad \Delta W_{ij} \sim \mathrm{TruncNormal}(0,\, 0.02), \tag{18}$$

with zero biases, so that $K_h \approx X_h$ and $\widetilde{K}_h \approx X_h$ at initialization, and likewise for the value projections. All three mixers use this initialization for every feature projection. This initialization is particularly useful for FLARE++ because the quality of the synthesized queries depends on the representation produced by an upstream attention call. With a conventional random projection, the encoder initially sees routing keys K_h in an arbitrary rotated and rescaled basis while the synthesis call tries to learn useful queries in another, independently evolving basis. Near-identity initialization preserves the input representation at the start of training, so the routing score $Q_h K_h^\top \approx Q_h X_h^\top$ provides a direct learning signal between the synthesized queries and the existing input features from the first optimization steps. The same holds for $\widetilde{K}_h \approx X_h$, which lets the synthesis call begin by selecting among meaningful input features rather than among the outputs of an unrelated random transformation. The projections remain fully learnable, so the initialization restricts nothing about the final representation; it only provides a well-conditioned starting point.

A.6 Sharing synthesis keys and values

We find little or no loss in accuracy from sharing the synthesis keys and values, $\widetilde{V}_h = \widetilde{K}_h$, under which

$$Q_h^{\mathrm{d}} = \mathrm{softmax}\left(\frac{\mathcal{N}_a(\widetilde{Q}_h)\,\mathcal{N}_0(\widetilde{K}_h)^\top}{\sqrt{D}}\right)\mathcal{N}_0(\widetilde{K}_h). \tag{19}$$

Each synthesized query is then an attention-weighted convex combination of normalized input feature directions, and its magnitude reflects their directional agreement, which further motivates preserving the magnitude of Q_h^{d} rather than renormalizing it after aggregation. The additional projection, seeds, and gates increase the mixer parameter count from FLARE's $3C^2 + MC$ to $4C^2 + 2MC + H$ per block, excluding biases and normalization parameters. Sharing removes one $C \to C$ projection, saving $\mathcal{O}(NC^2)$ computation and $\mathcal{O}(NC)$ activation storage per block while preserving the behaviour of FLARE++ in our experiments, so shared synthesis keys and values are the default FLARE++ configuration.

A.7 Operations in one block

Table 2 lists, in execution order, every projection, normalization, and attention call inside one block of FLARE, FLARE++, and simplified FLARE++ at the default mixed-precision settings used for every FP16 run in the paper, taken from the reference implementation. All three share the block structure around the mixer: a pre-normalization of the residual stream, the mixer, a residual addition, and a normalized feed-forward update. The differences are confined to the routing scores: how the routing queries are formed and which per-head query and key tensors are normalized; the value, output, and feed-forward rows are identical. Simplified FLARE++ normalizes only the synthesis seeds and keys; FLARE++ also normalizes the fixed query and the routing keys, shares the synthesis values with the synthesis keys, and accumulates its gated blend in FP32. Each per-head normalization acts on the head dimension D and carries D parameters when it is affine.

A.8 Simplified FLARE++

The simplified variant routes with the synthesized query alone,

$$Q_h(X) = Q_h^{\mathrm{d}}(X) = \mathrm{SDPA}\big(\mathcal{N}_a(\widetilde{Q}_h), \mathcal{N}_a(\widetilde{K}_h), \widetilde{V}_h\big), \tag{20}$$

with no fixed reference, no gate, a separate and unnormalized value projection $\widetilde{V}_h = X\widetilde{W}_{V,h}$, a learned scale on the synthesis keys, and no normalization of the routing keys (Table 2). Its routing query therefore carries no magnitude bound. It has $5C^2 + MC$ mixer parameters and costs $\mathcal{O}(N(5C^2 + 3MC))$ per block. It was our first formulation, and we keep it as the ablation that isolates what the fixed reference adds.

Accuracy. Table 3 repeats the standard-benchmark comparison of Table 1 for both variants. In FP32 the two are interchangeable. The simplified variant is lower on Darcy and Pipe and FLARE++ on Elasticity, Airfoil, and DrivAerML-40K, and averaged over the five benchmarks the simplified variant reduces FLARE's error by 24% and FLARE++ by 25%. On the (M, B) grid of Section 5.1, both improve on FLARE in all 21 cells; the simplified variant is lower in eight of the nine Elasticity cells, by 3–13%, and the two split the twelve Darcy cells evenly.

Table 2: Operations in one mixer block, in execution order, for the three mixers in mixed precision with RMSNorm. X is the block input, $\mathcal{N}_a$ an RMSNorm over the head dimension with a learned scale, $\mathcal{N}_0$ one without, and σ the logistic function; "learned" marks a free parameter tensor and "–" an operation the mixer does not perform. Every projection is a plain linear map initialized near the identity (Section A.5). The gate logits γ_h are initialized at 0.25, so every gate starts at $\sigma(0.25) \approx 0.56$. Parameter counts exclude biases and normalization scales.

Operation	**FLARE**	**FLARE++**	**Simplified FLARE++**
Block input	$X \leftarrow \mathrm{RMSNorm}(X)$	same	same
Synthesis seeds $\widetilde{Q}_h$	–	learned, $\mathcal{N}_a$	learned, $\mathcal{N}_a$
Synthesis keys $\widetilde{K}_h$	–	$X\widetilde{W}_{K,h}, \mathcal{N}_0$	$X\widetilde{W}_{K,h}, \mathcal{N}_a$
Synthesis values $\widetilde{V}_h$	–	shared, $\widetilde{V}_h = \widetilde{K}_h$	$X\widetilde{W}_{V,h}$, no norm
Synthesized query Q_h^{d}	–	$\mathrm{SDPA}(\widetilde{Q}_h, \widetilde{K}_h, \widetilde{V}_h)$	$\mathrm{SDPA}(\widetilde{Q}_h, \widetilde{K}_h, \widetilde{V}_h)$
Fixed query Q_h^{f}	learned, no norm	learned, $\mathcal{N}_0$	–
Routing query Q_h	Q_h^{f}	$Q_h^{\mathrm{f}} + \sigma(\gamma_h)\, Q_h^{\mathrm{d}}$ in FP32	Q_h^{d}, no norm
Routing keys K_h	$XW_{K,h}$, no norm	$XW_{K,h}, \mathcal{N}_a$	$XW_{K,h}$, no norm
Values V_h	$XW_{V,h}$, no norm	same	same
Encoder	$Z_h = \mathrm{SDPA}(Q_h, K_h, V_h)$	same	same
Decoder	$Y_h = \mathrm{SDPA}(K_h, Q_h, Z_h)$	same	same
Output	YW_O	same	same
Feed-forward	$X + \mathrm{FFN}(\mathrm{RMSNorm}(X))$	same	same
Projections	3	4	5
SDPA calls	2	3	3
Per-head norms	0	4	2
Learned latents	Q^{f}	$Q^{\mathrm{f}}, \widetilde{Q}, \gamma$	$\widetilde{Q}$
Mixer parameters	$3C^2 + MC$	$4C^2 + 2MC + H$	$5C^2 + MC$

Table 3: Test relative L^2 error (in %) of FLARE, FLARE++, and simplified FLARE++ on the standard PDE benchmarks under the protocol of Table 1; single seed, FP32.

Model	**Elasticity**	**Darcy**	**Airfoil**	**Pipe**	**DrivAerML-40K**
FLARE	0.64	0.76	0.57	0.51	7.22
Simplified FLARE++	0.38	0.59	0.52	0.34	6.04
FLARE++	0.37	0.62	0.50	0.35	5.80

Why the fixed reference. Three properties separate the variants where the small FP32 benchmarks do not. First, FLARE++ contains FLARE as the $g_h \to 0$ limit and starts training from a stable learned routing basis, whereas the simplified variant must learn useful routing directions while its only routing vectors are moving (Section A.2). Second, FLARE++ is far less sensitive to mixed precision. In the precision sweep of Section D.2, its FP16 error stays within 8% of its FP32 error on Elasticity, Darcy, and Airfoil, against up to 23% for the simplified variant; on Pipe, the simplified variant's FP16 error reaches $3.5\times$, $4.8\times$, and $12.9\times$ its FP32 error at $B = 2, 4, 8$ while FLARE stays within $1.4\times$. Every large-scale run in the paper is in FP16, so this property decides which variant can be used at scale. Third, sharing the synthesis keys and values removes one projection: under the protocol of Figure 6, FLARE++ costs 1.2–$1.3\times$ FLARE's step time against 1.25–$1.5\times$ for the simplified variant, 12–20% less at $M = 64$, at the same peak memory.

B MULTI-GPU CONTEXT PARALLELISM

Table 4: Cost of the distributed FLARE encoder, per rank and per invocation, in the accounting of Figure 1 ($C = HD$, so $HMD = MC$). Communication counts the values a rank contributes to a collective; local compute counts arithmetic on tensors the rank already holds. The encoder collectives communicate only latent quantities: $\overline{K}_r$ and $\overline{V}_r$ stay on the shard that owns those keys and values. The accounting excludes synchronization of replicated projection and backbone parameter gradients.

(a) Forward pass. The first row, the fused local attention over the rank's own shard, is the only N-dependent entry.

Step	Collective	Communication	Local compute
$Z_r, L_r = \mathrm{SDPA}(Q, K_r, V_r)$	—	—	$\mathcal{O}(N_r MHD)$
$L_{\max} = \max_r L_r$	all-reduce max	$\mathcal{O}(HM)$	—
$d_r = \exp(L_r - L_{\max})$	—	—	$\mathcal{O}(HM)$
$U_r = d_r Z_r$	—	—	$\mathcal{O}(HMD)$
$d = \sum_r d_r$	all-reduce sum	$\mathcal{O}(HM)$	—
$U = \sum_r U_r$	all-reduce sum	$\mathcal{O}(HMD)$	—
$Z = U/d$	—	—	$\mathcal{O}(HMD)$
$L = L_{\max} + \log d$	—	—	$\mathcal{O}(HM)$
Encoder total	3 collectives	$\mathcal{O}(HM(D+2))$	$\mathcal{O}(N_r MHD)$

(b) Backward pass for the general two-output derivative; the zero log-normalizer cotangent needs no reduction in FLARE or FLARE++. The upper block differentiates the merge; the lower block denotes a local attention backward accepting cotangents of both the output and log-normalizer. It returns $\overline{K}_r$, $\overline{V}_r$, and $\overline{Q}^{(r)}$ without materializing S_r; the text describes fused-backward reuse when $\overline{L} = 0$.

Step	Collective	Communication	Local compute
$\overline{Z} = \sum_r \overline{Z}^{(r)}$	all-reduce sum	$\mathcal{O}(HMD)$	—
$\overline{L} = \sum_r \overline{L}^{(r)}$	all-reduce sum	$\mathcal{O}(HM)$	—
$\overline{Z}_r = \exp(L_r - L)\,\overline{Z}$	—	—	$\mathcal{O}(HMD)$
$\overline{L}_r = \exp(L_r - L)\left(\overline{L} + \langle \overline{Z}, Z_r - Z \rangle_D\right)$	—	—	$\mathcal{O}(HMD)$
$\overline{Q}^{(r)}, \overline{K}_r, \overline{V}_r = \mathrm{SDPA}_{\mathrm{bwd}}(\overline{Z}_r, \overline{L}_r)$	—	—	$\mathcal{O}(N_r MHD)$
$\overline{Q} = \sum_r \overline{Q}^{(r)}$	all-reduce sum	$\mathcal{O}(HMD)$	—
Encoder total	3 collectives	$\mathcal{O}(HM(2D+1))$	$\mathcal{O}(N_r MHD)$

The construction in this appendix is the log-sum-exp merge of partial softmax outputs that underlies online softmax (Milakov & Gimelshein, 2018), memory-efficient and fused attention (Rabe & Staats, 2021; Dao et al., 2022), blockwise and Ring Attention (Liu & Abbeel, 2023; Liu et al., 2024), and split-K Flash-Decoding (Dao et al., 2023), applied to a cross-attention whose queries are replicated and whose keys and values are token-sharded. We claim no methodological novelty for it; the appendix fixes notation, records the numerically stable reduction schedule and the backward pass that the implementation uses, and accounts for the communication of FLARE and FLARE++.

B.1 TOKEN-SHARDED REPRESENTATION

Let the input to a token-mixing block be $X \in \mathbb{R}^{N \times C}$, with $C = HD$ the channel width for H heads of dimension D; we omit the batch dimension, along which every operation below acts independently. The context-parallel group contains R ranks, and rank r owns

$$X_r \in \mathbb{R}^{N_r \times C}, \qquad X = [X_0; \ldots; X_{R-1}], \qquad \sum_{r=0}^{R-1} N_r = N. \tag{21}$$

Shards need not be spatially contiguous, since FLARE assigns no meaning to token order, and unequal N_r are allowed as long as local masks exclude padding from the attention normalization.

All operations that act independently on tokens remain local: rank r computes

$$K_r = f_K(X_r), \qquad V_r = f_V(X_r), \qquad K_r, V_r \in \mathbb{R}^{H \times N_r \times D}. \tag{22}$$

and the input and output projections, normalization, feed-forward networks, residual connections, and prediction head use the same sharding. The learned query or seed tensors, of shape $\mathbb{R}^{H \times M \times D}$, are replicated across the group.

B.2 Exact distributed FLARE encoder

Consider a FLARE encoder with a replicated query tensor $Q \in \mathbb{R}^{H \times M \times D}$, stacking the per-head Q_h, and token-sharded keys and values. On the concatenated token sequence, the desired result is

$$Z = \mathrm{softmax}\big(sQK^\top\big)\, V, \qquad K = [K_0; \ldots; K_{R-1}], \qquad V = [V_0; \ldots; V_{R-1}], \tag{23}$$

where $s = D^{-1/2}$ and $Z \in \mathbb{R}^{H \times M \times D}$. The global K and V are notation only and are never assembled.

Rank r evaluates the encoder on its local token shard,

$$S_r = sQK_r^\top \in \mathbb{R}^{H \times M \times N_r}, \tag{24}$$

$$L_r = \log \sum_{j=1}^{N_r} \exp(S_{r,j}) \in \mathbb{R}^{H \times M}, \tag{25}$$

$$Z_r = \mathrm{softmax}(S_r) V_r \in \mathbb{R}^{H \times M \times D}. \tag{26}$$

Because Z_r is normalized only over shard r, the local outputs cannot be averaged uniformly. With the global rowwise log-normalizer and the attention mass assigned to rank r,

$$L = \log \sum_{r=0}^{R-1} \exp(L_r), \qquad \alpha_r = \exp(L_r - L), \tag{27}$$

the exact global output is

$$Z = \sum_{r=0}^{R-1} \alpha_r Z_r, \tag{28}$$

since α_r restores the fraction of each globally normalized attention row that falls on shard r.

B.3 Stable reductions and fused local attention

Neither log-sum-exp above may be evaluated literally in finite precision, since both exponentiate unbounded arguments. Both are instead evaluated through the exact shift identity

$$\log \sum_i \exp(x_i) = m + \log \sum_i \exp(x_i - m), \tag{29}$$

with $m = \max_i x_i$, so that every exponential lies in $(0, 1]$. For the local reduction equation 25 the fused SDPA primitive already does this: its online softmax carries a running rowwise maximum and rescales its accumulators, so the Z_r and L_r it returns are computed in shifted form. For the cross-rank reduction equation 27 we apply the shift with $m = L_{\max}$ in the following schedule. An all-reduce maximum forms

$$L_{\max} = \max_r L_r \in \mathbb{R}^{H \times M} \tag{30}$$

at $\mathcal{O}(HM)$ values; each rank then computes, without communication,

$$d_r = \exp(L_r - L_{\max}), \qquad U_r = d_r Z_r, \tag{31}$$

with d_r broadcast over the value dimension; two all-reduce sums form

$$d = \sum_{r=0}^{R-1} d_r, \qquad U = \sum_{r=0}^{R-1} U_r, \tag{32}$$

at $\mathcal{O}(HM)$ and $\mathcal{O}(HMD)$ values; and the exact output and log-normalizer follow elementwise on every rank,

$$Z = \frac{U}{d}, \qquad L = L_{\max} + \log d. \tag{33}$$

Since $d = \exp(-L_{\max}) \sum_r \exp(L_r)$, $L_{\max} + \log d$ reproduces L of equation 27 and U/d reproduces equation 28. Table 4a lists the schedule with the cost of each step. No step of the merge touches N, only the sum for U carries the value dimension, and the merge arithmetic is elementwise on tensors the rank already holds, independent of the local token count. FLARE performs one encoder invocation per mixer and FLARE++ two; neither all-gathers X, K, V, or Y, and the decoder adds no collective. Aggregate network traffic and latency still depend on R and on the collective implementation, which this per-rank accounting does not model.

A locally empty attention row must return $L_r = -\infty$ and $Z_r = 0$ rather than NaN, which the shift maps to $d_r = U_r = 0$; the merge is valid provided at least one shard holds a valid key for every row. The score tensor S_r is never materialized: a FlashAttention-backed SDPA primitive (Dao et al., 2022; Dao, 2024) streams the local keys and values and returns both Z_r and L_r. Exposing L_r is essential, because the output of an independently normalized local call alone cannot reconstruct the global result.

B.4 FLARE AND FLARE++ COMMUNICATION SCHEDULES

In FLARE, the fixed queries Q are replicated and one distributed encoder constructs the latent values,

$$Z = \mathrm{Enc}_{\mathrm{dist}}(Q, \{K_r\}, \{V_r\}), \tag{34}$$

which the reductions in equation 28 leave replicated; rank r then decodes locally,

$$Y_r = \mathrm{SDPA}(K_r, Q, Z), \tag{35}$$

with no communication, because the query for each output token is local and both latent tensors are replicated.

FLARE++ invokes the same distributed encoder twice. The first invocation constructs the synthesized routing queries,

$$Q(X) = \mathrm{Enc}_{\mathrm{dist}}\left(\widetilde{Q}, \{\widetilde{K}_r\}, \{\widetilde{V}_r\}\right), \tag{36}$$

where the learned latent seeds $\widetilde{Q}$ are replicated and the projected input tensors $\widetilde{K}_r, \widetilde{V}_r$ remain sharded. The second invocation gathers the physical values,

$$Z = \mathrm{Enc}_{\mathrm{dist}}\left(Q(X), \{K_r\}, \{V_r\}\right), \tag{37}$$

and each rank applies the local decode

$$Y_r = \mathrm{SDPA}(K_r, Q(X), Z). \tag{38}$$

FLARE thus uses one communicating encoder and one local decoder, FLARE++ two communicating encoders and one local decoder; the output stays token-sharded and enters the next block without reconstructing an N-token tensor.

B.5 BACKWARD PASS

The distributed encoder composes a purely local attention on rank r, which maps Q, K_r, V_r to the shard-normalized pair (Z_r, L_r), with the global merge equation 33, which maps $\{(Z_r, L_r)\}_r$ to the replicated pair (Z, L). Reverse mode follows the same decomposition: given the cotangents of Z and L, the merge produces the cotangents of each rank's Z_r and L_r. The local derivative must account for both outputs; an output-only SDPA backward does not expose this general interface. Handing the cotangent of Z to a local backward directly would differentiate R independently normalized attentions instead of the globally normalized one. We write $\overline{A} = \partial\mathcal{L}/\partial A$ for the cotangent of any forward tensor A with respect to the scalar loss $\mathcal{L}$.

Incoming gradient of the merge. The all-reduces in equation 33 leave every rank holding its own copy $Z^{[r]} = Z$, $L^{[r]} = L$, and rank r consumes its copy in the local decode equation 35. Replication is a fan-out in the computational graph, so its adjoint is a sum: writing $\overline{Z}^{(r)} = \partial\mathcal{L}/\partial Z^{[r]}$ and $\overline{L}^{(r)} = \partial\mathcal{L}/\partial L^{[r]}$ for the cotangents rank r computes from its own copy,

$$\overline{Z} = \sum_{r=0}^{R-1} \frac{\partial \mathcal{L}}{\partial Z^{[r]}} = \sum_{r=0}^{R-1} \overline{Z}^{(r)}, \qquad \overline{L} = \sum_{r=0}^{R-1} \overline{L}^{(r)}. \tag{39}$$

Because every rank needs $\overline{Z}$ and $\overline{L}$ in full, the reduction is an all-reduce sum of $\mathcal{O}(HMD)$ and $\mathcal{O}(HM)$ values. In FLARE and FLARE++ the decoder reads Z alone, so $\overline{L} = 0$ in practice and its all-reduce can be omitted; we carry it so that the merge is differentiable in both outputs and may be composed hierarchically.

Differentiating the merge. In its unshifted form the merge is

$$Z = \sum_{r=0}^{R-1} Z_r \exp(L_r - L), \qquad L = \log \sum_{r=0}^{R-1} \exp(L_r), \tag{40}$$

a convex combination of the local outputs with weights $\exp(L_r - L)$, already computed in the forward pass as d_r/d. Everything below is rowwise over (h, m). Rank r enters equation 40 only through Z_r and L_r, with partial derivatives

$$\frac{\partial Z}{\partial Z_r} = \exp(L_r - L)\, I_D, \tag{41}$$

$$\frac{\partial L}{\partial L_r} = \exp(L_r - L), \tag{42}$$

$$\frac{\partial Z}{\partial L_r} = \exp(L_r - L)\,(Z_r - Z)\,, \tag{43}$$

the third following from $\partial \exp(L_q - L)/\partial L_r = \exp(L_q - L)\,(\delta_{qr} - \exp(L_r - L))$: raising L_r slides the convex combination from Z towards Z_r. The chain rule gives the local cotangents

$$\overline{Z}_r = \exp(L_r - L)\,\overline{Z} \in \mathbb{R}^{H\times M\times D}, \tag{44}$$

$$\overline{L}_r = \underbrace{\exp(L_r - L)\,\overline{L}}_{\text{through } L} + \underbrace{\exp(L_r - L)\,\langle \overline{Z}, Z_r - Z\rangle_D}_{\text{through } Z} \in \mathbb{R}^{H\times M}, \tag{45}$$

where $\langle A, B\rangle_D$ is the inner product over the value dimension, one scalar per row. Shard r receives the fraction $\exp(L_r - L)$ of the output cotangent that its attention mass earned in the forward pass, and $\overline{L}_r$ collects the two paths by which L_r reaches the loss, through the merged normalizer and through the reweighting of shard r against the others. Neither expression communicates, since rank r already holds L_r, Z_r from the forward pass and $Z, L, \overline{Z}, \overline{L}$ from the reductions.

Local backward. Rank r then runs the single-device backward with $(\overline{Z}_r, \overline{L}_r)$ as incoming gradients and the saved Q, K_r, V_r, Z_r, L_r. Composing it with 44 and 45 yields a closed form in the globally normalized probability block $P_r = \exp(S_r - L)$, with L broadcast along the local keys. Writing $c = \langle \overline{Z}, Z\rangle_D$, the gradients are

$$\overline{V}_r = P_r^\top \overline{Z}, \tag{46}$$

$$\overline{S}_r = P_r \odot \left(\overline{Z} V_r^\top - c + \overline{L}\right), \tag{47}$$

$$\overline{K}_r = s\,\overline{S}_r^\top Q, \tag{48}$$

$$\overline{Q} = s \sum_{r=0}^{R-1} \overline{S}_r K_r, \tag{49}$$

with c and $\overline{L}$ broadcast over the N_r local keys. The composition collapses because $\exp(S_{r,j} - L_r)\exp(L_r - L) = \exp(S_{r,j} - L)$: the shard-normalized backward driven by $(\overline{Z}_r, \overline{L}_r)$

equals the globally normalized backward driven by $\overline{Z}$. A conventional local backward on $\overline{Z}$ would instead use shard-normalized probabilities and $c = \langle \overline{Z}, Z_r \rangle_D$, dropping the between-shard term of equation 45. For $\overline{L} = 0$, an implementation can reuse a fused SDPA backward if it accepts the merged Z and L in place of the local pair, reconstructing P_r without materializing the score matrix. For nonzero $\overline{L}$, the additional $P_r \odot \overline{L}$ term must also be included; this requires the generalized derivative above. Writing $\overline{Q}^{(r)} = s\,\overline{S}_r K_r$, the query gradient $\overline{Q} = \sum_r \overline{Q}^{(r)}$ is the only collective in this stage, one all-reduce sum of $\mathcal{O}(HMD)$ values, while $\overline{K}_r$ and $\overline{V}_r$ stay on the shard that owns them; gradients of replicated parameters are synchronized across the data- and context-parallel groups as usual. Table 4b lists three backward collectives for the general two-output derivative, or two when the zero $\overline{L}$ reduction is omitted. All act on N-independent latent quantities, with the only N-dependent work the fused local backward.

For FLARE++, each rank's decoder contributes to the cotangent of the replicated $Q(X)$; these contributions are summed across the group as in equation 39 and added to the reduced $\overline{Q}$ from the physical-value encoder. The resulting global query cotangent is passed once to the query-synthesis encoder, without reducing it a second time, and propagated by the same two-stage construction to local gradients for $\widetilde{K}_r$, $\widetilde{V}_r$ and a synchronized gradient for the seeds $\widetilde{Q}$. Each encoder invocation must save its own output and log-normalizer.

B.6 Measured scaling

Table 5 reports the measurements summarized in Section 3.3, taken on a single node of four NVIDIA H100 GPUs; we did not have access to more ranks, so the table does not speak to cross-node interconnects.

Table 5: Context-parallel strong scaling on NVIDIA H100 GPUs, FP16, at $C = 128$, $H = 8$, $B = 8$. Both columns are efficiencies normalized within a row against that row's single-rank measurement, so ideal scaling is 1.00 in each: E is the usual parallel efficiency in step time, and E_{mem} is the per-rank peak-memory reduction divided by the rank count. $M = 64$ and $M = 128$ agree to within 1.2% on efficiency and 0.1% on memory, so only $M = 128$ is shown. This is not a comparison between FLARE and FLARE++.

Model	N	$R = 1$		$R = 2$		$R = 4$	
		E	E_{mem}	E	E_{mem}	E	E_{mem}
FLARE	500K	Ref.	Ref.	0.98	0.97	0.94	0.97
	1M	Ref.	Ref.	1.01	0.98	0.98	0.97
FLARE++	500K	Ref.	Ref.	0.97	0.96	0.92	0.95
	1M	Ref.	Ref.	0.96	0.97	0.97	0.96

Efficiency is measured over 30 timed steps per configuration and normalized within each model against its single-rank step time; one entry exceeds unity by about a percent. The table reports timing estimates without uncertainty intervals. Within these measurements, sharding reduces per-rank peak memory and step time close to proportionally to the rank count. Memory is reported as a within-row ratio rather than in gigabytes because absolute peak memory is not comparable between this harness and the single-GPU harness of Figure 6, which is the sole source of absolute memory in this paper: the context-parallel harness reports a single-rank FLARE peak $1.89\times$ the single-GPU figure while agreeing on FLARE++ to 3% and on step time to 5%, a discrepancy whose cause remains unresolved. Within-row ratios describe scaling in the context-parallel harness, but do not establish why its absolute memory differs from the single-GPU harness.

C Dataset and protocol details

C.1 Benchmark datasets

Table 6 summarizes each benchmark: the task, the discretization, the feature counts, and the splits. The four standard benchmarks use the splits and resolutions of Transolver (Wu et al., 2024), and

DrivAerML-40K uses the split of the FLARE study (Puri et al., 2026); Appendix D of that study describes each dataset and its source in full. Figure 2 shows one test case of each two-dimensional benchmark, and Figure 3 shows a full-surface DrivAerML geometry.

Table 6: Benchmark tasks. Point counts, feature counts, and splits of the first five rows are reproduced from Table 4 of the FLARE study (Puri et al., 2026); the last row is the full-surface task of Section 4.2.

Benchmark	Task	Grid	Points	In/Out features	Train/Test cases
Elasticity	Stress field of an elastic solid from the point cloud of its geometry	2D unstructured	972	2 / 1	1000 / 200
Darcy	Pressure of flow through a porous medium from its permeability field on an 85×85 grid	2D structured	7,225	1 / 1	1000 / 200
Airfoil	Mach number field around deformed NACA-0012 profiles on a 221×51 body-fitted mesh	2D structured	11,271	2 / 1	1000 / 200
Pipe	Horizontal velocity in pipes of varying centerline on a 129×129 mesh	2D structured	16,641	2 / 1	1000 / 200
DrivAerML-40K	Surface pressure of morphed DrivAer vehicles from a fixed 40K-cell subsample per geometry, used for both training and evaluation	3D unstructured	40,000	3 / 1	387 / 97
DrivAerML (full surface)	Surface pressure and wall shear stress from cell coordinates and normals; 100K random cells per training step, the full mesh at test	3D unstructured	8.2M	6 / 4	400 / 50

The two DrivAerML tasks. The paper uses DrivAerML (Ashton et al., 2024) in two settings whose numbers are not comparable. DrivAerML-40K, the last column of Table 1, follows the FLARE study (Puri et al., 2026): 40K surface cells are sampled once per geometry, the same fixed subsample serves for training and evaluation, the target is surface pressure, the split is $387/97$, and the metric is relative L^2 error. The full-surface task of Section 4.2 instead trains on a fresh random subsample of each geometry at every step and evaluates on the complete surface mesh. The full-surface DrivAerML experiment uses the split of Alkin et al. (2025): 400 training and 50 test geometries, with the 34 validation geometries unused. Each surface cell carries its coordinates, scaled to the training-split bounding box, and its unit normal as input; the target is the four-channel vector of mean pressure and mean wall shear stress, standardized per channel by the training-split mean and standard deviation. The training loss and the reported metric are the mean squared error of the standardized channels, averaged over cells and channels and multiplied by 100; the test value is computed on the complete 8.2M-cell surface of each test geometry and averaged over the test set. Every model uses the backbone of Table 7 with $C = 128$ and $H = 8$, the same optimizer, peak learning rate, one-cycle schedule, and 500 epochs as the standard benchmarks, FP16 execution with RMSNorm in place of LayerNorm, weight decay 10^{-4}, and context parallelism over two H100 GPUs. Each training step takes one geometry and a uniform random subset of 100K of its cells drawn without replacement, redrawn at every step; no stratified or strided partitioning of the mesh is used. Inference runs the full mesh through the model in a single forward pass under context parallelism, with no chunked decoding, for every model compared.

C.2 Training and model configuration

Table 7 gives the settings held fixed across all datasets and mixers, and Table 8 gives the per-dataset settings for the models we train under the matched backbone: full self-attention, Transolver,

Transolver++, Transolver-3, Luna, AB-UPT, FLARE, and FLARE++. AB-UPT uses $M = 512$ anchor points on every dataset. No hyperparameter is tuned per mixer. Set Transformer uses the same settings as a complete architecture, and PerceiverIO, GNOT, and LNO are reproduced from the FLARE study (Puri et al., 2026) under its configurations (Table 9). All author-run accuracy results reported here use a single seed. Experiments use NVIDIA H100 GPUs. FP32 runs use LayerNorm, while mixed-precision runs use RMSNorm following the stability behavior observed in our runs (Zhang & Sennrich, 2019; Micikevicius et al., 2018). The same choice applies to the per-head normalizations inside FLARE++, which are $\mathcal{N}_a$ on the seeds $\widetilde{Q}_h$ and the routing keys K_h and $\mathcal{N}_0$ on the shared $\widetilde{V}_h = \widetilde{K}_h$ and the fixed query Q_h^{f}; FLARE++ accumulates the gated sum in FP32 and initializes every gate logit at $\gamma_h = 0.25$, and the simplified variant of Section A.8 applies $\mathcal{N}_a$ to $\widetilde{Q}_h$ and $\widetilde{K}_h$ only (Table 2). Every feature projection is initialized near the identity, $W = I + \Delta W$ (Section A.5).

Table 7: Settings held fixed across all datasets and all matched-backbone mixers. The input and output projections follow the FLARE study (Puri et al., 2026).

Setting	**Value**
Architecture	
Channel width C	128
Attention heads H	8
Head width $D = C/H$	16
Input/output projection depth	2 layers, following Puri et al. (2026)
Output projection normalization	enabled
Block normalization	pre-norm LayerNorm
Block feed-forward network	GELU MLP
Feed-forward MLP ratio	2.0
Optimization	
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$
Peak learning rate	10^{-3}
Schedule	one-cycle
Exponential moving average	enabled
Execution	
Precision	FP32
Hardware	NVIDIA H100 GPU

Table 8: Per-dataset settings. The depth B is chosen to suit the size of each benchmark and is shared by every mixer on that dataset, as is the latent budget M (latent subsequence length, or slices for the Transolver family); AB-UPT uses $M = 512$ anchor points throughout. Full self-attention has no latent budget. Batch size is the global batch; all runs here use a single device.

Dataset	**Blocks B**	**Latents M**	**Batch size**	**Weight decay**	**Epochs**
Elasticity	8	64	2	10^{-5}	500
Darcy	8	128	2	10^{-5}	500
Airfoil	8	64	2	10^{-5}	500
Pipe	2	64	2	10^{-5}	500
DrivAerML-40K	4	64	1	10^{-2}	500

C.3 Comparison models

Table 9 lists every model of Table 1, how it is evaluated, its latent budget, and where it departs from Table 7 and Table 8; Table 10 gives the resulting parameter counts.

We evaluate Set Transformer and PerceiverIO as complete architectures because their block structure differs from the shared token-mixer backbone. Set Transformer's induced set attention block (ISAB) contains two cross-attention blocks, each with its own projections, residual paths, normalization, and

Table 9: Comparison models. Mixers are substituted into the shared residual backbone of Table 7; the latent budget M and the depth B take the per-dataset values of Table 8, and deviations are listed in the last column. Complete architectures replace the backbone.

Model	Evaluated as	Latent budget	Deviations from Tables 7–8
Full self-attention	Mixer	None	None; not run on Pipe or DrivAerML-40K owing to quadratic cost.
FLARE, FLARE++	Mixer	M latent queries	None.
Transolver, Transolver++, Transolver-3	Mixer	M slices	None; variants without convolution.
Luna	Mixer	M latent tokens	None.
AB-UPT	Mixer	512 anchors	Latent budget fixed at 512 anchors on every dataset; at 32–128 anchors its loss is markedly worse.
Set Transformer	Complete architecture; an ISAB replaces each block	M inducing points	None.
PerceiverIO	Complete architecture; reproduced from the FLARE study	512 latents	$C = 128$, $H = 8$, $B = 8$ latent self-attention blocks, MLP ratio 4 (Puri et al., 2026).
LNO	Complete architecture; reproduced from the FLARE study	256 latent modes	$C = 128$ (192 on Elasticity), $H = 8$, 4 latent self-attention blocks (8 on Airfoil and Pipe) (Puri et al., 2026).
GNOT	Complete architecture; reproduced from the FLARE study	None (linear attention with expert gating)	$C = 128$, $H = 8$, $B = 8$, MLP ratio 2, 3 experts (Puri et al., 2026).

Table 10: Parameter counts, in millions, of the author-run models of Table 1 at the settings of Table 7 and Table 8, including biases and normalization parameters, computed from the reference implementation. The models reproduced from the FLARE study (Puri et al., 2026) use that study's configurations and counts: PerceiverIO 1.87M, LNO 0.92M, and GNOT 4.88M.

Model	Elasticity	Darcy	Airfoil	Pipe	DrivAerML-40K
Full self-attention	1.16	1.16	1.16	0.36	0.63
Set Transformer	1.76	1.83	1.76	0.52	0.93
Luna	1.57	1.57	1.57	0.47	0.84
Transolver	1.04	1.05	1.04	0.34	0.57
Transolver++	0.92	0.94	0.92	0.30	0.51
Transolver-3	1.04	1.05	1.04	0.34	0.57
AB-UPT	1.16	1.16	1.16	0.36	0.63
FLARE	1.09	1.16	1.09	0.35	0.60
FLARE++	1.29	1.42	1.29	0.40	0.70

feed-forward network. PerceiverIO instead encodes into a latent sequence, processes it with latent self-attention, and decodes at the output. A comparison using only their attention operations would evaluate a different architectural adaptation. Here, Set Transformer replaces each attention-plus-feed-forward block with an ISAB, while PerceiverIO and LNO use the latent-processing depths of their reported configurations.

Reproducibility of FLARE. The FLARE entries in Table 1 differ from Table 1 of the FLARE study (Puri et al., 2026) because the two tables answer different questions. That table compares complete surrogate architectures at an approximately matched parameter count: FLARE uses residual

MLPs for its key and value projections and its feed-forward block, and reduces the channel width to $C = 64$ to offset their cost, on the hypothesis that mapping smooth fields to smooth fields benefits from deeper residual networks. Our comparison instead holds the residual depth, channel width, and feed-forward block fixed across mixers, uses linear key and value projections inside every mixer, and lets the parameter count vary (Table 10), so that the mixers are compared in isolation. The FLARE study reports the same kind of identical-backbone comparison in its Appendix G.1, and our FLARE results agree with those.

Transolver without convolution. Transolver supplements its physics-attention token mixer with convolution layers that aggregate over the structured grid (Wu et al., 2024). The FLARE study compares Transolver with and without these layers and reports a significant regression without them on Darcy, Airfoil, and Pipe (Puri et al., 2026). To keep the comparison isolated to the token mixer, we likewise remove the convolution layers from Transolver, Transolver++, and Transolver-3 (Table 9). Our Transolver results are in line with the Transolver results of the FLARE study under the same setting.

Reproducibility of Transolver++. The Transolver++ entries in Table 1 are our runs under the shared protocol, and they are less accurate than the results reported in its paper (Luo et al., 2025). We report the values we obtained. The difficulty of matching the published numbers with the official release and its stated hyperparameters is documented independently by the FLARE study (Puri et al., 2026), by AB-UPT (Alkin et al., 2025), and by the NVIDIA PhysicsNeMo documentation (NVIDIA PhysicsNeMo Team, 2026), whose external-aerodynamics experiments report no gains from Transolver++.

Reproducibility of Transolver-3. We also attempted to include Transolver-3 (Zhou et al., 2026) in the full-surface DrivAerML comparison of Section 4.2. The official Transolver-3 repository provides only an FP32 training recipe, and we know of no stable FP16 or BF16 recipe for it; every attempt to train it on DrivAerML in 16-bit precision diverged. Running it in FP32 would break the matched-precision protocol of that experiment and does not fit the memory budget of the full-surface evaluation, so Transolver-3 is omitted from Figures 5 and 8.

D Extended results and ablations

D.1 Random-seed variance

Table 11 repeats the FLARE and FLARE++ Elasticity runs of Table 1 with ten random seeds each. The FLARE++ test error has mean 0.39% and standard deviation 0.021%, a coefficient of variation of about 5%, and ranges from 0.37% to 0.43% across seeds. The FLARE test error has mean 0.71% and standard deviation 0.081%, a coefficient of variation of about 11%, and ranges from 0.58% to 0.86%. The two ranges do not overlap, so every FLARE++ seed is below every FLARE seed, and every FLARE++ seed also remains below the next-best efficient model on Elasticity in Table 1, Set Transformer at 0.57%. Table 1 reports these two means for Elasticity; its other entries are single-seed runs.

Table 11: Train and test relative L^2 error (in %) of FLARE and FLARE++ on Elasticity across ten random seeds. The standard deviation is the sample standard deviation.

Seed	0	1	2	3	4	5	6	7	8	9	**Mean**	**Std**
FLARE												
Train	0.383	0.460	0.521	0.466	0.491	0.528	0.664	0.449	0.576	0.496	0.503	0.077
Test	0.577	0.683	0.728	0.663	0.711	0.760	0.865	0.636	0.790	0.701	0.711	0.081
FLARE++												
Train	0.235	0.239	0.231	0.234	0.259	0.250	0.252	0.222	0.236	0.248	0.241	0.011
Test	0.368	0.395	0.378	0.383	0.431	0.408	0.394	0.369	0.369	0.411	0.391	0.021

D.2 MIXED-PRECISION SENSITIVITY

This section details the precision sweep summarized in Section 5.3. Figure 7 repeats the depth sweep in both precisions for full self-attention, FLARE, FLARE++, and the simplified variant of Section A.8, at the latent budget each dataset uses in Table 1, on the three benchmarks with a full self-attention run in both precisions; every number is a single seed. Pipe was also measured but is not plotted, since it has no FP32 full self-attention run.

The two execution settings change errors materially, while FLARE++ retains lower error than FLARE in every comparison. The largest increase in this sweep is for full self-attention on Darcy at $B = 2$, whose error rises by 82%; full self-attention on Airfoil at $B = 8$ instead improves by 30%. FLARE++ is the least precision-sensitive mixer on all three benchmarks: its FP16 error stays within 8% of its FP32 error at every depth, against ratios of 0.88–1.33 for FLARE, 0.95–1.23 for the simplified variant, and 0.70–1.82 for full self-attention. The row that does move is the reference: in FP32, FLARE++ is more accurate than full self-attention on Airfoil at $B = 8$ (0.497 against 0.683), and in FP16 the order reverses (0.497 against 0.480). In principle a synthesized query is the more exposed construction, since a low-precision perturbation of the synthesis softmax moves the queries defining both rank-M factors, whereas FLARE's queries are exact parameters. The simplified variant shows this exposure: its FP16 error is 1.12–$1.23\times$ its FP32 error on Elasticity, and on Pipe, which is not plotted, reaches $3.5\times$, $4.8\times$, and $12.9\times$ at $B = 2, 4, 8$ while FLARE stays within $1.4\times$. The fixed reference removes it on the three benchmarks plotted. The settings change normalization as well as precision (Section C.2), so this experiment measures their combined effect.

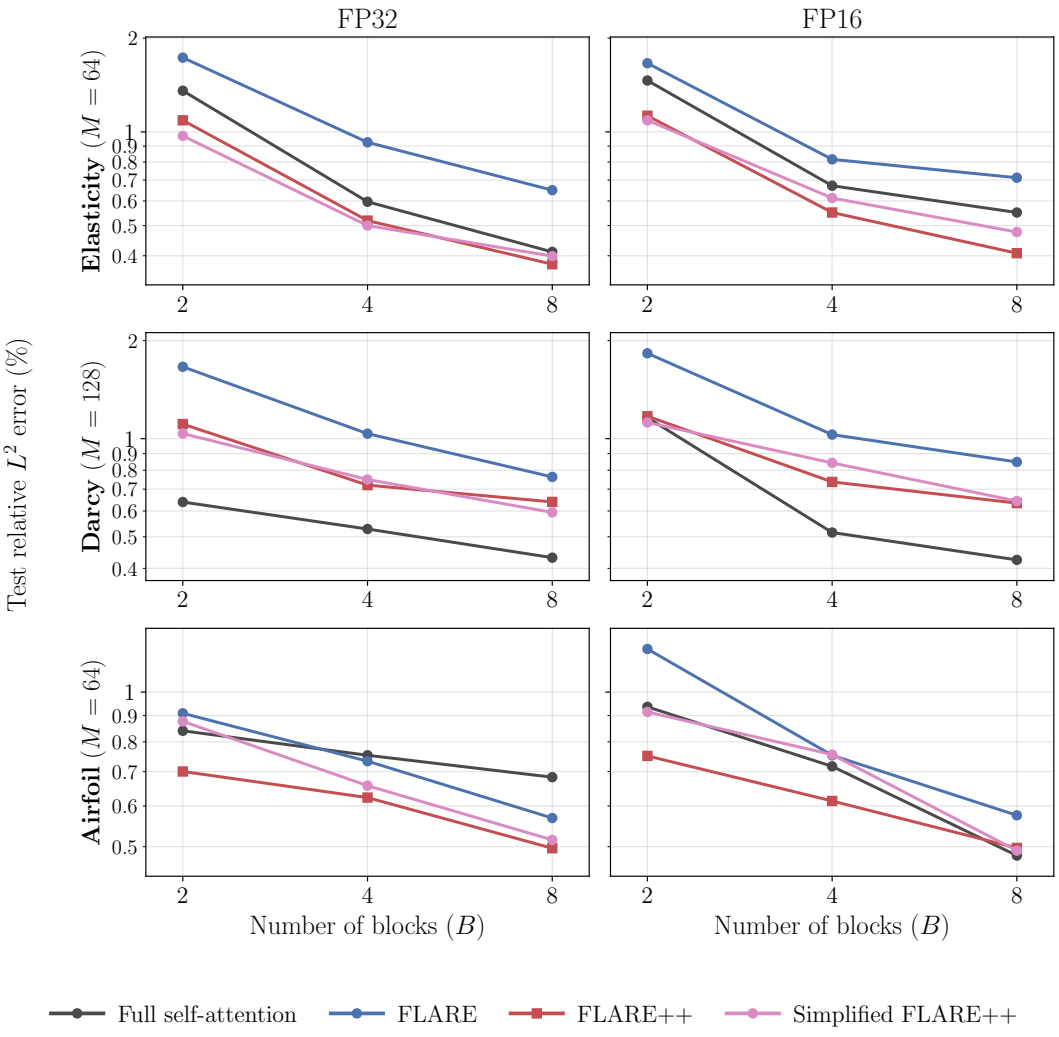


Figure 7: Test error against depth in FP32 (left) and FP16 (right), one row per benchmark and one line per token mixer, at each benchmark's latent budget from Table 1 ($M = 128$ on Darcy, $M = 64$ otherwise). The two panels in a row share a vertical axis, so precision sensitivity appears as a change in the shape or ordering of the curves rather than their position; axes are not shared across rows.

D.3 Single-GPU time and memory

Figure 6 reports forward-plus-backward time and peak memory of complete models at $B = 8$, $C = 128$, $H = 8$ in FP16 on one NVIDIA H100, with the number of tokens varied from 1K to 1M. Fused SDPA keeps every memory curve linear in N; full self-attention separates on time, and at 500K tokens is two orders of magnitude slower than FLARE++. FLARE++ costs 1.2–1.3× FLARE's step time and 1.18× its peak memory. The time ratio is stable beyond 100K tokens and is smaller than the $4/3$–$3/2$ increase in mixer arithmetic of Section 3.2 because the shared feed-forward network dilutes the mixer overhead. Transolver-3 is run as a matched-backbone mixer without the tiling of Zhou et al. (2026); at 500K tokens it needs 1.65× FLARE's memory at 64 slices and 2.47× at 128. Absolute peak memory is comparable only within this harness; Section B.6 explains why the context-parallel measurements are reported as within-row ratios instead.

D.4 DrivAerML depth, latent budget, and training cost

Figure 8 plots the runs of Figure 5 against depth, at every latent budget. FLARE and FLARE++ each form a tight bundle across latent budgets, and FLARE++ lies below FLARE at every depth. Both

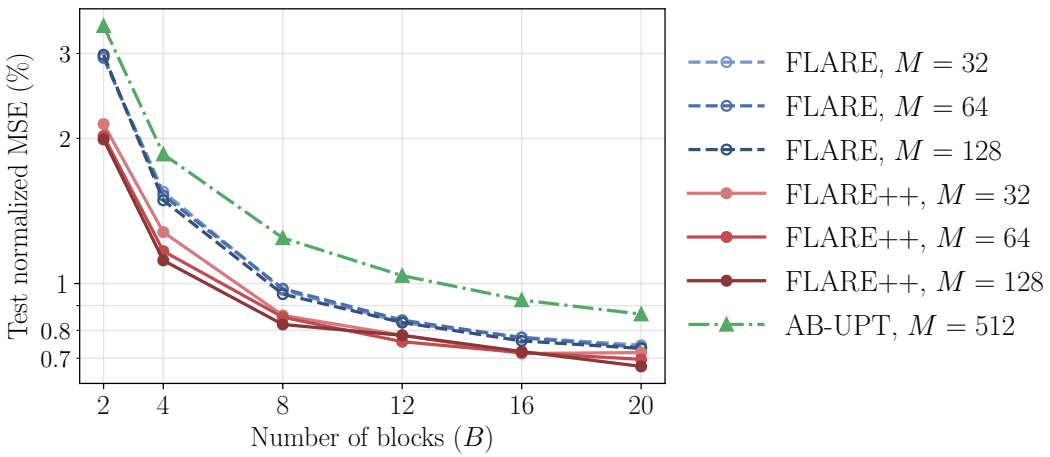


Figure 8: Test normalized MSE (in %) against depth B on DrivAerML's full 8.2M-cell surface mesh, in FP16 under context parallelism, for FLARE, FLARE++, and AB-UPT. The same runs are plotted against training cost in Figure 5.

models continue improving with depth; with the best latent budget at each depth, FLARE's error falls by 49% from two to four blocks, by a further 36% from four to eight, and by 23% more from eight to twenty. Synthesized routing queries are therefore especially useful in shallow networks, while remaining beneficial as the backbone grows.

The GPU hours of Figure 5 are estimated from separate single-GPU FP16 timing runs at the training configuration, as 500 times the wall time of one steady-state epoch including data loading, and exclude compilation and full-mesh evaluation. The ratio of 1.1–1.2× between matched FLARE++ and FLARE runs is therefore an end-to-end figure, unlike the isolated step times of Section D.3. Below 2.5 GPU hours, each FLARE run is the most accurate run at its cost, because the matching FLARE++ run costs 11–16% more.

D.5 Long Range Arena

FLARE++ uses the Transformer-block backbone, input and output projections, and feed-forward network of the FLARE LRA experiments. The table combines baseline reruns from that study with published results; its caption identifies their provenance. The comparison includes full self-attention (Vaswani et al., 2017); local attention and the original LRA baselines (Tay et al., 2021b); Reformer (Kitaev et al., 2020); Sparse Transformer (Child et al., 2019); Sinkhorn Transformer (Tay et al., 2020); Linformer (Wang et al., 2020); Performer and FAVOR++ (Choromanski et al., 2020); Funnel-Transformer (Dai et al., 2020); Synthesizer (Tay et al., 2021a); linear attention (Katharopoulos et al., 2020); Longformer (Beltagy et al., 2020); BigBird (Zaheer et al., 2020); Norm attention (Qin et al., 2022a); cosFormer (Qin et al., 2022b); Nyströmformer (Xiong et al., 2021); Skyformer (Chen et al., 2021); Hedgehog (Zhang et al., 2024); Luna (Ma et al., 2021); and Transolver (Wu et al., 2024). FLARE uses the lightweight configuration of the FLARE study, with linear key/value projections, a GELU feed-forward network, and query/key normalization; FLARE++ uses the matched configuration with the additional query-synthesis branch. The comparison covers attention-based encoders; it does not evaluate state-space sequence models. As discussed by Miralles-González et al. (2025), LRA tasks also contain locality and positional biases, so we use this benchmark to assess transfer beyond PDEs.

Table 12: Accuracy (%) on Long Range Arena (LRA) tasks (Tay et al., 2021b). Rows marked with † use the reported result from Zhang et al. (2024); the row marked with ‡ uses the Luna result (projected length 256) reported by Ma et al. (2021) under its own training setup; the other baseline results and the FLARE result reproduce the reruns reported alongside the FLARE arXiv paper (Puri et al., 2026). The best result in each column is **bold** and the second best is underlined.

Model	**ListOps**	**Text**	**Retrieval**	**Image**	**Pathfinder-32**	**Avg**
Full self-attention	36.70	64.93	77.18	38.22	70.52	57.51
Local attention†	15.82	52.98	53.39	41.46	66.63	46.06
Reformer†	37.27	56.10	53.40	38.07	68.50	50.67
Sparse Transformer†	17.07	63.58	59.59	44.24	71.71	51.24
Sinkhorn Transformer†	33.67	61.20	53.83	41.23	67.45	51.29
Linformer (KV sharing)	36.95	51.74	77.86	42.82	50.02	51.88
Performer	35.90	64.21	68.42	37.60	53.83	51.99
Funnel-Transformer	<u>38.50</u>	61.17	61.55	**53.10**	49.98	52.86
Synthesizer†	36.99	61.68	54.67	41.61	69.45	52.88
Linear attention	17.95	<u>66.00</u>	71.84	34.66	75.00	53.09
Longformer†	35.63	62.85	56.89	42.22	69.71	53.46
Linformer (headwise sharing)	36.70	53.00	64.72	43.42	70.09	53.59
BigBird†	36.05	64.02	59.29	40.83	74.87	55.01
Norm attention	18.30	63.08	76.07	<u>48.22</u>	70.15	55.16
Performer (FAVOR++)	36.00	64.26	76.74	35.60	69.67	56.45
cosFormer	36.20	64.59	76.78	41.52	75.38	58.89
Nyströmformer†	37.15	65.52	79.56	41.58	70.94	58.95
Skyformer†	**39.25**	64.70	<u>82.06</u>	40.77	70.73	59.50
Hedgehog†	37.15	64.60	**82.24**	40.15	74.16	59.66
Luna‡	37.98	65.78	79.56	47.86	<u>78.55</u>	**61.95**
Transolver	17.65	60.95	76.56	34.50	66.94	51.32
FLARE (Puri et al., 2026)	36.85	65.23	78.07	36.86	73.39	58.08
FLARE++ (ours)	38.05	**66.56**	79.37	45.14	**78.86**	<u>61.60</u>

E QUALITATIVE FIELD PREDICTIONS

Aggregate test errors do not reveal *where* a surrogate fails. Figures 9–14 therefore show, for each benchmark, the reference solution and both predictions above the two pointwise error maps, on a test case selected by FLARE++'s median error, with both models on the same case. The DrivAerML example is selected from ten scored test geometries, as specified in Figure 13; it is not the median over the full test split. The three field panels share one colour scale and the two error panels share another, so a smaller error appears as an emptier panel rather than a rescaled one. The examples complement the aggregate errors with a view of their spatial distribution. The error is unevenly distributed, concentrating near the void boundary on Elasticity, the permeability interfaces on Darcy, the leading edge and wake on Airfoil, the near-wall region on Pipe, and the edges and creases of the DrivAerML surface; the FLARE and FLARE++ maps on a given case look broadly similar and differ mainly in amplitude.

Rendering conventions. Every colour bar is labelled with the quantity it shows and its units. Field panels show the predicted quantity as the dataset releases it: the Elasticity stress carries no units in the Geo-FNO release, the Darcy pressure and Pipe velocity are dimensionless, Airfoil shows the Mach number, and DrivAerML stores OpenFOAM's kinematic surface pressure $p_s = p/\rho$ and kinematic wall-shear magnitude $\tau = |\boldsymbol{\tau}_w|/\rho$, both in $\mathrm{m}^2/\mathrm{s}^2$. Error panels show, at every point, the prediction minus the reference divided by the root-mean-square (RMS) of the reference field over the same case, in percent. Two-dimensional panels show this signed value on a diverging scale centred at zero; surface panels show its magnitude on a sequential scale, with the RMS taken over the cells visible from the camera. This is the pointwise analogue of the relative L^2 error in the tables: the RMS of an error panel over its case is that case's relative L^2 error in percent. The two error panels of a figure share one colour scale whose limit is the 99.5th percentile of the larger model's error magnitude. Surfaces are rendered from cell centres and normals with VTK under fixed lighting and a parallel projection, with field limits at the 99.5th percentile over visible points. Grey marks regions with no data, including the Elasticity void.

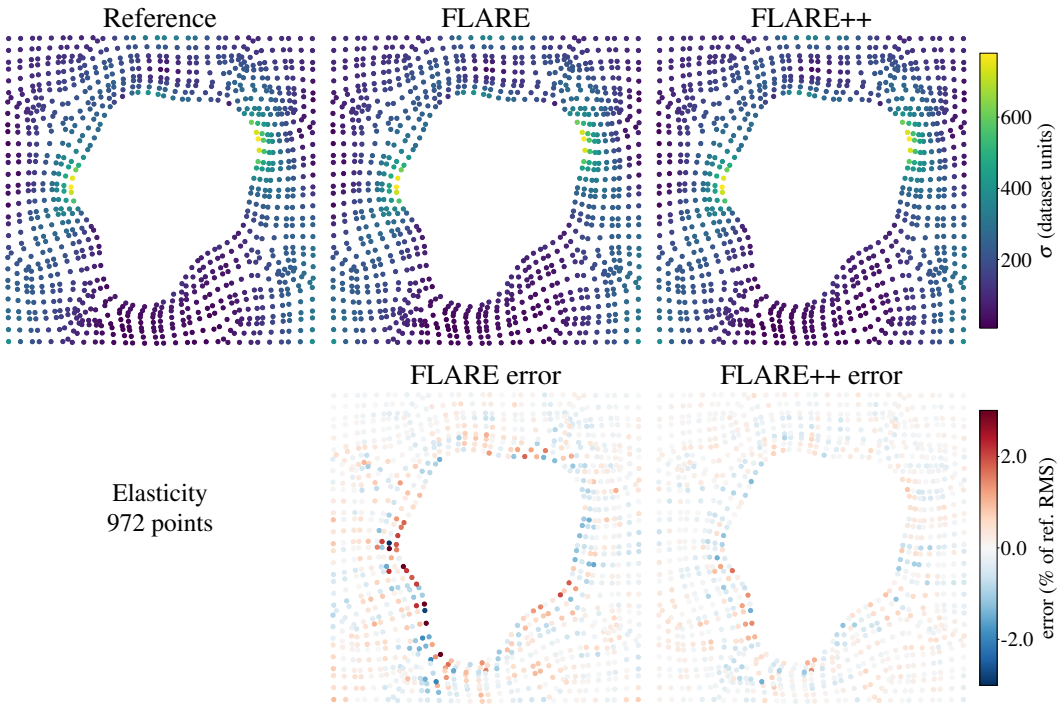


Figure 9: Elasticity: the von Mises stress σ on the unstructured unit cell, in the units of the Geo-FNO release, which states none. Top row, the reference solution and the two predictions; bottom row, each model's signed pointwise error in percent of the reference field's RMS.

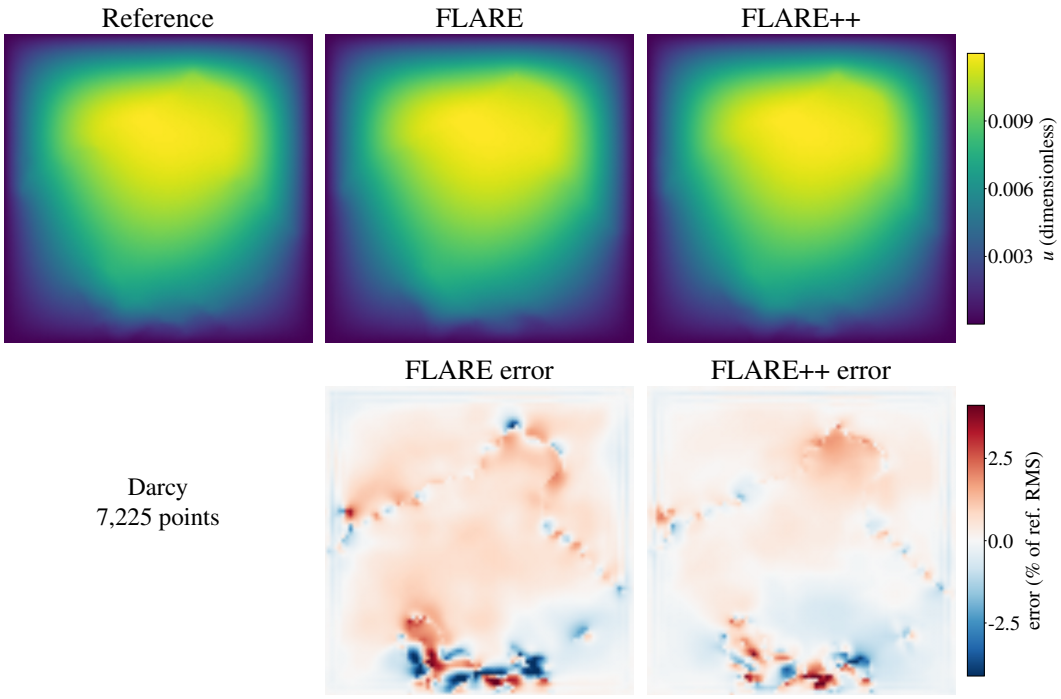


Figure 10: Darcy: the dimensionless pressure u. Top row, the reference solution and the two predictions; bottom row, each model's signed pointwise error in percent of the reference field's RMS.

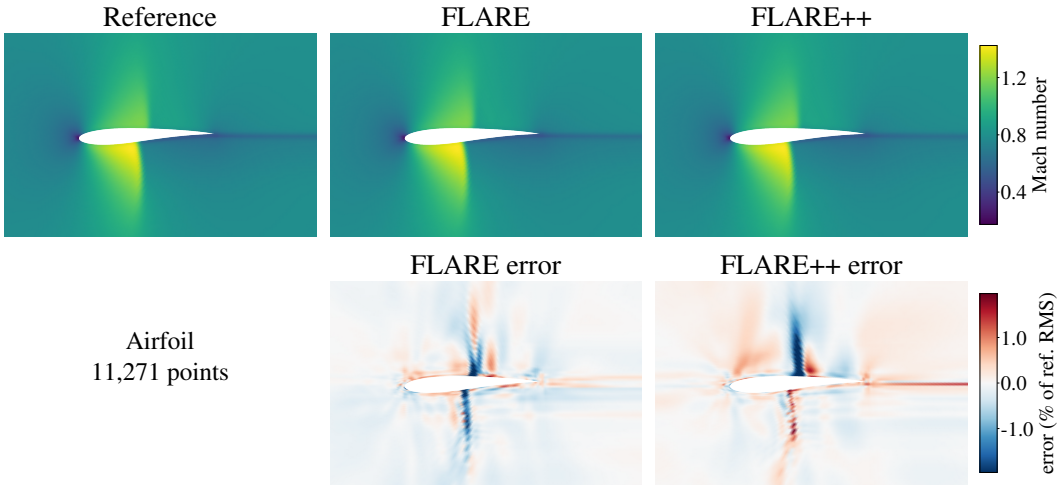


Figure 11: Airfoil: the Mach number (dimensionless) on the body-fitted mesh, cropped to the airfoil. Top row, the reference solution and the two predictions; bottom row, each model's signed pointwise error in percent of the reference field's RMS over the full mesh.

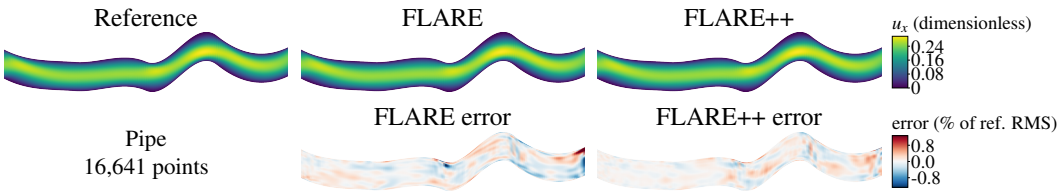


Figure 12: Pipe: the dimensionless streamwise velocity u_x. Top row, the reference solution and the two predictions; bottom row, each model's signed pointwise error in percent of the reference field's RMS.

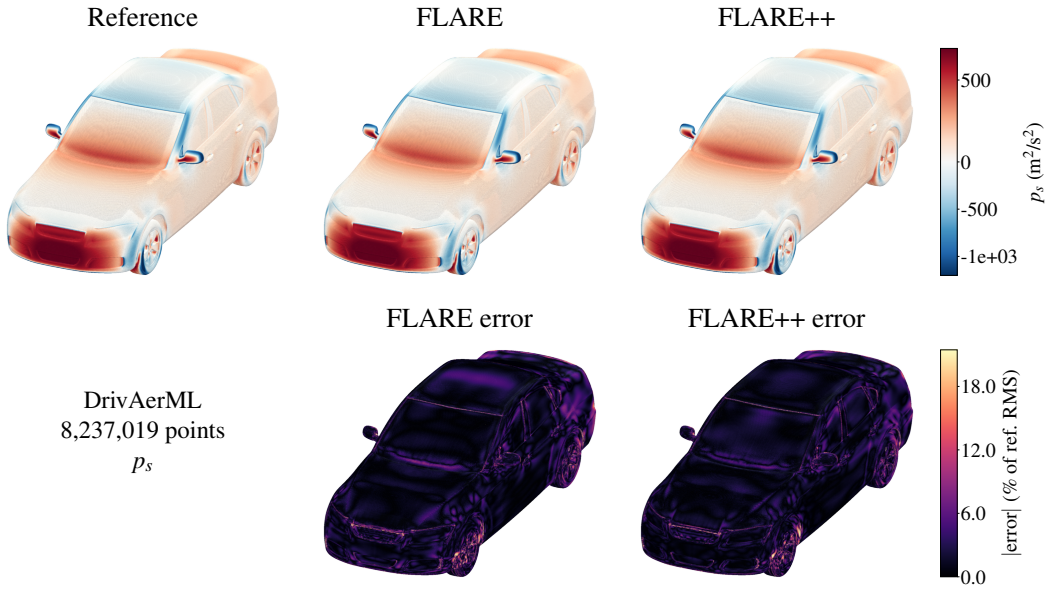


Figure 13: DrivAerML on the full surface mesh: the kinematic surface pressure $p_s = p/\rho$ in m^2/s^2. Top row, the reference solution and the two predictions; bottom row, the magnitude of each model's pointwise error in percent of the reference field's RMS over the visible cells. This is the full-surface task of Section 4.2, not DrivAerML-40K. The case shown is the median over the first 10 of the 50 test geometries, the subset scored for this figure.

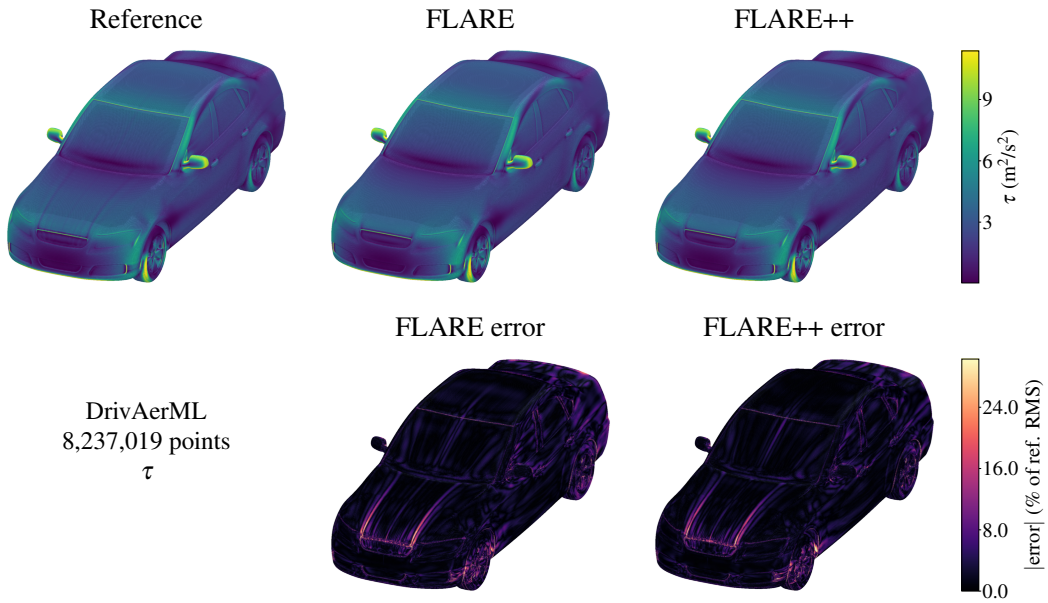


Figure 14: DrivAerML on the full surface mesh: the kinematic wall-shear magnitude $\tau = |\boldsymbol{\tau}_w|/\rho$ in m^2/s^2. The case-selection subset is the same as in Figure 13. Top row, the reference solution and the two predictions; bottom row, the magnitude of each model's pointwise error in percent of the reference field's RMS over the visible cells.