DOES TRANSOLVER REALLY NEED A TRANSFORMER?

Anonymous authors
Paper under double-blind review

ABSTRACT

The widely used Transolver family of neural operators is based on physics-attention, which softly assigns the points of an unstructured mesh to a small number of slices, applies self-attention among the resulting tokens, and broadcasts the result back to the points. We provide a comprehensive empirical and theoretical analysis to elucidate the mechanisms which are responsible for model performance. To this end, we perform careful ablations on a challenging suite of nine 3D fluid dynamics benchmarks to find that replacing token attention with a constant linear map does not affect the accuracy. Thus, Transolver does not need a Transformer at all. However, removing the global mixing (slicing/deslicing) or doing it only once leads to performance collapse. We leverage the theory of averaging neural operators to explain and corroborate our findings by showing that just slicing/deslicing, in conjunction with pointwise MLPs, already suffices for universal approximation of continuous operators and attention is redundant in this context. Finally, we provide a novel FlashAttention-style efficient implementation of the key slicing/deslicing module of Transolver. This FlashSlice kernel streams slice and deslice over the points without materializing the heavy slice-weight tensor, while reproducing the best available implementation to floating point error. At the same time, it leads to very significant memory and compute savings, particularly at large slice counts.

1 INTRODUCTION

Partial differential equations (PDEs) describe diverse phenomena across the physical sciences and engineering (Evans, 2010), and simulating them accurately is a central task of computational science. Classical numerical methods (Quarteroni & Valli, 1994) do so reliably, but a high-fidelity simulation of a flow around an industrial geometry can occupy an HPC cluster for hours to days. This cost becomes prohibitive in the *many-query* settings of design optimization, uncertainty quantification and inverse problems, where the same equation is solved for thousands of geometries and operating conditions. Scientific machine learning (Mishra & Townsend, 2024), in the specific form of *operator learning*, addresses this bottleneck by learning the *solution operator* of a PDE from a dataset of simulations, so that a new instance costs a very cheap single forward model pass.

The first generation of operator learning algorithms was designed for structured grids: the Fourier neural operator (Li et al., 2021), DeepONet (Lu et al., 2021), convolutional neural operators (Raonić et al., 2023) and operator vision transformers (Herde et al., 2024) are based on Cartesian grids. Industrial engineering simulations, however, are based on unstructured meshes and point clouds that can resolve localized features such as boundary layers, shocks and vortices at widely varying point densities. A growing literature targets this setting with graph neural networks (Pfaff et al., 2021; Mousavi et al., 2026), geometry-informed hybrids (Li et al., 2023b; Wen et al., 2025) and transformers whose tokens are mesh point features (Cao, 2021; Hao et al., 2023; Li et al., 2023a). Among the latter, Transolver (Wu et al., 2024) has become a widely used choice. Its *physics-attention* layer softly assigns each of the N mesh points to a small number G of *slices*, computes self-attention among the G resulting tokens, and broadcasts the result back to the original points, at a cost linear in N. It has been extended by Transolver++ (Luo et al., 2025), Transolver-3 (Zhou et al., 2026) and GeoTransolver (Adams et al., 2025), among others, and adopted in industrial frameworks such as NVIDIA PhysicsNeMo.

Despite its popularity, we argue that the Transolver model itself has not been properly analyzed and understood. In particular, which of its three ingredient operations: slicing/deslicing, attention

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.

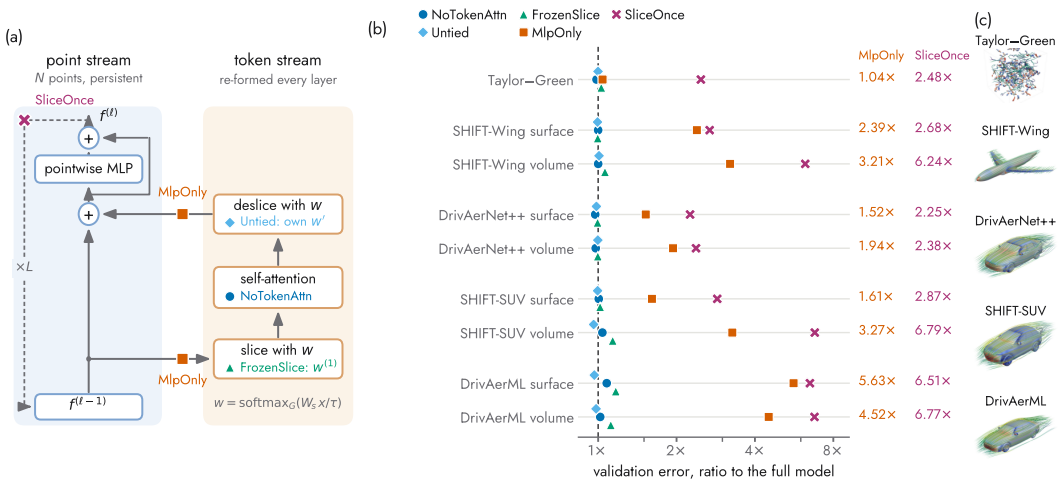

Figure 1: **What Transolver's accuracy does, and does not, need.** (a) A Transolver layer: a persistent full-resolution point stream, coupled through slice and deslice with shared per-point slice weights w to $H \times G$ tokens that are re-formed at every layer. Each ablation is drawn on the element it changes; $w^{(1)}$ denotes the layer-1 weights reused by FROZENSLICE and w' the separate deslice weights of UNTIED. (b) Validation-error ratio to the full model for the five ablations on nine benchmarks, markers as in (a). The variants that keep the coupling (NOTOKENATTN, UNTIED, FROZENSLICE) sit at $\approx 1\times$; removing it (MLPONLY) costs 1.04–$5.6\times$ and collapsing the point stream (SLICEONCE) up to $6.8\times$. (c) One reference solution per benchmark family, each beside the rows of (b) it belongs to: Taylor–Green vorticity, and surface pressure with streamlines on SHIFT-Wing, DrivAerNet++, SHIFT-SUV and DrivAerML.

among the tokens and persistent pointwise MLPs, is responsible for the accuracy shown by the model is completely unclear. Although the focus of successive works on Transolver has been to modify the attention mechanism, it is not even clear whether *Transolver really needs a Transformer inside it or not?* or whether memory-heavy operations such as slicing/deslicing contribute significantly to its accuracy. Our main motivation for this paper is to perform a comprehensive empirical and rigorous mathematical analysis of the Transolver model to precisely answer such questions. In this regard, our contributions in this paper are,

- **A careful empirical analysis of physics-attention across nine benchmarks** (§2–3). We perform controlled ablations of every design decision within the Transolver model, trained under an identical protocol on nine challenging 3D benchmarks (see Fig. 1) to demonstrate the *surprising fact that replacing token self-attention by a constant linear map matches the full model on all nine* (-2.6% *to* $+7.9\%$), at every slice count from 8 to 256. Moreover, we identify that the slice-deslice operations are essential and removing them leads to significant loss of accuracy, increasing the error by 1.04–$5.6\times$. Finally, we also show that slicing and deslicing have to be repeated every layer. Not doing so and simply running a deep transformer on the tokens leads to very large drops in accuracy, with errors 2.2–$6.8\times$ those of the baseline model.
- **A mathematical explanation from the theory of neural operators** (§4). By interpreting slicing/deslicing as finite-rank nonlocal integral operators with a learned partition of unity and leveraging the universal approximation theorem of Lanthaler et al. (2023), we show that the transformer-free Transolver is already universal, that a purely pointwise stack is bounded away from every nonlocal target at any width and depth, and that slicing/deslicing once is not as efficient as doing it each layer.
- **FlashSlice** (§5). We implement the key slicing/deslicing module as fused Triton kernels that stream over the points and never write the slice-weight tensor, with a single-tile path for small slice counts and a G-blocked path with saved softmax statistics for arbitrary ones. They are exact and bit-deterministic, and train 1.35–$1.7\times$ faster in up to 25% less memory

at the standard configuration over the current implementation and 4–6.5× faster in 85–96% less memory at large slice counts.

Thus, we provide a comprehensive empirical analysis that identifies the key causal mechanism underlying Transolver's accuracy, explain it with mathematical theory and provide a highly efficient implementation of this core layer.

2 PHYSICS-ATTENTION AS A LOW-RANK NONLOCAL OPERATOR

Problem formulation. We consider PDEs $\mathcal{L}(u; a) = 0$ in $\Omega \subset \mathbb{R}^d$ with boundary conditions $\mathcal{B}(u; a) = 0$ on $\partial\Omega$, where $a \in X$ collects the geometry, coefficients, initial data and operating conditions of an instance and $u \in Y$ is its solution and X, Y are suitable function spaces. The solution operator $\mathcal{S} : X \to Y$, $a \mapsto u$, is what a neural operator Ψ_θ approximates, given training pairs $(a_k, \mathcal{S}(a_k))$ with a_k drawn from a distribution μ on X. In practice a and u are given on a mesh $\{x_i\}_{i=1}^N \subset \Omega$, so Ψ_θ maps per-point inputs to per-point outputs.

The Transolver layer. Transolver (Wu et al., 2024) lifts the per-point inputs to features $f_i^{(\ell)} \in \mathbb{R}^c$ and applies L layers, with $1 \le \ell \le L$, of two interleaved streams. The *token stream* is physics-attention: per head, a linear projection of the features with a learned temperature τ produces *slice weights* $w_{i,g} = \mathrm{softmax}_g\big((W_s f_i + b)/\tau\big)$, a soft assignment of each point to G slices (with layer index ℓ suppressed for notational convenience); the *slice* operator outputs tokens as weighted averages $z_g = \sum_i w_{i,g} v_i / \sum_i w_{i,g}$ of projected features v_i; self-attention is applied among the G tokens; and *deslice* broadcasts the result back, $u_i = \sum_g w_{i,g} z'_g$, added residually to the features. The *point stream* is a pointwise two-layer MLP with residual connection at full resolution after every attention sublayer, forming a persistent point-level representation. Two properties are worth mentioning: the slice weights are recomputed from the current features at every layer, and the same weights are used in the slice and deslice operators, i.e. they are *tied*.

A rank-G integral operator. In the continuum limit, the slice weights define G functions $s_g : \Omega \to [0, 1]$, $w_{i,g} = s_g(x_i)$, with $\sum_g s_g \equiv 1$, forming a learnable, data-dependent partition of unity. Informally, one can think of slice as analysis against this set of functions and deslice as synthesis in its span, so that one physics-attention sublayer implements the rank-G integral operator

$$(\mathcal{K}v)(x) \;=\; \sum_{g,h=1}^{G} s_g(x)\, A_{gh}\, \frac{\int_\Omega s_h(y)\, v(y)\, \mathrm{d}\mu(y)}{\int_\Omega s_h(y)\, \mathrm{d}\mu(y)}. \tag{1}$$

Here the core $A \in \mathbb{R}^{G \times G}$ is the nonlinear map produced by the token self-attention in BASELINE version of Transolver, and is therefore content-dependent, whereas in the attention-free variant NOTOKENATTN below the core is constant: A is the identity, so the tokens do not interact, and only a learned linear map acts on the channels of each token; a full layer reads $\bar{f}^{(\ell)} = f^{(\ell-1)} + \mathcal{K}_\ell f^{(\ell-1)}$, $f^{(\ell)} = \bar{f}^{(\ell)} + \mathrm{MLP}\big(\bar{f}^{(\ell)}\big)$. Equation (1) is clearly a low-rank instance of the nonlocal neural operator layer of Kovachki et al. (2023), distinguished by the partition of unity $\{s_g^\ell\}$, re-estimated from $f^{(\ell-1)}$ at every layer and by the interleaving with a persistent pointwise stream. Its range is $\mathrm{span}\{s_g^\ell\}$ regardless of A: thus, the attention re-parameterizes the core but cannot enlarge what the slice/deslice pair, the *coupling*, expresses.

Ablation suite. We isolate each design decision in Transolver with variants of BASELINE. Each ablation changes exactly one ingredient in the full Transolver model, except UNTIED-LN, which is deliberately the conjunction of two ingredients plus a normalization and reproduces the LinearNO layer, recently considered in Hu et al. (2026) (Appendix E).

NOTOKENATTN	token self-attention → per-token linear map (no $QK^\top$ softmax; tokens do not interact)
MLPONLY	attention sublayer removed entirely (pointwise residual MLP stack; no cross-point path)
MLPONLY-WIDE	MLPONLY widened to 112% of BASELINE parameters (capacity control)
UNTIED	deslice gets its own projection and temperature (tests weight tying)
UNTIED-LN	UNTIED + NOTOKENATTN + analysis-side normalization along N (LinearNO)
FROZENSLICE	slice weights from layer 1, reused by all layers (tests per-layer re-slicing)
SLICEONCE	slice once → transformer on G tokens → deslice once (Perceiver; no point stream)

3 EXPERIMENTS

Benchmarks. We use nine dataset configurations from CFD, the three-dimensional Taylor–Green vortex which is the prototype of turbulent time-varying flows (Molinaro et al., 2024) and aerodynamics at industrial scale, namely DrivAerNet++ (Elrefaie et al., 2024) surface and volume (fixed inflow), SHIFT-SUV surface and volume (fixed inflow, geometry the only input), the transonic SHIFT-Wing surface and volume (varying inflow and geometry), and DrivAerML (Ashton et al., 2024) surface and volume (fixed inflow; up to 1.3×10^8 points per sample). The suite spans a very diverse set of operators, ranging from inputs that contain the local field state to solutions set by a global interaction of geometry and operating conditions (Appendix B).

Protocol. For each benchmark, all variants of Transolver are trained to a common step budget on identical data pipelines with the Transolver backbone of Wu et al. (2024) (L=8, width 256, 8 heads, G=32; 3.87M parameters); the headline pair BASELINE/NOTOKENATTN uses three seeds. We report the best validation relative L^1 error for each variant. Step budgets are checked rather than assumed and every ablation is verified from the trained artifacts through checkpoint contents and bit-level freeze tests, see Appendix C for details.

The token self-attention is removable on all nine benchmarks, i.e., Transolver does not really need a Transformer! We observe from Table 1 that NOTOKENATTN, which replaces the content-dependent attention among the G tokens by a learned linear map applied to each token separately, so that the tokens no longer interact, matches BASELINE everywhere: it lies within -2.6% to $+1.8\%$ on seven benchmarks and at $+3.8\%$ and $+7.9\%$ on the remaining two (SHIFT-SUV volume and DrivAerML surface), against the 1.04–$6.8\times$ spread of the coupling ablations in the same table, and on four benchmarks it is the best variant overall (tied with BASELINE on SHIFT-Wing volume).

Untying the projections is not what makes it removable. A possible explanation of the above result is that tying the slice and deslice projections confines feature interaction to the same token, so that the attention has to supply the missing cross-slice routing. On this reading, untying these projections is what renders attention redundant (Hu et al., 2026). This explanation and ours (§4) disagree on the tied and attention-free layer, which Hu et al. (2026) predict to perform poorly, whereas NOTOKENATTN already demonstrates that this is not the case. Moreover, in Fig. 2(a), we present results for the 2×2 grid of token attention $\times$ projection tying on SHIFT-SUV surface, the untied and attention-free cell corresponds exactly to the LinearNO layer of Hu et al. (2026) ported into our scaffold at matched latent width and capacity (Appendix E). The grid clearly shows that there is no difference between these configurations: all four cells lie within 1.05% of one another, and both main effects ($+0.54\%$ for removing the attention, -0.51% for untying) and their interaction (-0.41%) are about one standard deviation of the baseline's own three-seed spread of $\pm 0.47\%$. The attention is redundant, and the tying is clearly orthogonal to that redundancy. LinearNO also normalizes its slice weights by a softmax over all points instead of per point, and at its published latent width a training iteration is $6.8\times$ slower with no change in accuracy (Appendix E).

The result does not depend on the slice count. The core A is $G \times G$, so a larger number of slices G provides attention more information to mix. In Fig. 2(b), we present results with G swept over a $32\times$ range at a matched budget. NOTOKENATTN stays within 1.5% of BASELINE at every G and within 1% for any $G \geq 32$, the gap tightening as G grows, including at G=256, where the tokens per head outnumber the head dimension eightfold. Moreover, model capacity is always matched along the sweep (G: $32 \rightarrow 256$ adds 1.6% to the parameter count). Notably, the baseline itself moves by only 0.6% over the sweep on this benchmark.

The slice/deslice coupling and the point stream are not removable. Removing the entire attention sublayer (MLPONLY) costs $1.04\times$ to $5.6\times$ (Table 1). This is not explainable in terms of model capacity: widening the pointwise MLP to 112% of the baseline's parameters (MLPONLY-WIDE, Fig. 2(c)) recovers at most 3% of the gap and leaves the model $1.6\times$ (surface) and $1.9\times$ (volume) less accurate on DrivAerNet++. Nor is this difference attributable to budget differences; on SHIFT-SUV the MLPONLY ratio grows from $1.52\times$ to $1.62\times$ (surface) and from $2.43\times$ to $3.24\times$ (volume) as the budget rises from 10k to 50k steps (Appendix C). Conversely, SLICEONCE keeps the attention and has *more* parameters than the baseline, but slices once and confines all further depth to

Table 1: **Ablating physics-attention across nine benchmarks.** Best validation relative L^1 (lower is better), averaged over the predicted quantities in normalized units; per-quantity errors in physical units are in Tables D.2 and D.3. Mean±std over three seeds for BASELINE and NOTOKENATTN, single seed otherwise; best per row in **bold**. Three significant figures; standard deviations to one.

Dataset	BASELINE	NOTOKENATTN	MLPONLY	UNTIED	FROZENSLICE	SLICEONCE
Taylor–Green	$0.0756_{\pm.0001}$	$\mathbf{0.0745}_{\pm.0001}$	0.0786	0.0755	0.0777	0.188
SHIFT-Wing						
surface	$0.0580_{\pm.0005}$	$0.0579_{\pm.0005}$	0.139	**0.0576**	0.0577	0.155
volume	$\mathbf{0.127}_{\pm.0007}$	$\mathbf{0.127}_{\pm.0008}$	0.408	0.128	0.135	0.793
DrivAerNet++						
surface	$0.193_{\pm.005}$	$\mathbf{0.188}_{\pm.002}$	0.294	0.190	0.192	0.435
volume	$0.163_{\pm.0003}$	$\mathbf{0.159}_{\pm.001}$	0.315	0.162	0.162	0.386
SHIFT-SUV						
surface	$0.157_{\pm.0007}$	$0.157_{\pm.00003}$	0.252	**0.156**	0.159	0.449
volume	$0.0601_{\pm.0006}$	$0.0624_{\pm.0009}$	0.197	**0.0577**	0.0683	0.408
DrivAerML						
surface	$0.089_{\pm.001}$	$0.096_{\pm.000}$	0.501	**0.086**	0.104	0.579
volume	$0.110_{\pm.005}$	$0.112_{\pm.004}$	0.497	**0.108**	0.123	0.745

the G tokens; it is the worst variant on every benchmark, at 2.2–6.8×. On the other hand, re-using the layer-1 slice weights in every layer (FROZENSLICE) costs only -0.5% to $+17\%$, mostly on the volume tasks. Thus, cross-point coupling through slice/deslice, re-estimated at every layer and interleaved at full resolution with the pointwise stream, is clearly the most performant mechanism.

The coupling is needed where the physics is nonlocal. The MLPONLY penalty tracks the physics of each task. It is negligible (1.04×) on Taylor–Green, whose inputs contain the current field state at every point so that short-horizon prediction is largely local. On the aerodynamic benchmarks the field at a point is set by a global interaction between the whole body and the flow around it, which a function of local descriptors can absorb only in part (Appendix D.1), and the size of the penalty does not follow the inflow: it is moderate (1.5–3.3×) for DrivAerNet++ and SHIFT-SUV and largest for DrivAerML (up to 5.6×), all three at fixed inflow, while SHIFT-Wing, the only benchmark whose inflow varies, lies between these (2.4–3.2×). SHIFT-SUV isolates a second axis, the kind of target: same cars, split, budget and fixed condition, differing only in whether the target is a trace on the body or a field transported away from it, and every coupling-sensitive number roughly doubles from surface to volume (MLPONLY 1.61 → 3.27×, SLICEONCE 2.87 → 6.79×); SHIFT-Wing shows the same changes under varying inflow (MLPONLY 2.39 → 3.21×, SLICEONCE 2.68 → 6.24×). The per-field decomposition makes the same point inside each task (Figure 2(d)): on every aerodynamic benchmark the pressure, which can be recovered as the solution of a global elliptic problem, degrades more than the velocity or wall shear beside it when the coupling is removed, by 1.8–14.8× against 1.3–5.0×. Figure D.1 (Appendix D) shows the corresponding fields, and Appendix D.2 the same figure on every benchmark.

4 WHY THE ATTENTION IS REMOVABLE AND THE COUPLING IS NOT

Our aim in this section is to explain the results of Sec. 3 using mathematical theory. Our starting point is the following proposition on the universality of the Transolver layer, Eq. (1),

Proposition 1 (The constant-core class is universal)**.** *Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipschitz domain and $\Psi^\dagger : \mathcal{X} \to \mathcal{Y}$ a continuous operator between C^s or $W^{s,p}$ spaces of functions on Ω. Fix $G \geq 1$ and let Ψ_A be the class of operators realized by a stack of layers of the form (1) with a constant core A (*NOTOKENATTN*), interleaved with pointwise MLPs of unbounded width. Then, for every compact $\mathsf{K} \subset \mathcal{X}$ and $\epsilon > 0$ there exists an operator $\Psi \in \Psi_A$ satisfying $\sup_{u \in \mathsf{K}} \|\Psi(u) - \Psi^\dagger(u)\|_{\mathcal{Y}} \leq \epsilon$.*

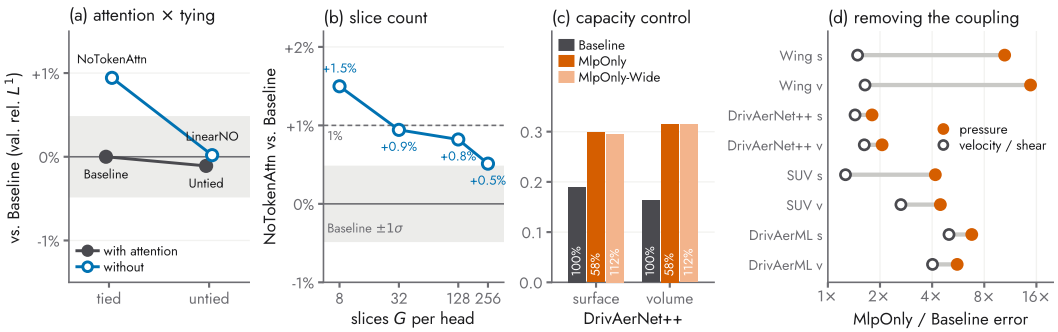

Figure 2: **Four controls.** (a) Token attention × projection tying on SHIFT-SUV surface, change against the tied BASELINE; UNTIED-LN is the released LinearNO layer (Hu et al., 2026) in our scaffold. (b) NOTOKENATTN against BASELINE across the slice count G, same benchmark. Grey band in (a, b): the baseline's three-seed spread. (c) Capacity control on DrivAerNet++; bars labelled by parameter count relative to BASELINE. (d) Test error of MLPONLY over BASELINE per field, pressure against velocity or wall shear (Table D.2).

The proof of this proposition is detailed in Appendix A and is based on the universality of *averaging neural operators* (ANOs) of Lanthaler et al. (2023), where the authors show that neural operators of the form,

$$\Psi(u)(x) \;=\; Q\Big(\sigma\big(\textstyle\int_\Omega R(u(y),y)\,\mathrm{d}y\big),\, x\Big), \tag{2}$$

with pointwise MLPs R, Q already suffice for universal approximation of continuous operators. Setting all slice logits to zero makes $s_g \equiv 1/G$ a partition of unity, every token becomes the global mean, and Eq. (1) provides the ANO coupling equation 2, while the pointwise MLPs supply R and Q, and the construction uses no explicit property of G.

The proposition leads to the following immediate conclusions i) to the best of the authors' knowledge, it provides the first universality result for the original Transolver architecture as it is straightforward to realize token attention weights to obtain a constant A in equation 1, making Transolver universal and ii) the result clearly suggests that using a transformer is not necessary for universal approximation and makes this observation of Sec. 3 unsurprising.

We can also explain why pointwise MLPs do not suffice for accuracy as reported in Sec. 3. To this end, we define

Definition 1 (Locality gap)**.** *Call Φ pointwise if $\Phi(u)(x)$ depends on u only through $u(x)$. For a continuous operator $\Psi^\dagger : \mathcal{X} \to \mathcal{Y}$ and compact $\mathsf{K} \subset \mathcal{X}$ let $\delta_{\mathrm{loc}}(\Psi^\dagger) := \inf_{\Phi \text{ pointwise}} \sup_{u\in\mathsf{K}} \|\Psi^\dagger(u) - \Phi(u)\|_{\mathcal{Y}}$.*

Corollary 1 (Depth does not repair MLPONLY)**.** *A composition of pointwise maps is pointwise, and the pointwise class is closed under uniform limits on K. Hence for every $\Psi^\dagger$ with $\delta_{\mathrm{loc}}(\Psi^\dagger) > 0$, the error of* MLPONLY *is at least $\delta_{\mathrm{loc}}(\Psi^\dagger)$ at every depth and every width.*

Thus, MLPONLY cannot be universal as long as the operator is not entirely pointwise. Moreover, this is the sense in which the MLPONLY gap is not a capacity deficit that MLPONLY-WIDE could close. It also orders the tasks: δ_{loc} is largest for operators with global support, the elliptic pressure problem of low-Mach aerodynamics being the canonical case, and smallest for the short-horizon map of Taylor–Green.

Finally, the following proposition is suggestive of some of the other observations of Sec. 3,

Proposition 2 (Nonlocal budgets)**.** *Let c be the width of the point stream. (i) If $\Psi^\dagger$ is ϵ-approximable by an ANO equation 2 of encoding width $J \le c-k$, with k channels carrying the coordinates, then a constant-core stack of layers of the form (1) attains accuracy ϵ for* every *$G \ge 1$. (ii) One application of $\mathcal{K}$ in equation 1 transmits at most $G\,c$ real functionals of the field, and every point receives a fixed linear image of them. A stack that couples at each of L layers carries at most $L\,G\,c$;* SLICEONCE *carries at most $G\,c$ whatever the depth of its token stack;* MLPONLY *carries* 0.

Part (i) of the above proposition suggests why accuracy may be flat over a $32\times$ range of G (Figure 2(b)): universality is not bought with the slice count. Part (ii) provides the asymmetry behind SLICEONCE: all nonlocal information reaching its output passes through one round of $G\,c$ numbers, whereas variants that couple at every layer have L rounds, each re-estimated from transformed features, and only these improve with depth. The bound is rather loose ($G\,c \approx 8 \times 10^3$ here); the size of the penalty comes from a second missing ingredient, the universal pointwise decoder Q of equation 2, which SLICEONCE replaces by an affine readout in the tokens with coefficients $s_g(x)$ (Table A.1 in Appendix A summarizes these results).

5 FLASHSLICE: THE REDUCED LAYER AS A KERNEL

The bottleneck. After removing attention, a physics-attention sublayer is simply a slice $\to$ per-token linear map $\to$ deslice operation, and the coupling is what dominates compute and memory. The currently available eager implementation of Transolver forms the slice weights $w \in \mathbb{R}^{B\times H\times N\times G}$ explicitly and keeps them for the backward pass. For N in the millions, this is the dominant activation of the layer and the only per-layer term that also grows with G, so slice count and depth compete for the same memory budget (Figure 3(b, c)). Slice and deslice cost $O(NGD)$ multiply-adds for head width D, comparable to the pointwise MLPs at $G{=}32$ and most of the layer at $G{=}1024$. Hence, our aim in this section is to describe FlashSlice, a much more efficient implementation of the slicing kernel.

Design. FlashSlice never writes w. Each pass streams over the points in tiles, forms the membership tile in registers, applies it and moves on, and the backward pass recomputes the tile from what was saved: the trade FlashAttention (Dao et al., 2022) makes for softmax attention, applied to a pooling that is not attention. The query is a point, the keys are the G slot projections, the softmax runs over the slots, and the token state between the passes is $H \times G \times D$ floats (32 KB at our configuration). Two things differ from attention: slice is the *transpose* of deslice, accumulating per slot over points rather than per point over slots, so each point's normalizer must be known before its contribution is added; and the temperature τ is learned, its gradient cancelling row by row in exact arithmetic, which makes it the most rounding-sensitive quantity in the layer. Two kernel families sit behind one switch and are chosen by shape.

- *Single-tile kernels*, for G and D powers of two in $[16, 128]$, hold the whole slot axis in one tile, so that the softmax over G never leaves registers and no statistics are stored. Tiles are tuned per G (carrying the $G{=}32$ table to $G{=}128$ costs up to 40–80$\times$ through register spilling); they serve every shape of our accuracy experiments.
- *G-blocked kernels*, for any $G \geq 1$ and head or value widths up to 256, take the approach of FlashAttention's *backward*. One online pass forms the per-point softmax statistics (m_n, l_n), two floats per point and head, i.e. $2/G$ of w, and every kernel then recomputes $w_{ng} = \exp(\mathrm{logit}_{ng} - m_n)/l_n$ one G-block at a time. A deslice with no statistics in hand runs FlashAttention's forward instead, an online softmax over G, and hands its statistics to a tied slice; slice cannot run that loop, since its accumulators live on slots and mix many points, so its programs own a G-block, stream over N with the finished statistics, and write partials that the host reduces once. The price is the statistics pass and one more logits recompute per backward, so routing prefers the single-tile kernels wherever they apply.

Neither family uses atomics, so two runs are bitwise identical, and both are registered as PyTorch custom operators (Ansel et al., 2024), which makes the model one compiled graph instead of 34 and lets every ablation variant of this paper run through the same kernels. Shapes outside both families ($D > 256$) fall back to the eager path with a warning recorded on the model, so that a flag left on cannot silently train a baseline replica.

Relation to Transolver-3. Zhou et al. (2026) also avoid the full $N \times G$ weight tensor, by tiling the points in PyTorch and recomputing each tile under gradient checkpointing; each tile of w is still written to HBM, and the backward reruns the tile's whole forward. FlashSlice does the same inside the kernel, where the tile never leaves registers. Their second change, moving the input and output projections to token space by associativity, is orthogonal to ours and composes with it.

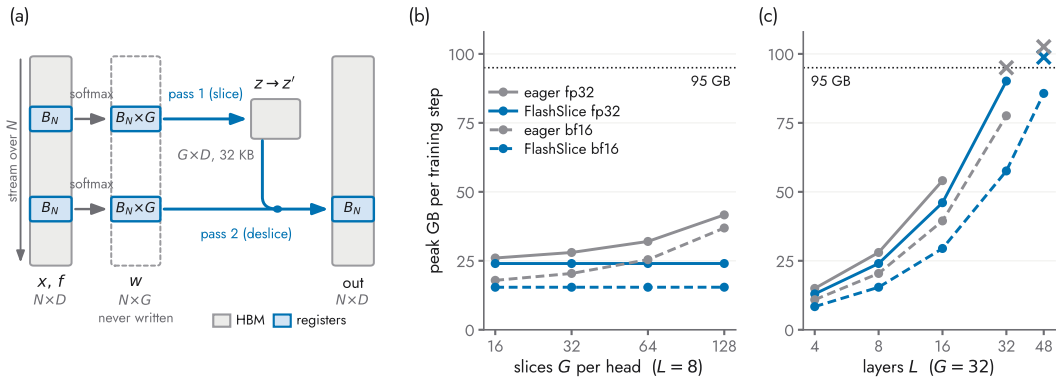

Figure 3: **FlashSlice turns the reduced layer into a systems object.** (a) The point tensors and the 32 KB token block live in HBM; the slice weights exist only as one $B_N \times G$ tile in registers, formed on the fly in each of two passes over the points and recomputed in the backward, so the $N \times H \times G$ tensor is never written. (b, c) Peak memory of a training step against slice count and depth (N=262k, eager PyTorch, no compilation or checkpointing); $\times$ marks the first configuration that does not fit in 95 GB. Step times of the same sweeps are in Figure F.1 (Appendix F).

Precision. We benchmark FlashSlice under both fp32 and bf16 autocast (Appendix F). Triton's default dot uses an fp32 FMA path, which becomes a compute bottleneck as G grows. We therefore provide several dot-precision levels, all with fp32 accumulation (Table F.1). The default uses fp32 dots; under bf16 autocast, tensor cores can be used for the value products and, at the `bf16` level, for the logits as well. At the other levels the logits remain in fp32 because the softmax Jacobian amplifies errors in the slice weights. In a 50k-step large-G training run, bf16 dots tracked a tf32 control throughout training while reducing step time by $3.5\times$ (Appendix F).

Exactness. FlashSlice is a drop-in, not an approximation: against an fp64 reference the fused error is at or below eager's on every output and every parameter gradient, for every ablation variant and both families, under bf16 autocast it matches the numerical class of the baseline's own tensor-core matmuls, and a 60-step optimizer trajectory from a shared initialization stays within 2×10^{-6} relative of eager. No accuracy result of this paper changes when the kernel is on (Appendix F).

FlashSlice is faster and smaller, and the advantage grows with the slice count. Table 2 measures both families against eager PyTorch on one GH200. At the configuration of our experiments (L=8, G=32), training steps are $1.35\times$ faster in fp32 and 1.5–$1.7\times$ faster under bf16 autocast at 14–25% less peak memory, inference is 1.5–$1.7\times$ faster at 20–26% less, and the largest sample one GPU processes grows from 8.4M to 10.5M points in fp32 and from 10.5M to 14.7M in bf16. The saving is structural: our memory is flat in G (15.42 GB at G=16, 15.43 GB at G=128) where eager's rises by 0.17 GB per slice (Figure 3b), and at a fixed 95 GB budget we train 48 layers where eager fits 32 (bf16) and 32 where it fits 16 (fp32). At large slice counts the G-blocked kernels turn this into a qualitative change: one layer holds 2 GB whether G is 48 or 1024, eager grows linearly and runs out of memory at G=512, N=1M, and with bf16 tensor-core dots the fused layer is 4.0–$6.5\times$ faster. Time at large G is a dot-throughput question: a point does eight times the multiply-adds at G=256 that it does at G=32, and Triton's exact fp32 dot is an FMA path at roughly 15 TFLOP/s, so in the default mode the blocked kernels are compute-bound and land near eager (0.85–$1.36\times$), whereas tensor-core dots pull them ahead by 3–$6\times$ in bf16 and up to $1.9\times$ for fp32 inputs via a three-term tf32 split. Against *compiled* eager the kernels keep 1.24–$1.26\times$, and they compose with activation checkpointing and point-sharding (Appendix F).

6 Discussion

Summary. Our aim was to present a thorough empirical and theoretical analysis of the widely used Transolver architecture in order to uncover the mechanisms that underwrite its accuracy in operator learning on arbitrary domains. By very careful and controlled ablations on a challenging suite of nine

Table 2: **FlashSlice against eager PyTorch on one GH200** (PyTorch 2.5, Triton 3.0; medians of ≥ 10 iterations measured in one job; no compilation or checkpointing; "OOM" = does not fit in 95 GB). Top: the model of our experiments (L=8, C=256, H=8, G=32), single-tile kernels, exact fp32 dots in fp32 and bf16 tensor-core dots in bf16. Bottom: one layer (H=8, D=32) at large slice counts, G-blocked kernels, bf16 inputs and dots.

			time (ms)			peak memory (GB)		
		N	eager	ours	speedup	eager	ours	saved
full model, G=32, single-tile kernels								
training	fp32	262k	171	126	1.35×	28.0	24.0	14%
training	bf16	262k	143	93.1	1.53×	20.4	15.4	24%
training	bf16	1M	585	345	1.70×	81.3	61.3	25%
inference	fp32	8.4M	1944	1160	1.68×	80.5	64.5	20%
inference	bf16	12.6M	OOM	1411	—	—	66.7	—
one layer, bf16, G-blocked kernels								
training	G=256	262k	47.5	11.9	3.98×	13.7	2.0	85%
training	G=256	1M	293.7	45.5	6.45×	54.5	8.1	85%
training	G=512	262k	90.6	20.0	4.53×	26.7	2.0	92%
training	G=512	1M	OOM	80.3	—	—	8.1	—
training	G=1024	262k	180.2	37.6	4.79×	52.7	2.1	96%
inference	G=256	4.2M	157.3	53.2	2.95×	68.0	8.3	88%

benchmarks in fluid dynamics and industrial aerodynamics, we demonstrated that i) token attention can be completely removed and replaced by a constant linear map implying that the Transolver does not require a Transformer ii) the slicing/deslicing global mixing operators are the key source of accuracy and removing them leads to collapse in performance as pointwise MLPs cannot recover global operators and iii) repeating slicing/deslicing in every layer is necessary. We use the theory of averaging neural operators to explain all these findings, providing a clear understanding of the causal mechanisms for Transolver's function. Finally, we provide a novel and significantly more efficient implementation of the slicing operator by proposing FlashSlice as a computational kernel.

Related Work. Transolver was originally proposed in Wu et al. (2024) and several extensions have been provided by (Luo et al., 2025; Zhou et al., 2026; Adams et al., 2025), among others. The focus of most of these papers is on modifications of the token attention module of Transolver in order to improve performance. On the other hand, there is very little literature on the analysis and understanding of the model itself with the notable exception of Hu et al. (2026), who observe that the physics-attention is a special case of linear attention and find that deleting the slice attention often improves accuracy on six standard PDE benchmarks. However, they attribute this fact to the tying of the projections, and propose LinearNO, which unties the projections and removes the attention; Yang et al. (2026) reach a compatible reading from a low-rank-kernel viewpoint. As argued in Sec. 3, we also investigate this claim and this is clearly not the case. Rather, tying is orthogonal to the redundancy of the attention: with or without it, removing the attention costs nothing measurable, and all four cells of Figure 2(a) lie within 1.05% of one another. Moreover, to the best of our knowledge, our paper is the first to provide theoretical justifications of the extensive empirical findings, leading to a holistic understanding of the mechanisms that govern Transolver's behavior.

Limitations and Future Work. Our paper focuses on one, albeit very widely used, neural operator architecture and the findings about the role of global mixing vs. attention might not carry over to other unrelated neural operators. Moreover, our theory is based on model expressivity. It does not take generalization and optimization into account, nor are the bounds sharp. Yet, we could use the theory to suggest hypotheses which were validated with extensive empirical evidence. Going forward, we would like to leverage the insights presented here to develop novel neural operators that have optimized and non-redundant kernels, leading to further accuracy on AI for industrial engineering.

AI USE STATEMENT

We used generative AI tools for coding assistance, manuscript editing, and assistance with mathematical arguments. The authors reviewed all AI-assisted text and mathematical claims, checked the proofs, and tested AI-assisted code. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix B describes the datasets, metrics, and training settings; Appendix C documents the experimental protocol; and Appendices A and F provide proofs and implementation details. We plan to release the code upon acceptance.

REFERENCES

Corey Adams, Rishikesh Ranade, Sheel Nidhan, Mohammad Amin Nabian, Deepak Akhare, Ram Cherukuri, and Sanjay Choudhry. Geotransolver: Learning physics on irregular domains using multi-scale geometry aware physics attention transformer. *arXiv preprint arXiv:2512.20399*, 2025. URL https://arxiv.org/abs/2512.20399.

Jason Ansel, Edward Yang, Horace He, et al. PyTorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

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

Shuhao Cao. Choose a transformer: Fourier or Galerkin. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Mohamed Elrefaie, Florin Morar, Angela Dai, and Faez Ahmed. DrivAerNet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 2 edition, 2010.

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 (ICML)*, 2023.

Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel De Bezenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

Wenjie Hu, Sidun Liu, Peng Qiao, Zhenglun Sun, and Yong Dou. Transolver is a linear transformer: Revisiting physics-attention through the lens of linear attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2511.06294.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021.

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.

Samuel Lanthaler, Zongyi Li, and Andrew M. Stuart. Nonlocality and nonlinearity implies universality in operator learning. *arXiv preprint arXiv:2304.13221*, 2023.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research (TMLR)*, 2023a.

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

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

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

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

Siddhartha Mishra and Alex Townsend. *Numerical Analysis meets Machine Learning*. Handbook of Numerical Analysis. Elsevier, 2024.

Roberto Molinaro, Samuel Lanthaler, Bogdan Raonić, Tobias Rohner, Victor Armegioiu, Stephan Simonis, Dana Grund, Yannick Ramic, Zhong Yi Wan, Fei Sha, et al. Generative ai for fast and accurate statistical computation of fluids. *arXiv preprint arXiv:2409.18359*, 2024.

Sepehr Mousavi, Shizheng Wen, Levi Lingsch, Maximilian Herde, Bogdan Raonic, and Siddhartha Mishra. Rigno: A graph-based framework for robust and accurate operator learning for pdes on arbitrary domains. *Advances in Neural Information Processing Systems*, 38:150039–150101, 2026.

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

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8: 143–195, 1999.

Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*. Springer, 1994.

Bogdan Raonić, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional neural operators for robust and accurate learning of PDEs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry aware operator transformer as an efficient and accurate neural surrogate for PDEs on arbitrary domains. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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

Zherui Yang, Haiyang Xin, Tao Du, and Ligang Liu. Simple yet effective: Low-rank spatial attention for neural operators. *arXiv preprint arXiv:2604.03582*, 2026. URL https://arxiv.org/abs/2604.03582.

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. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.04940.

APPENDIX

A Slice/deslice as a learnable basis: statements and proofs **14**
A.1 Equation (1) as a low-rank nonlocal neural operator 14
A.2 Proof of Proposition 1, and what it implies 14
A.3 Locality, and the proof of Corollary 1 15
A.4 Proof of Proposition 2, the looseness of the budget, and the frozen basis 15
A.5 From the continuum to the mesh 16
A.6 The learned basis, empirically 16
A.7 Summary 17

B Datasets, metrics, and hyperparameters **17**

C Experimental protocol **18**
C.1 Training budgets 18
C.2 Implementation checks 19
C.3 Full training curves 19

D Additional results **19**
D.1 Pointwise predictions on aerodynamic benchmarks 19
D.2 Field predictions across benchmarks 20
D.3 Per-variable errors 31

E LinearNO implementation and latent width **33**

F FlashSlice: implementation and evaluation **34**
F.1 Kernel design 34
F.2 Numerical validation 35
F.3 Single-GPU performance 36
F.4 Multi-GPU execution and compilation 38

A SLICE/DESLICE AS A LEARNABLE BASIS: STATEMENTS AND PROOFS

This section of the appendix provides details for §4: the placement of Eq. (1) within the nonlocal neural operator template, the proofs of Proposition 1, Corollary 1 and Proposition 2 and further implications and limitations of these results.

A.1 EQUATION (1) AS A LOW-RANK NONLOCAL NEURAL OPERATOR

The ℓ-th hidden layer of the nonlocal neural operator (NNO) of Kovachki et al. (2023) takes the form $\mathcal{L}_\ell(v)(x) = \sigma\big(W_\ell v(x) + b_\ell + \sum_m \langle T_{\ell,m} v, \psi_{\ell,m}\rangle_{L^2}\, \phi_{\ell,m}(x)\big)$; Eq. (1) is the instance with $M = G$, $\psi_{\ell,g} = \tilde{s}_g := s_g / \int_\Omega s_g \,\mathrm{d}\mu$, $\phi_{\ell,g} = s_g$ and $T_{\ell,gh} = A_{gh}$, the special case that Kovachki et al. (2023, §4.2) call the *low-rank neural operator*. Two features distinguish physics-attention within this template, and both matter below: (i) the "basis" $\{s_g\}$ is a learned, data-dependent partition of unity re-estimated from $f^{(\ell-1)}$ at every layer, not a fixed dictionary, nor is it a basis in a rigorous sense, we simply use the term here for narrative convenience; (ii) the coupling is interleaved with a pointwise nonlinearity and a persistent full-resolution pointwise stream, rather than just applied once.

The averaging neural operator (ANO) of Lanthaler et al. (2023) is the simplest member of the family: a single hidden layer with $G = 1$ and a constant, non-adaptive slice function, so that the normalized token is the global mean and Eq. (2) results, with pointwise lifting and projection networks R, Q of width J (their d_c).

As Lanthaler et al. (2023) argue, this minimal averaging neural operator suffices for universality. Their precise statement is reproduced verbatim:

Theorem 1 (Universality of the ANO; Lanthaler et al. 2023, Thms. 2.1–2.2)**.** *Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipschitz domain and $\Psi^\dagger : \mathcal{X} \to \mathcal{Y}$ a continuous operator between C^s or $W^{s,p}$ spaces of functions on Ω. For every compact $\mathsf{K} \subset \mathcal{X}$ and $\epsilon > 0$ there exist $J \in \mathbb{N}$ and networks R, Q of width J such that Ψ of the form (2) satisfies $\sup_{u\in\mathsf{K}} \|\Psi^\dagger(u) - \Psi(u)\|_{\mathcal{Y}} \le \epsilon$.*

The proof of Theorem 1 (Lanthaler et al. 2023, §2.5) proceeds in two steps. (1) Any target operator is $\epsilon/2$-approximated by a finite sum $\sum_{j=1}^{J} \alpha_j(u)\, \eta_j$ of an *encoder*, J continuous nonlinear functionals $\alpha_j : L^1(\Omega;\mathbb{R}^k) \to \mathbb{R}$, against a fixed *decoder* basis $\eta_j \in \mathcal{Y}$. (2) Each functional is $\epsilon/2J$-approximated by one global average of a pointwise-lifted input, $\alpha_j(u) \approx \sigma\big(\int_\Omega R_j(u(y), y)\,\mathrm{d}y\big)$, because averaging against $R_j(u(y), y) = u(y) \cdot \xi_j(y)$ for a smooth ξ_j recovers a mollified inner product with the j-th element of a basis of $L^2(\Omega)$. Step (2) is where all of the nonlocality is spent; step (1), the decoding, is pointwise. The encoding width $J = J(\epsilon)$ is generically unbounded as $\epsilon \to 0$: nothing in the theorem caps how much must be encoded through the single average, and a small fixed G is not the regime it addresses directly. §A.4 locates where that width lives in a Transolver stack, namely in the persistent point stream rather than in the slice count.

A.2 PROOF OF PROPOSITION 1, AND WHAT IT IMPLIES

Proof of Proposition 1. Take the G slice logits identically zero, so that $s_1 = \cdots = s_G \equiv 1/G$, a legal partition of unity reached exactly rather than in a limit. Every normalized token is then the global mean, $z_h = \big(\frac{1}{G}\int_\Omega v\,\mathrm{d}\mu\big)\big/\big(\frac{1}{G}|\Omega|\big) = \bar{v}$, and with the constant core scaled so that $\sum_{g,h} A_{gh}/G = 1$, Eq. (1) reduces exactly to the ANO coupling $(\mathcal{K}v)(x) = \bar{v}$. The pointwise MLP sublayers that precede and follow it realize R and Q up to the usual universal approximation of ordinary networks (Lanthaler et al. 2023, Lemma A.5, after Pinkus 1999), the coordinate x being available to Q through the residual stream, so Theorem 1 applies verbatim. The construction uses no property of G and is available at every $G \ge 1$. □

Proposition 1 is deliberately not the range argument that a similar claim tempts one to make. It is true, and used in §2, that the range of $\mathcal{K}$ is $\mathrm{span}\{s_g\}$ regardless of A, so that a content-dependent core cannot enlarge what a single coupling expresses. That alone shows only that the attention cannot *enlarge* the layer's range, not that it is dispensable, since a constant and a content-dependent core parameterize different maps within the same range; the density statement is what removes the expressivity-based reason to expect the attention to be necessary.

What the proposition does not imply is the stronger reading that the attention therefore costs nothing at the budget a practitioner trains at. Universality is shared by the two classes: NoTokenAttn's class is contained in Baseline's, both are universal, and this density cannot separate them; and the construction spends unbounded pointwise width, which no experiment has. Our own results supply the cautionary case: by the same argument the frozen-basis class is universal (§A.4), and FrozenSlice nonetheless costs up to 17% (Table 1). Universality predicts the absence of a ceiling, not the absence of a cost. That the cost of removing the attention is indistinguishable from zero on nine benchmarks at matched capacity is an empirical finding; the theory makes it unsurprising rather than deriving it from first principles.

A.3 Locality, and the proof of Corollary 1

Lemma 1 (Locality is closed under composition)**.** *Let $\Phi_1, \dots, \Phi_L$ be pointwise operators, $\Phi_\ell(v)(x) = \sigma(W_\ell v(x) + b_\ell)$, possibly with pointwise residual connections. Then $\Phi_L \circ \cdots \circ \Phi_1$ is pointwise: its output at x depends on the input only through the input's value at x.*

Proof. By induction on L: if $\Phi_1(v)(x)$ depends on v only through $v(x)$ and Φ_2 is pointwise, then $\Phi_2(\Phi_1(v))(x)$ depends on $\Phi_1(v)$ only through $\Phi_1(v)(x)$, hence on v only through $v(x)$. □

Proof of Corollary 1. MlpOnly is the pointwise MLP-with-residual stack that remains once $\mathcal{K}$ is removed, so by Lemma 1 it is pointwise at every depth L and every width. The infimum in Definition 1 is taken over a class that is closed in the topology of uniform convergence on K: if Φ_n are pointwise and $\Phi_n \to \Psi$ uniformly on K, then any $u_1, u_2 \in \mathsf{K}$ with $u_1(x) = u_2(x)$ satisfy $\Phi_n(u_1)(x) = \Phi_n(u_2)(x)$ for every n, hence $\Psi(u_1)(x) = \Psi(u_2)(x)$, and Ψ is pointwise. The gap is therefore a genuine obstruction: every pointwise operator, MlpOnly at any size included, has error at least $\delta_{\mathrm{loc}}(\Psi^\dagger)$ on K. □

Following Lanthaler et al. (2023, §2.1), $\delta_{\mathrm{loc}} > 0$ for every operator with genuine nonlocal dependence, the shift $\Psi^\dagger(u)(x) = u(x+h)$ being the canonical example. More relevant here, no pointwise map can represent the solution operator of an elliptic boundary value problem, whose Green's function has support equal to all of Ω. This is exactly the pressure field in incompressible or low-Mach aerodynamics. The MlpOnly penalty of Table 1 is thus a finite-sample, finite-capacity estimate of δ_{loc}, and the per-task and per-channel ranking of that penalty in §3 is a ranking by δ_{loc}: largest on DrivAerML, smallest for Taylor–Green, whose short-horizon map is closest to the identity restricted to a neighborhood of x, and within every aerodynamic task larger for the pressure than for the velocity or shear. One caveat keeps this an ordering rather than a precise prediction: velocity is not local either, being coupled to the same global pressure through ∇p. The empirical results agree with this interpretation where the comparison is cleanest, on SHIFT-SUV (same cars, fixed condition, geometry the only input): the pressure/velocity separation in the volume ($4.5\times$ against $2.6\times$) is narrower than the pressure/shear separation on the surface ($4.2\times$ against $1.3\times$), where the companion quantity is a wall trace far more nearly determined by local geometry (Table D.2). On DrivAerNet++ and DrivAerML the two separations are about equal, and the transonic SHIFT-Wing volume is the exception, at $14.8\times$ against $1.6\times$ for the velocity. Quantifying δ_{loc} directly, for instance through the decay of the governing Green's function, rather than reading it off a trained MlpOnly, is a natural next step we leave open.

Corollary 1 also yields a prediction beyond what we test: parameter-matched MlpOnly models at $L = 4, 8, 16, 32$ should show no improvement trend at all, unlike NoTokenAttn or Baseline. We suggest this depth sweep as a companion to the MlpOnly-Wide capacity control of Figure 2(c), isolating locality from capacity a second, orthogonal way.

A.4 Proof of Proposition 2, the looseness of the budget, and the frozen basis

Theorem 1 reaches universality with a single coupling but a possibly unbounded encoding width J. It is tempting to say that Transolver, which holds G fixed and grows L instead, trades depth for that width. The results do not support this trade: what a coupling transmits is bounded not by G but by G times the channel width, and the encoding is carried, between couplings, in the persistent point stream.

Proof of Proposition 2. (i) is the construction of Proposition 1 read at finite width: the uniform basis $s_g \equiv 1/G$ makes the coupling the global mean in all c channels at once, the preceding pointwise MLP realizes the ANO lifting R into $J \leq c - k$ of them, and the following one realizes Q from those channels together with the coordinates carried in the reserved k. No step varies with G. (ii) The tokens are $z_h = \int s_h v \, \mathrm{d}\mu / \int s_h \, \mathrm{d}\mu \in \mathbb{R}^c$ for $h = 1, \ldots, G$, that is $G\,c$ scalars; deslice returns $\sum_{g,h} s_g(x) A_{gh} z_h$, a linear image of them at each x, and by Lemma 1 no pointwise map applied afterwards adds information about v at a distance. In SLICEONCE those $G\,c$ scalars are produced once and every later layer acts on the G tokens alone, so the bound is independent of that depth; the point-level skip that survives in SLICEONCE carries the pointwise part of the input, not further nonlocal content. □

The budget is an upper bound, and a loose one. At our configuration $G\,c = 32 \times 256 \approx 8 \times 10^3$ scalars pass through a single coupling, which is not obviously too few, so part (ii) does not on its own account for the size of the SLICEONCE penalty (2.2–6.8×). A second structural difference does, and Theorem 1 makes it explicit: the ANO needs a universal pointwise decoder Q acting at full resolution on (encoding, x). SLICEONCE spends its depth transforming the G tokens and then reads out affinely at the points, so what it realizes is a nonlinear map of the encoding followed by a map that is linear in the tokens with x-dependent coefficients $s_g(x)$; the second ingredient of the theorem is missing, not merely narrow. What part (ii) establishes is the qualitative asymmetry, budgets 0, $G\,c$ and $L\,G\,c$, and the prediction that only the third improves with depth; the magnitude is not something we derive.

The frozen basis. Part (i) uses a constant, non-adaptive basis, $s_g \equiv 1/G$ fixed once as in FROZENSLICE, and still attains universality, because all the adaptivity it needs comes from the pointwise networks between rounds. This matches the ANO itself, whose single nonlocal ingredient is a fixed uniform average. Read literally, it predicts that a frozen basis is not subject to a hard ceiling analogous to MLPONLY or SLICEONCE: given enough depth, a fixed basis reused at every layer can still reach any target. What re-estimating $\{s_g\}$ from $f^{(\ell-1)}$ buys is efficiency: a data-dependent basis can spend each of its L rounds probing whichever G-dimensional projection of the current field is most informative, whereas a frozen basis probes the same projections every round and relies on the pointwise nonlinearity between rounds to make each repetition informative. This is the distinction of §A.2, no ceiling but a cost, and the cost is what Table 1 measures, at a few percent to 17%. It also names the discriminating experiment: under a depth sweep the MLPONLY and SLICEONCE gaps should persist at every L, while the FROZENSLICE gap should close. We state this as a prediction; the depth sweep is not among the runs we report.

A.5 FROM THE CONTINUUM TO THE MESH

Theorem 1 and Propositions 1–2 are stated for the continuum operator $\mathcal{K}$; the implementation sees only N mesh points. Writing the discrete slice weights as quadrature weights, $w_{i,g}\,\omega_i \approx s_g(x_i)\,\mathrm{d}\mu(x_i)$ for a per-point volume element ω_i with $\sum_i \omega_i = |\Omega|$, the discrete coupling $\sum_i w_{i,g} v_i / \sum_i w_{i,g}$ is a Riemann-sum quadrature of $\int_\Omega s_g(y) v(y)\,\mathrm{d}\mu(y) / \int_\Omega s_g(y)\,\mathrm{d}\mu(y)$, with error controlled, for Lipschitz s_g and v, by the fill distance of $\{x_i\}$ in Ω, the same collocation argument used to pass from continuum to discrete FNOs in Kovachki et al. (2021). We do not develop explicit rates here, since our point clouds are highly non-uniform unstructured surface and volume meshes for which fill-distance bounds would be conservative. Instead we note that the subsampling-noise measurement of Table D.1 is an empirical measurement of exactly this discretization error, and that its magnitude ($\leq 0.18\%$ relative, three orders of magnitude below every ablation effect in Table 1) is consistent with the discretization term being negligible relative to the architectural effects this paper studies.

A.6 THE LEARNED BASIS, EMPIRICALLY

We compare the slice functions learned by independently trained BASELINE and NOTOKENATTN on DrivAerML. At the last layer, both models produce spatially coherent patterns corresponding to recognizable parts of the car, including the greenhouse and upper body, front fascia, and flanks (Figure A.1). Matching slices by cosine similarity over points gives similarities of 0.82, 0.72, and 0.69 for the three closest pairs.

At the first layer, assignments are nearly uniform in both models: the mean per-point entropies are 3.47 and 3.46 nats, close to the maximum $\ln 32 = 3.47$, and all 32 slices are effectively used. The spatial patterns therefore develop in later layers. At the last layer, NoTokenAttn uses more effective slices than Baseline (20.4 vs. 15.4), but has lower mean per-point entropy (2.22 vs. 2.48 nats). Thus, it uses more slices overall while making more concentrated assignments at individual points. This is consistent with the interpretation in §A.4: when the core is held constant, the learned basis may take on a greater role in input-dependent routing.

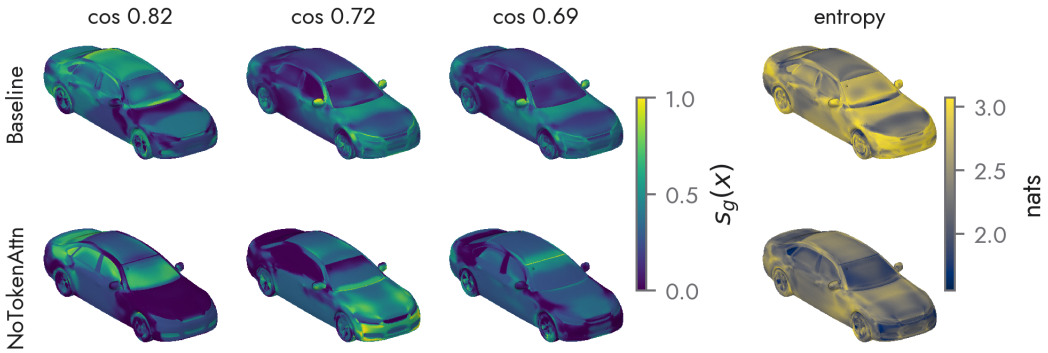

Figure A.1: **Learned slice functions with and without attention.** Last-layer slice functions $s_g(x)$ for Baseline (top) and NoTokenAttn (bottom), shown on the DrivAerML test car from Figure D.1. The first three columns show the closest pairs under Hungarian matching by cosine similarity over points (0.82, 0.72, and 0.69). Each slice is normalized by its own maximum. The rightmost column shows per-point assignment entropy on a shared color scale.

A.7 Summary

Table A.1 separates what the theory settles from what only experiments can.

Table A.1: **What the theory settles, and what it leaves to experiment.** Nonlocal budget and hard ceiling per variant, against the measured cost relative to Baseline in Table 1.

variant	nonlocal budget	hard ceiling?	measured cost
Baseline / Untied	LGc	no (Prop. 1)	reference
NoTokenAttn	LGc	no (Prop. 1)	-2.6% to $+7.9\%$
FrozenSlice	LGc	no (§A.4)	-0.5% to $+17\%$
SliceOnce	Gc	yes, rank (Prop. 2)	2.2–6.8×
MlpOnly	0	yes, locality (Cor. 1)	1.04–5.6×

B Datasets, Metrics, and Hyperparameters

Benchmarks. Table B.1 summarizes the nine benchmark configurations, including data splits, training budgets, point counts after the `physical_domain` crop, and input and output fields. Splits follow the standard protocol for each dataset. During training, a fresh random subset of points is drawn each time a sample is loaded, using the ratios listed in the table. The measured relative variation in the evaluation metric due to subsampling is at most 0.18% (Table D.1). All listed input features, including freestream conditions where provided, are available at every point. SHIFT-SUV provides no freestream attributes and uses a single fixed flow condition, so its input features consist only of normals (surface) or signed distance (volume).

Metric. We report relative L^1 error, computed separately for each sample s and output group g, then averaged: $\mathrm{err} = \mathrm{mean}_{s,g} \|\hat{u}_{s,g} - u_{s,g}\|_1 / \|u_{s,g}\|_1$. For the aerodynamic benchmarks, the

Table B.1: **Benchmark configurations.** Training steps, split sizes, points per sample after cropping and before subsampling, training subsampling ratios, and input/output channels. *Evaluated at the common checkpoint of 140k steps, reached within one 12-hour run using a 250k-step cosine schedule (Appendix C.1).

Benchmark	steps	train/val/test	points/sample	subsample	in→out ch.	inputs	outputs
Taylor–Green	250k	1.55M/19.3k/38.6k	32^3	1.0	5→5	state at t (+ lead time)	state at $t+\Delta t$
SHIFT-Wing surface	250k	8377/1000/1000	1.6M	0.1	8→4	normals, freestream (varying)	p, τ
SHIFT-Wing volume	140k*	8377/1000/1000	5.85M	0.046	6→4	SDF, freestream (varying)	p, u, v, w
DrivAerNet++ surface	184k	5825/1148/1154	459k	0.5	8→4	normals, freestream (const.)	p, τ
DrivAerNet++ volume	180k	5825/1148/1154	11.0M	0.02	6→4	SDF, freestream (const.)	p, u, v, w
SHIFT-SUV surface	50k	798/100/100	2.57M	0.092	3→4	normals *only*	p, τ
SHIFT-SUV volume	50k	798/100/100	46.5M	0.0059	1→4	SDF *only*	p, u, v, w
DrivAerML surface	50k	400/34/50	7.8M	0.03	8→4	normals, freestream (const.)	p, τ
DrivAerML volume	19k	400/34/50	133M	0.002	6→4	SDF, freestream (const.)	p, u, v, w

groups are pressure and the velocity or wall-shear vector. For Taylor–Green, they are density and the velocity vector.

Model. The baseline uses a Transolver backbone with 8 layers, hidden width 256, 8 heads, 32 slices, an MLP ratio of 2, GELU activations, and no dropout (approximately 3.87M parameters). MLPONLY removes the attention sublayer (2.25M parameters), and MLPONLY-WIDE increases the MLP ratio to 4 (4.35M). SLICEONCE applies slicing once, followed by an 8-layer token transformer and a single deslicing operation (4.37M).

The UNTIED-LN variant. For UNTIED-LN, we use the released layer implementation of Hu et al. (2026), integrated without changes into our training framework (Appendix E). Relative to the baseline, this layer gives deslicing its own projection and temperature, removes token self-attention, and normalizes the analysis weights with a separate softmax over the N points (Figure E.1(a)). The last change distinguishes UNTIED-LN from combining UNTIED and NOTOKENATTN, which normalizes the per-point slice weights by their sums over points. The resulting model has 3.85M parameters.

Training. We use AdamW with a learning rate of 10^{-3}, cosine decay, 1% warmup, weight decay 5×10^{-5}, and gradient clipping at 5.0. Training uses a per-device batch size of 1–5, depending on the benchmark. Each run uses data parallelism on a single node with 4 H100 GPUs. Table B.1 lists the training step budgets.

C EXPERIMENTAL PROTOCOL

C.1 TRAINING BUDGETS

Table B.1 lists the training budget for each benchmark. Variants use the same number of steps unless noted below.

We assess budget sensitivity on SHIFT-SUV by comparing results from 10k to 50k steps. Over this range, the error ratio of MLPONLY to the baseline increases from 1.52 to 1.62 on the surface and from 2.43 to 3.24 in the volume. The corresponding ratios for NOTOKENATTN remain close to one: 0.996–1.009 on the surface and 1.015–1.035 in the volume. Thus, on this benchmark, extending training does not close the MLPONLY gap or substantially change the NOTOKENATTN comparison.

The following runs differ from the budgets in Table B.1:

- On DrivAerNet++, MLPONLY reaches 250k steps on the surface and 204k in the volume, compared with approximately 184k and 180k for the baseline, under equal wall-clock budgets.
- On DrivAerML volume, the MLPONLY run ends at 16.5k steps after its validation error plateaus. At the common checkpoint of 12.5k steps, its error is 4.1 times the baseline.

- On SHIFT-Wing surface, UNTIED ends at 234k of the planned 250k steps.

On SHIFT-Wing volume, all variants are evaluated at 140k steps, the last evaluation reached by every variant within one 12-hour run. These runs use a 250k-step cosine schedule. On SHIFT-SUV volume, the error ratios of MLPONLY and SLICEONCE to the baseline are still increasing at the 50k-step cutoff. Figure C.1 shows the validation curves for all benchmarks.

C.2 IMPLEMENTATION CHECKS

We verify each ablation using model structure, checkpoint contents, and parameter updates. Module and parameter-key checks confirm that the intended components are present or absent. For NOTOKENATTN and FROZENSLICE, we also check that parameters excluded from the computation remain unchanged between training checkpoints, while active parameters update. Assertions during model construction check that the requested ablation matches the instantiated architecture. Baseline replicas provide a reference for run-to-run variation.

C.3 FULL TRAINING CURVES

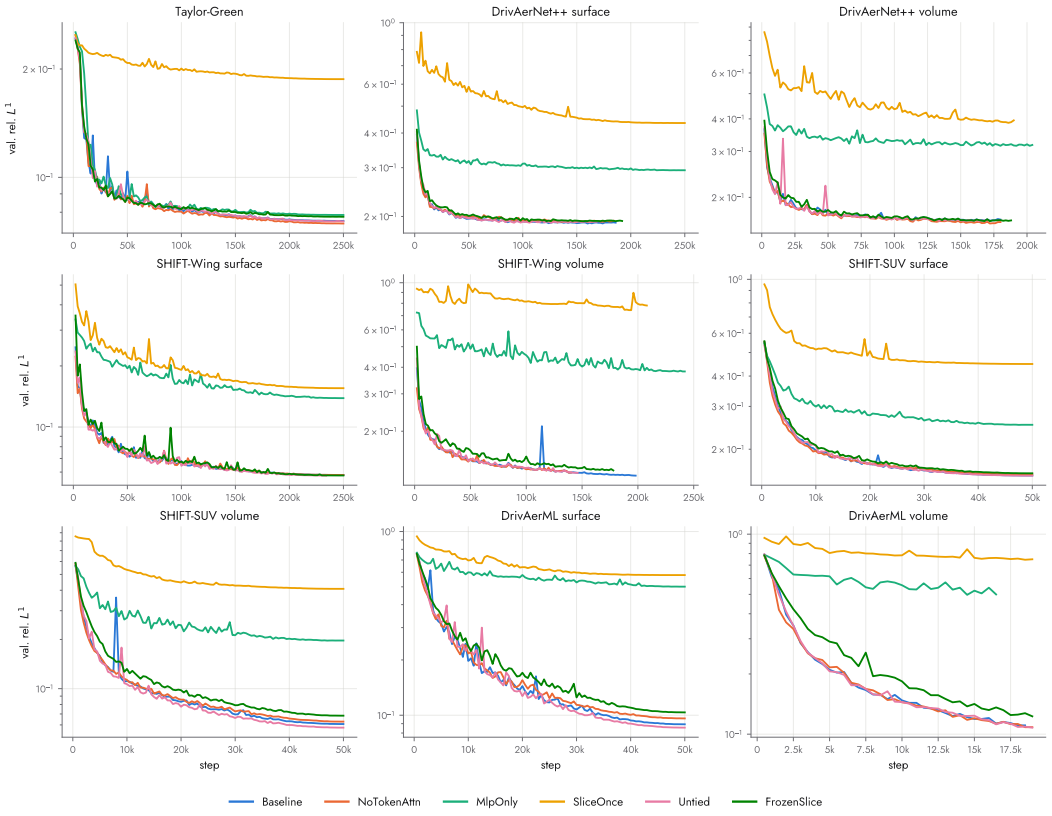

Figure C.1: Validation curves for all variants on the nine benchmark configurations (seed 42). Training budgets and exceptions are described in §C.1.

D ADDITIONAL RESULTS

D.1 POINTWISE PREDICTIONS ON AERODYNAMIC BENCHMARKS

A pointwise model receives the input features at each query point, including freestream conditions where provided (Table B.1), but cannot combine information across points. Local geometric features, such as surface normals and signed distance, do not generally determine the surrounding

flow: the field also depends on the rest of the body. This dependence remains when the inflow is fixed but the geometry varies between samples. Consistent with this, the largest MLPONLY error increase occurs on DrivAerML, which uses fixed inflow, rather than on the varying-inflow SHIFT-Wing benchmark (Table 1).

On the DrivAerML example in Figure D.1, MLPONLY smooths the wheel-arch suction and front stagnation patterns, with errors around the wheels, mirror, and A-pillar. The per-variable results in Figure 2(d) show a larger relative error increase for pressure than for velocity or wall shear across the aerodynamic benchmarks. These observations are consistent with a loss of information about the overall geometry. SLICEONCE shows a different distribution of errors across fields, suggesting that its limitations are not fully explained by the pointwise model's lack of spatial coupling.

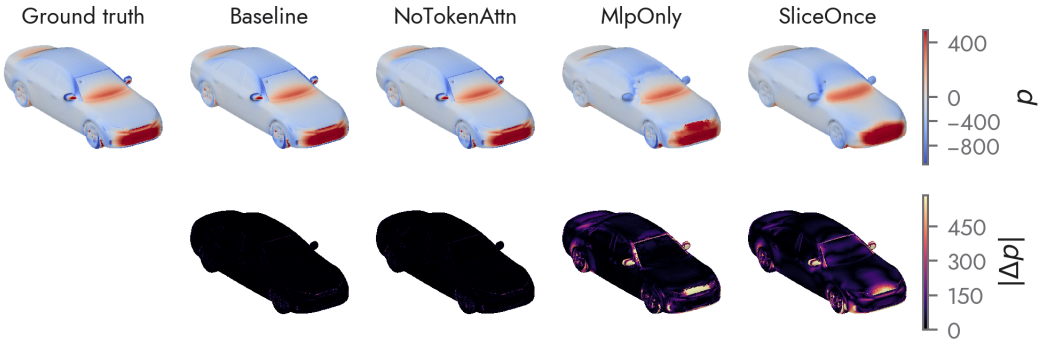

Figure D.1: **Surface pressure on a DrivAerML test car.** Top: ground truth and the four variants. Bottom: each variant's absolute error. The car is `run_472`. Appendix D.2 states the protocol and repeats the figure on the other eight benchmarks.

D.2 FIELD PREDICTIONS ACROSS BENCHMARKS

Figures D.2–D.17 extend the comparison in Figure D.1 to the remaining benchmarks and output fields. For each aerodynamic benchmark, we show pressure and either wall shear or velocity; for Taylor–Green, we show velocity. On the selected aerodynamic samples, the error of MLPONLY relative to the baseline is approximately 1.8–20 times larger for pressure and 1.3–5.1 times larger for the corresponding vector field.

Sample selection and evaluation. For each benchmark, we select the sample with median baseline relative L^1 error among the first fifty test samples (the full test split for DrivAerML). All evaluations use fp32 and the training subsampling ratios in Table B.1. For visualization, we pool predictions from several subsample seeds to obtain a denser point cloud while retaining the training point density in each model evaluation.

Rendering. Surfaces are rendered by depth-buffered point splatting with depth-based shading. Volume sections interpolate points within a thin slab around the displayed plane; the body is grey, and regions without sufficient point coverage are white. Within each figure, prediction panels share a color scale, and error panels share a separate scale spanning all variants. Reported errors are for the selected sample and therefore differ from the test-set averages in Table D.2.

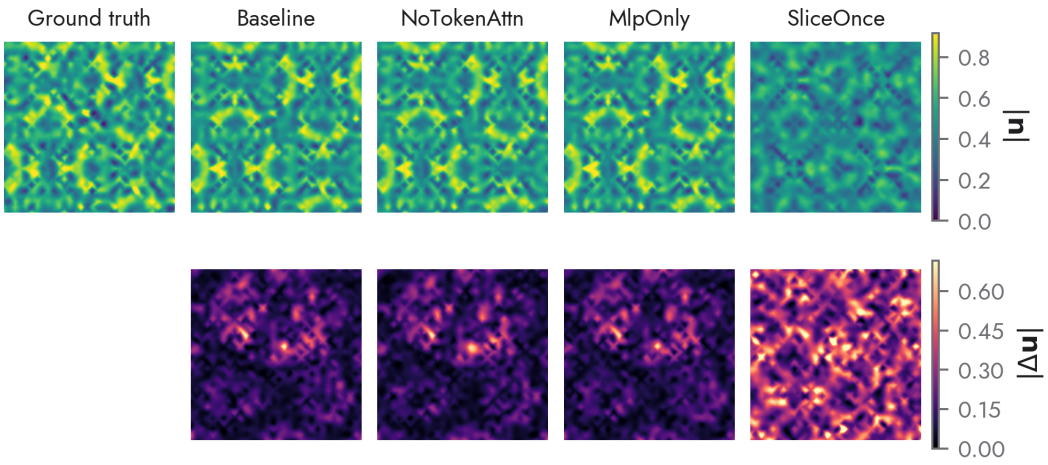

Figure D.2: **Taylor–Green velocity magnitude** on a mid-plane of the 32^3 cube, predicted from $t = 1.28$ to $t = 1.36$. Relative L^1 errors: BASELINE 0.149, NOTOKENATTN 0.139, MLPONLY 0.148, SLICEONCE 0.450. SLICEONCE smooths the fine vortex structure.

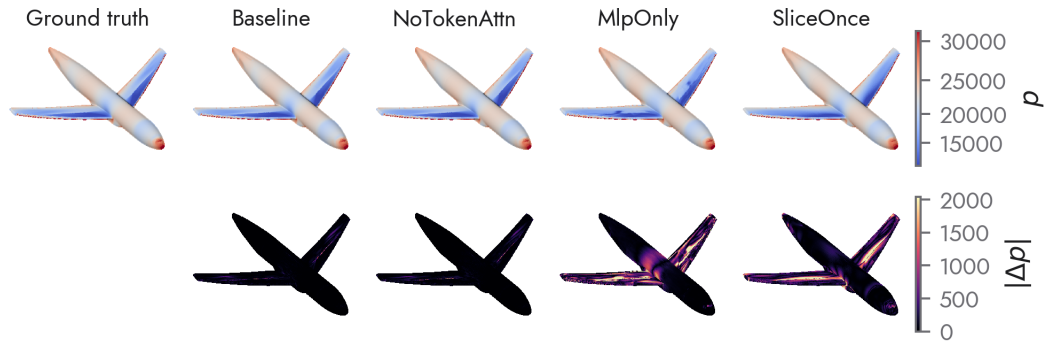

Figure D.3: **SHIFT-Wing surface pressure** for sample `016040`. Relative L^1 errors: BASELINE 0.00085, NOTOKENATTN 0.00088, MLPONLY 0.0098, SLICEONCE 0.0086. The latter two smooth the leading-edge pressure pattern.

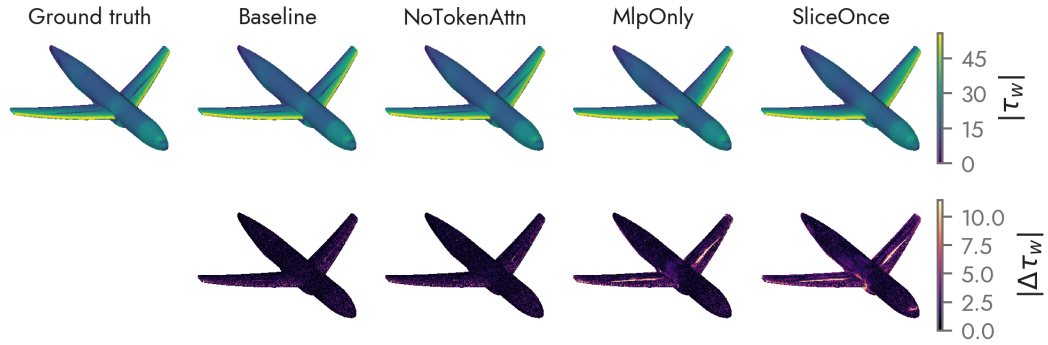

Figure D.4: **SHIFT-Wing surface wall shear** for the sample in Figure D.3. Relative L^1 errors: BASELINE 0.057, NOTOKENATTN 0.057, MLPONLY 0.082, SLICEONCE 0.104.

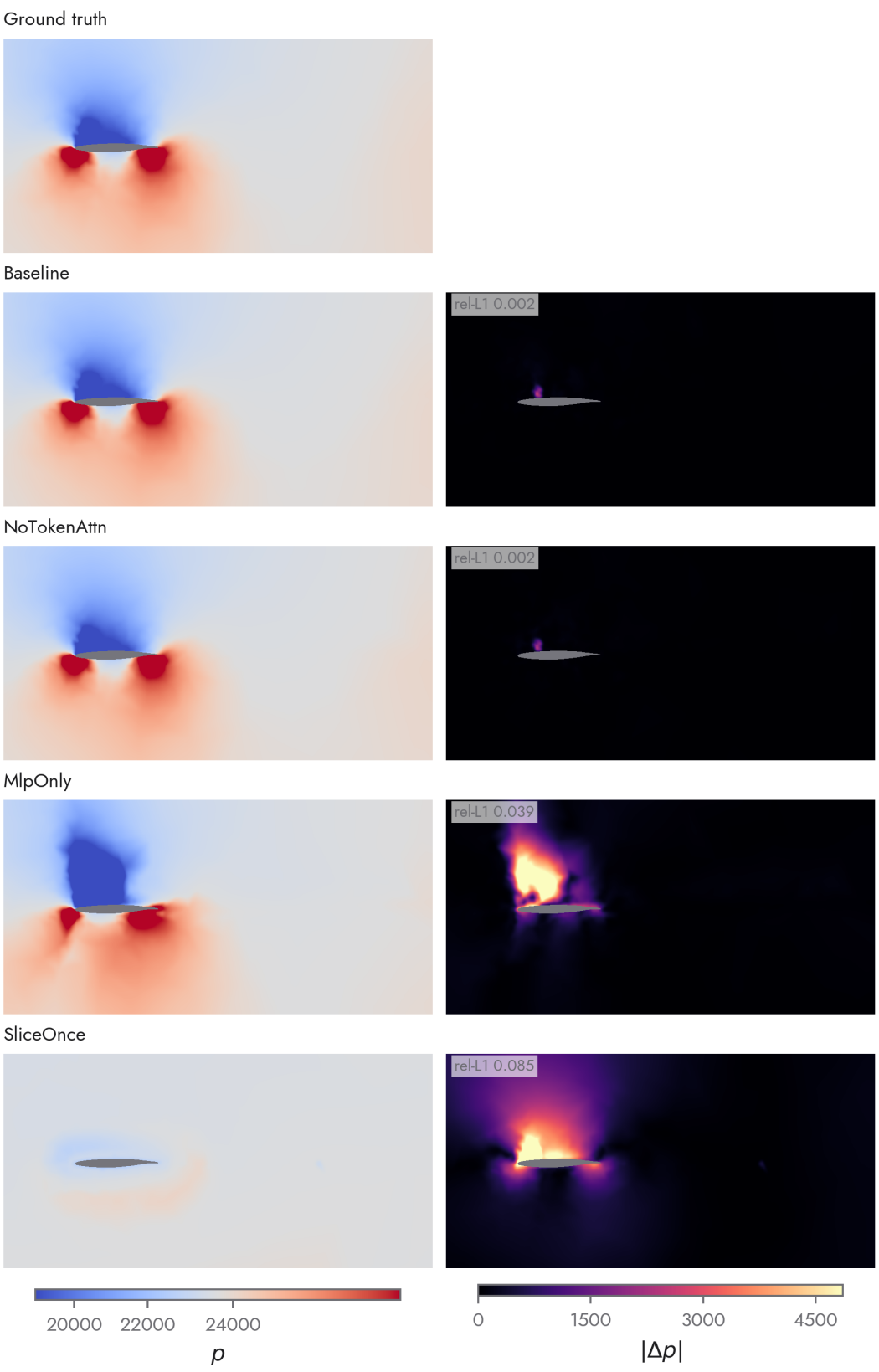

Figure D.5: **SHIFT-Wing volume pressure** at $y = 15.3$ for sample `016032`. Relative L^1 errors: BASELINE 0.0019, NOTOKENATTN 0.0022, MLPONLY 0.039, SLICEONCE 0.085. SLICEONCE produces a nearly uniform field.

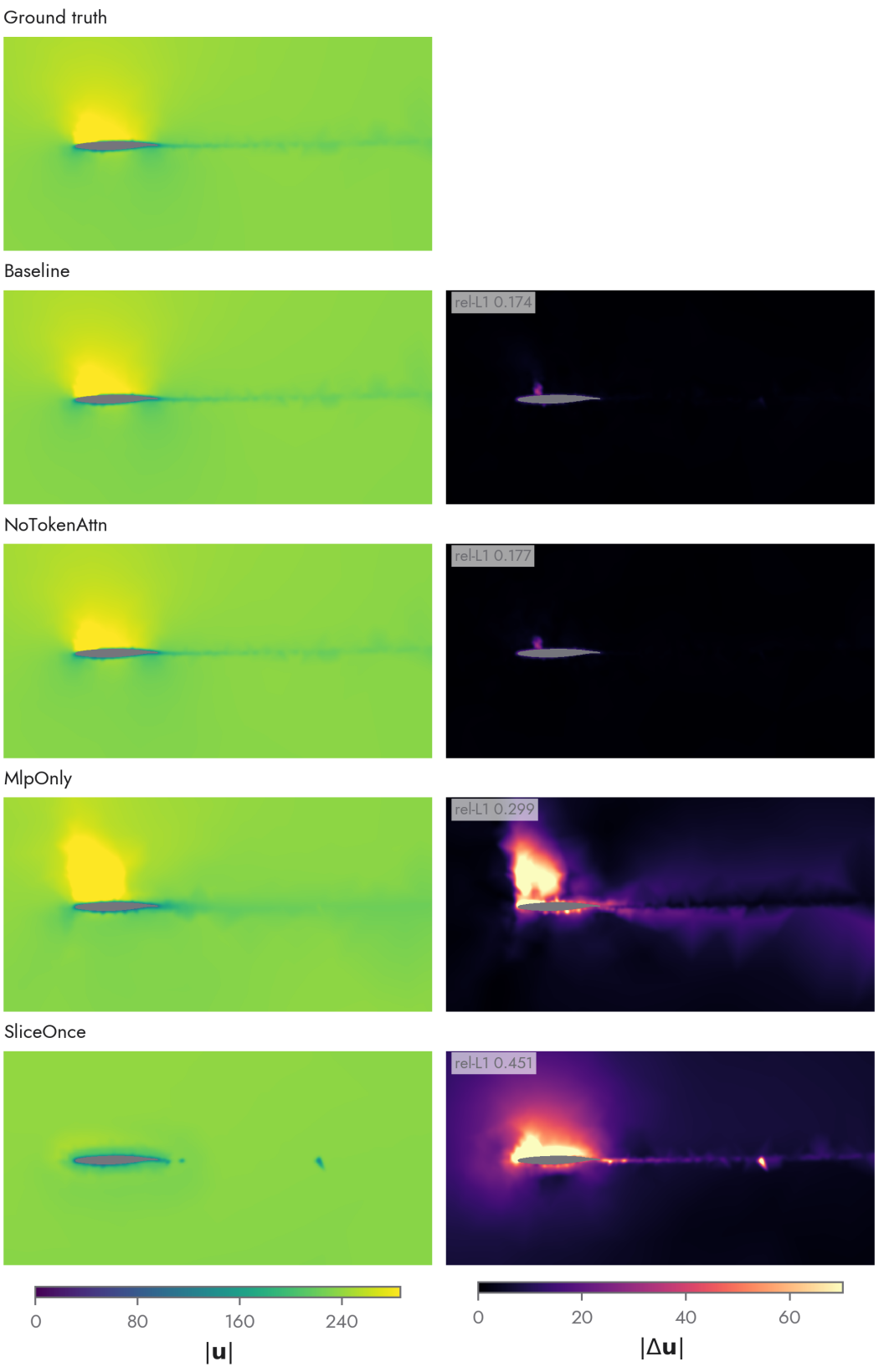

Figure D.6: **SHIFT-Wing volume velocity magnitude** on the section in Figure D.5. Relative L^1 errors: BASELINE 0.174, NOTOKENATTN 0.177, MLPONLY 0.299, SLICEONCE 0.451.

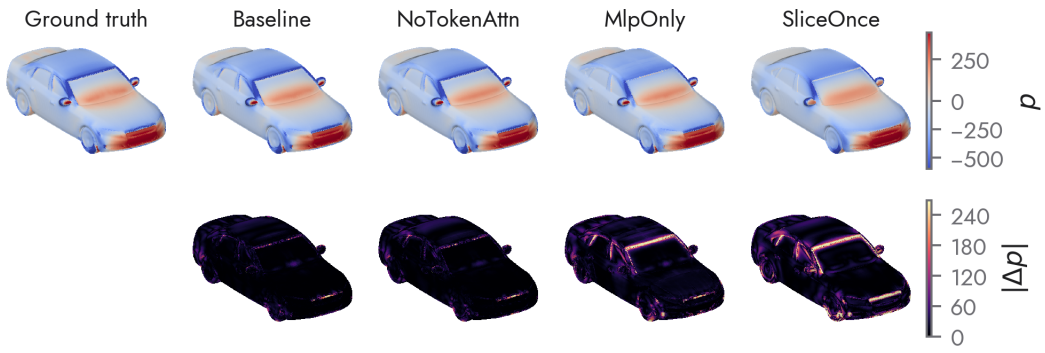

Figure D.7: **DrivAerNet++ surface pressure** for car `N_S_WWC_WM_388`. Relative L^1 errors: BASELINE 0.095, MLPONLY 0.168, SLICEONCE 0.210.

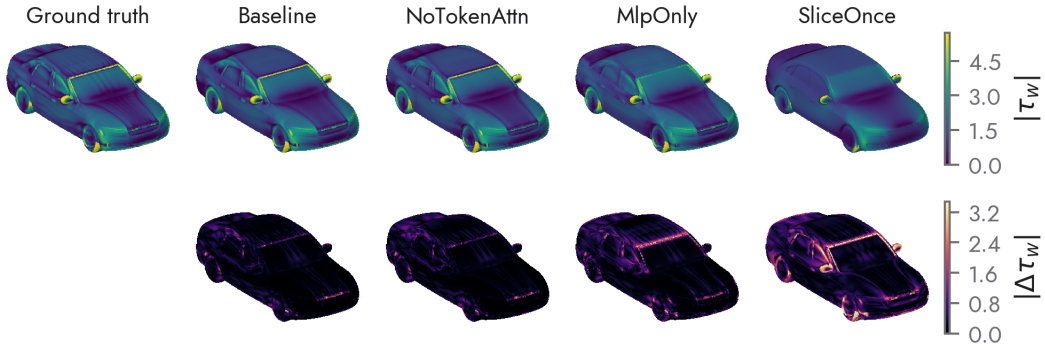

Figure D.8: **DrivAerNet++ surface wall shear** for the car in Figure D.7. Relative L^1 errors: BASELINE 0.194, NOTOKENATTN 0.204, MLPONLY 0.269, SLICEONCE 0.476.

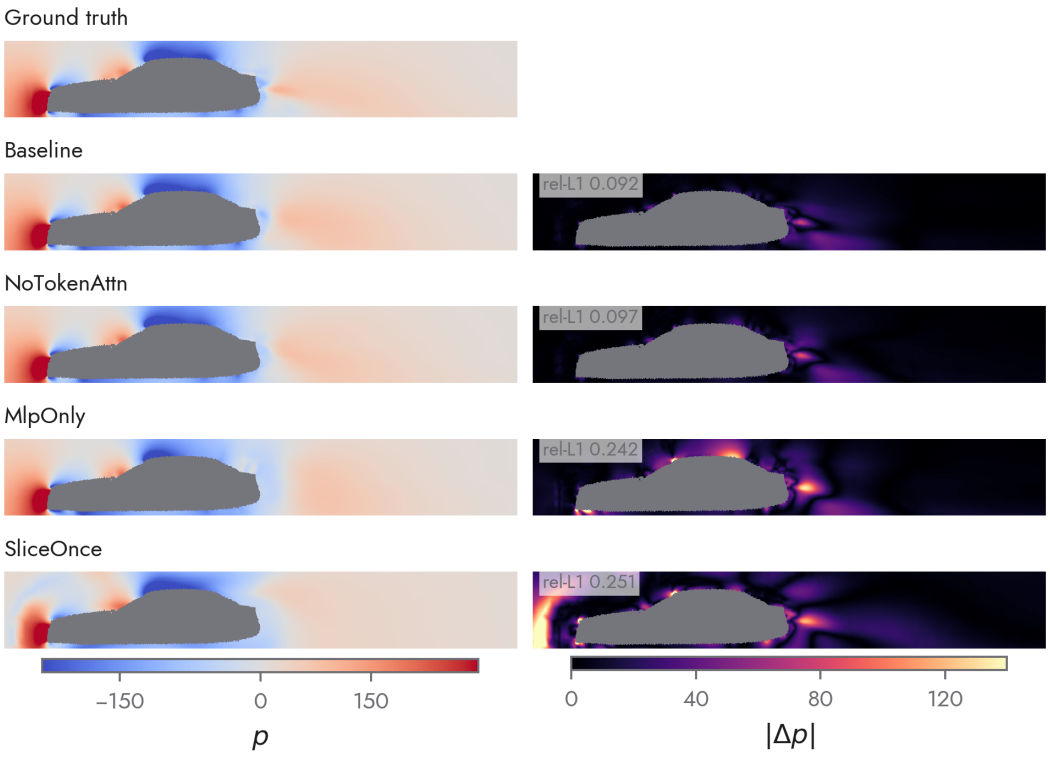

Figure D.9: **DrivAerNet++ volume pressure** on the symmetry plane for car `N_S_WWC_WM_391`. Relative L^1 errors: BASELINE 0.092, NOTOKENATTN 0.097, MLPONLY 0.242, SLICEONCE 0.251.

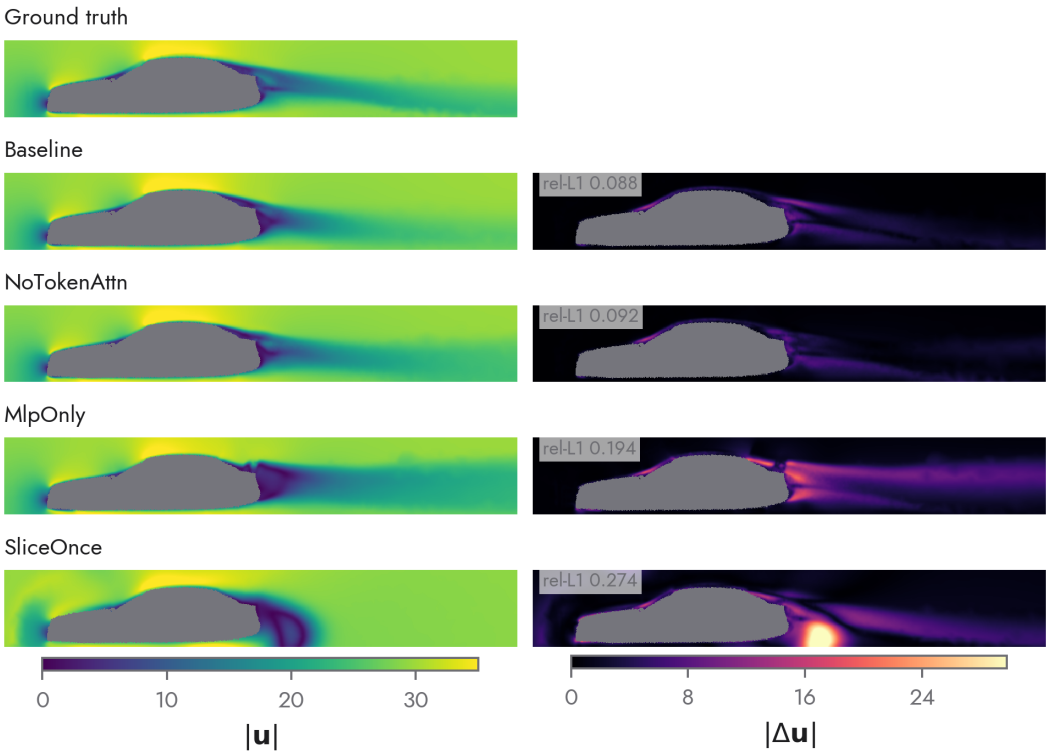

Figure D.10: **DrivAerNet++ volume velocity magnitude** on the plane in Figure D.9. Relative L^1 errors: BASELINE 0.088, NOTOKENATTN 0.092, MLPONLY 0.194, SLICEONCE 0.274.

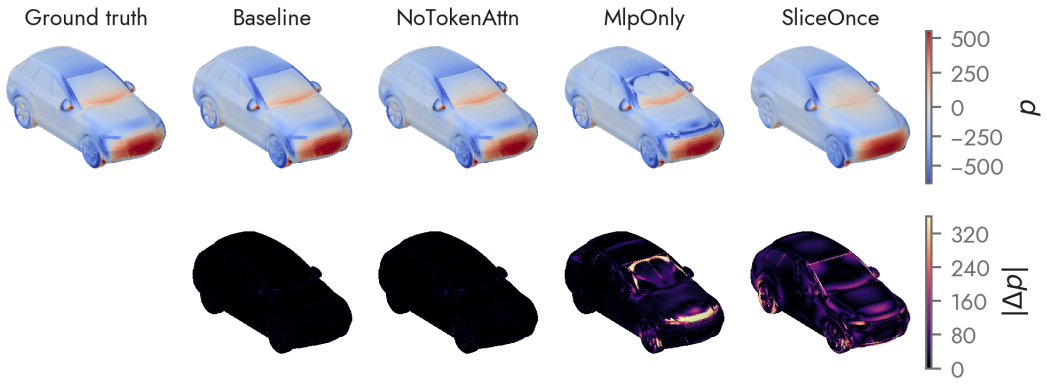

Figure D.11: **SHIFT-SUV surface pressure** for car `run_00939`, with surface normals as input. Relative L^1 errors: BASELINE 0.024, NOTOKENATTN 0.024, MLPONLY 0.113, SLICEONCE 0.195.

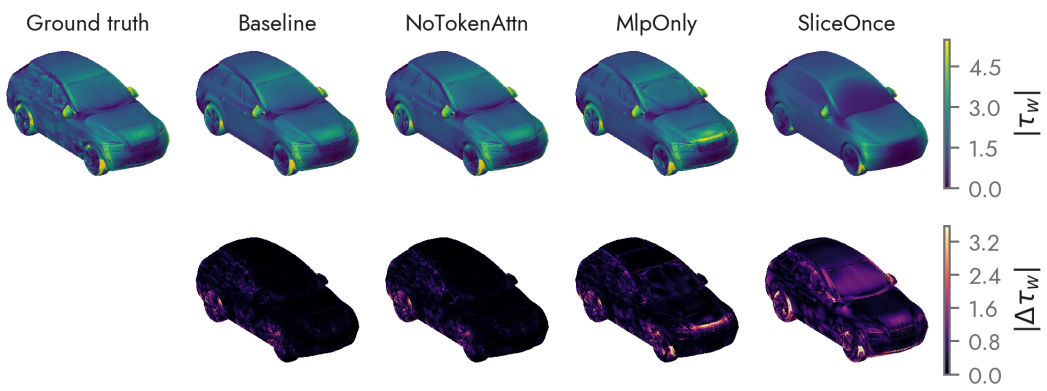

Figure D.12: **SHIFT-SUV surface wall shear** for the car in Figure D.11. Relative L^1 errors: BASELINE 0.256, NOTOKENATTN 0.257, MLPONLY 0.328, SLICEONCE 0.558.

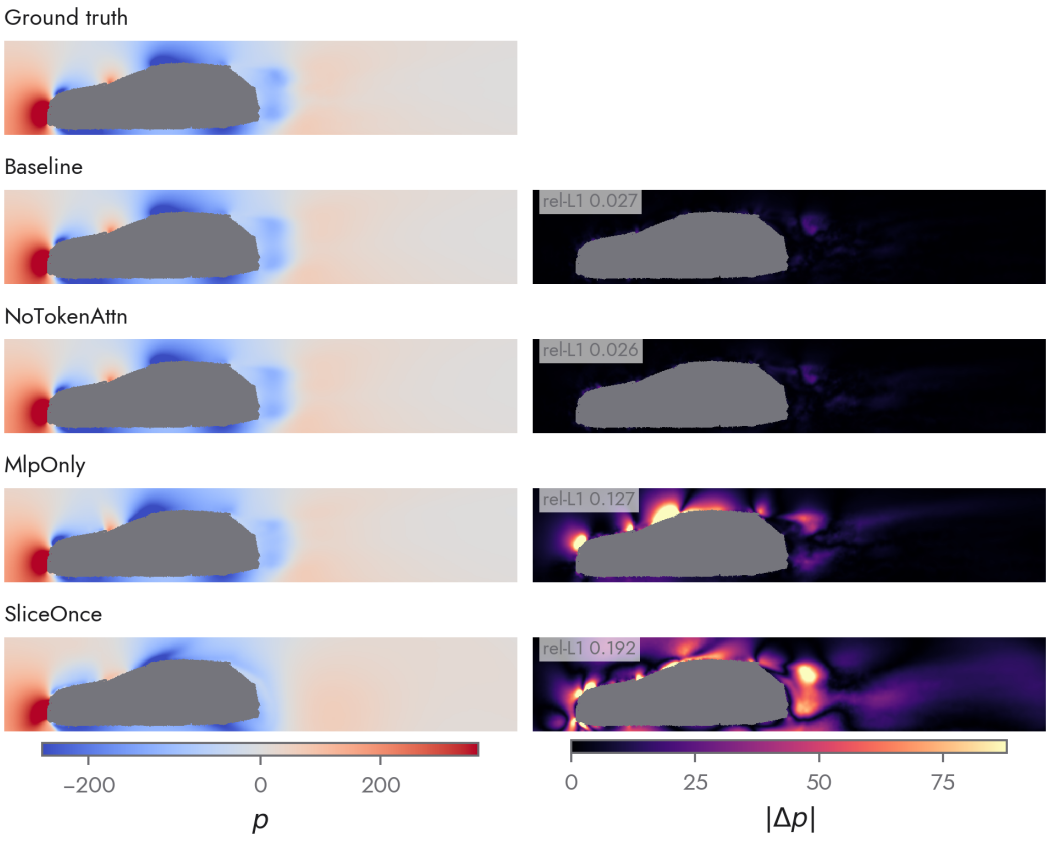

Figure D.13: **SHIFT-SUV volume pressure** on the symmetry plane for car `run_00939`, with signed distance as input. Relative L^1 errors: BASELINE 0.027, NOTOKENATTN 0.026, MLPONLY 0.127, SLICEONCE 0.192.

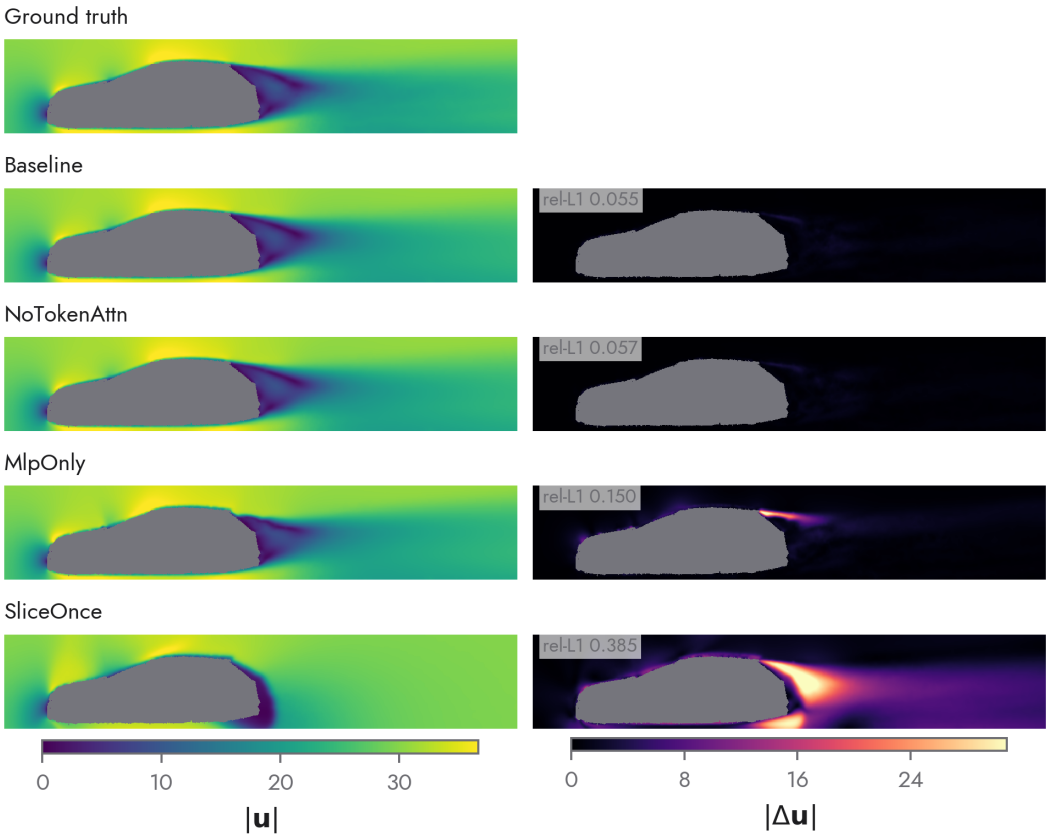

Figure D.14: **SHIFT-SUV volume velocity magnitude** on the plane in Figure D.13. Relative L^1 errors: BASELINE 0.055, NOTOKENATTN 0.057, MLPONLY 0.150, SLICEONCE 0.385.

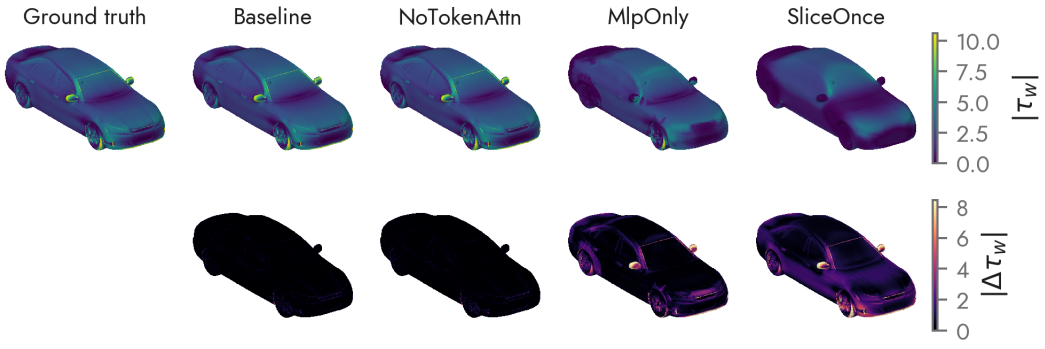

Figure D.15: **DrivAerML surface wall shear** for the car in Figure D.1. Relative L^1 errors: BASELINE 0.107, NOTOKENATTN 0.115, MLPONLY 0.546, SLICEONCE 0.736.

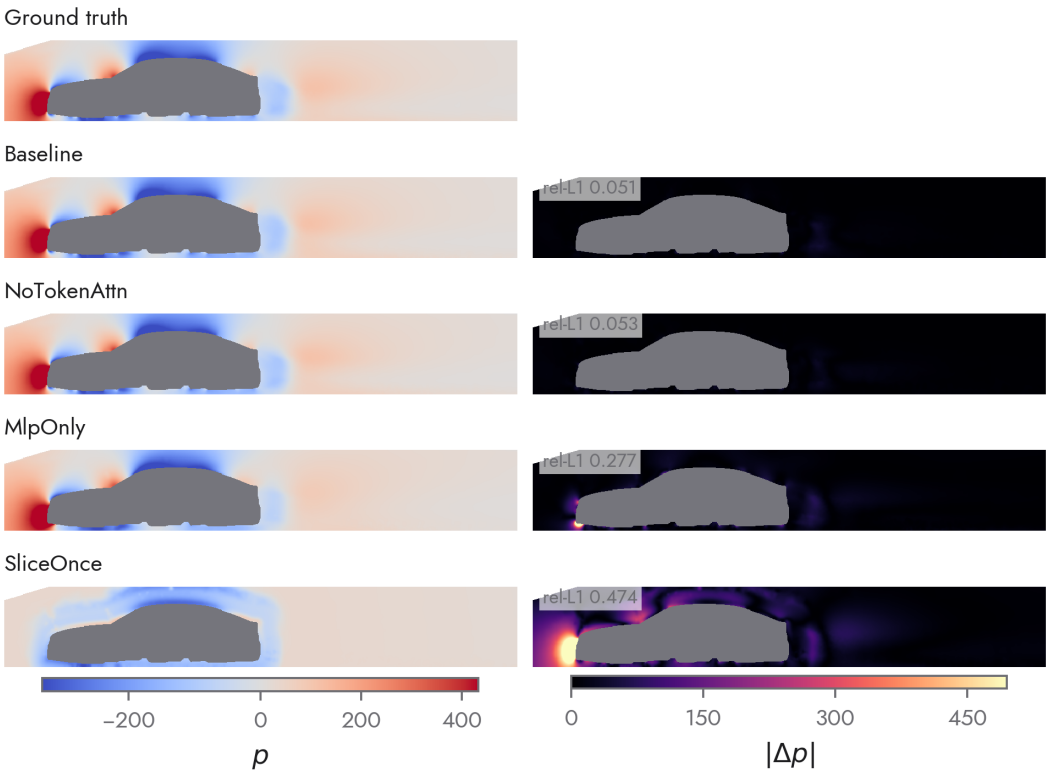

Figure D.16: **DrivAerML volume pressure** on the symmetry plane for car `run_108`. Relative L^1 errors: BASELINE 0.051, NOTOKENATTN 0.053, MLPONLY 0.277, SLICEONCE 0.474.

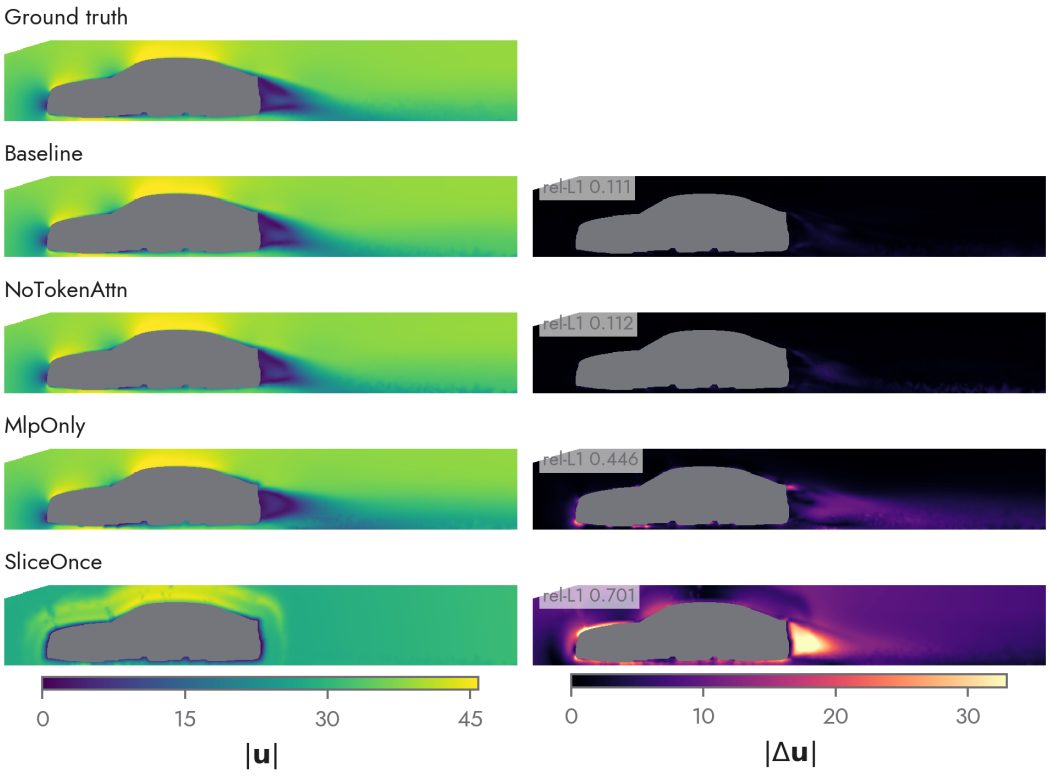

Figure D.17: **DrivAerML volume velocity magnitude** on the plane in Figure D.16. Relative L^1 errors: BASELINE 0.111, NOTOKENATTN 0.112, MLPONLY 0.446, SLICEONCE 0.701.

D.3 PER-VARIABLE ERRORS

Table 1 reports a composite relative error on standardized outputs. Here we evaluate each physical variable separately after reversing the output normalization. The resulting relative errors are dimensionless. Table D.2 reports relative L^1 errors, and Table D.3 reports relative L^2 errors, with scalar fields evaluated individually and vector fields as groups. Both metrics support the main comparison: NOTOKENATTN remains close to the baseline across variables, whereas MLPONLY shows larger error increases for pressure than for velocity or wall shear.

Evaluation. We evaluate the test split at the training subsampling ratio, using the selected checkpoint for each run. Results use seed 42, except for SHIFT-Wing volume, which uses seed 43. Taylor–Green is evaluated at full resolution.

Per-variable comparisons. Across the aerodynamic benchmarks, the pressure error of MLPONLY is 1.8–14.8 times the baseline error, compared with 1.3–5.0 times for velocity or wall shear. The largest contrast occurs on SHIFT-Wing volume: 14.8 times for pressure and 1.6 times for velocity. Taylor–Green shows a smaller effect; unlike the aerodynamic benchmarks, its inputs include the current field state.

For SHIFT-Wing, pressure errors are computed using dimensional pressure, whose large ambient offset ($p_\infty \approx 23.8\,\mathrm{kPa}$) dominates the denominator. This makes the relative pressure errors small. Comparisons between variants on the same benchmark are therefore more informative than comparisons of absolute relative errors across benchmarks.

Subsampling variability. We repeat each aerodynamic baseline evaluation with six independent subsample seeds (Table D.1). The relative standard deviation of the per-variable relative L^1 error is at most 0.18%. This indicates low sensitivity to the sampled point set at the evaluation ratios used here, but does not measure the difference from full-resolution evaluation.

Table D.1: **Subsampling noise of the evaluation metric.** Baseline checkpoint (seed 43 on SHIFT-Wing volume), test split, six independent subsample seeds; mean$\pm$std of the per-variable relative L^1 error.

Benchmark	p	$\mathbf{u}$ / $\boldsymbol{\tau}_w$	max rel. std
DrivAerNet++ surface	$0.10170 \pm .00003$	$0.19222 \pm .00004$	0.02%
DrivAerNet++ volume	$0.10634 \pm .00004$	$0.10096 \pm .00002$	0.04%
DrivAerML surface	$0.03876 \pm .00006$	$0.11286 \pm .00009$	0.16%
DrivAerML volume	$0.05184 \pm .00007$	$0.11112 \pm .00008$	0.14%
SHIFT-Wing surface	$0.001018 \pm .000001$	$0.05863 \pm .00001$	0.10%
SHIFT-Wing volume	$0.002251 \pm .000001$	$0.17355 \pm .00002$	0.03%
SHIFT-SUV surface	$0.02709 \pm .00002$	$0.26064 \pm .00006$	0.07%
SHIFT-SUV volume	$0.02986 \pm .00006$	$0.05827 \pm .00005$	0.18%

Table D.2: **Per-variable relative L^1 test error in physical units.** Best per row in bold; protocol and reading notes in §D.3.

Dataset	Variable	baseline	no-token-attn	MLP-only	untied	frozen	slice-once
Taylor–Green	ρ	0.00023	0.00035	0.00024	0.00029	**0.00022**	0.00026
	$\mathbf{u}$	0.1420	**0.1397**	0.1474	0.1416	0.1459	0.3581
DrivAerNet++ surf.	p	**0.1017**	0.1057	0.1830	0.1050	0.1047	0.2172
	$\boldsymbol{\tau}_w$	**0.1922**	0.1981	0.2759	0.1928	0.1992	0.4635
DrivAerNet++ vol.	p	0.1064	**0.0953**	0.2188	0.0979	0.0963	0.2253
	$\mathbf{u}$	0.1010	0.0940	0.1639	0.0934	**0.0933**	0.2418
DrivAerML surf.	p	0.0387	0.0417	0.2622	**0.0373**	0.0440	0.2592
	$\boldsymbol{\tau}_w$	0.1128	0.1224	0.5645	**0.1088**	0.1325	0.7440
DrivAerML vol.	p	0.0517	0.0512	0.2885	**0.0509**	0.0573	0.4612
	$\mathbf{u}$	0.1112	0.1089	0.4470	**0.1083**	0.1242	0.6873
SHIFT-Wing surf.	p	0.00102	0.00104	0.0107	**0.00099**	0.00100	0.00849
	$\boldsymbol{\tau}_w$	0.0586	0.0586	0.0870	**0.0584**	0.0584	0.1125
SHIFT-Wing vol.	p	0.00225	0.00222	0.0333	**0.00215**	0.00225	0.0766
	$\mathbf{u}$	**0.1735**	0.1755	0.2843	0.1758	0.1802	0.4471
SHIFT-SUV surf.	p	**0.0271**	0.0280	0.1131	0.0271	0.0286	0.2043
	$\boldsymbol{\tau}_w$	**0.2607**	0.2618	0.3296	0.2608	0.2637	0.5728
SHIFT-SUV vol.	p	0.0298	0.0305	0.1330	**0.0279**	0.0328	0.2035
	$\mathbf{u}$	0.0582	0.0595	0.1537	**0.0550**	0.0647	0.3843

Table D.3: **Per-variable relative L^2 test error in physical units** (same protocol as Table D.2).

Dataset	Variable	baseline	no-token-attn	MLP-only	untied	frozen	slice-once
Taylor–Green	ρ	0.00034	0.00044	0.00035	0.00039	**0.00031**	0.00054
	$\mathbf{u}$	0.1850	**0.1826**	0.1906	0.1845	0.1888	0.4470
DrivAerNet++ surf.	p	**0.1509**	0.1563	0.2510	0.1553	0.1564	0.3479
	$\boldsymbol{\tau}_w$	0.2265	0.2316	0.3098	**0.2262**	0.2332	0.5272
DrivAerNet++ vol.	p	0.1621	**0.1462**	0.3087	0.1481	0.1474	0.3709
	$\mathbf{u}$	0.1337	0.1234	0.2088	0.1229	**0.1222**	0.2770
DrivAerML surf.	p	0.0614	0.0656	0.4053	**0.0590**	0.0711	0.3840
	$\boldsymbol{\tau}_w$	0.1204	0.1294	0.6067	**0.1163**	0.1385	0.7404
DrivAerML vol.	p	0.0729	**0.0725**	0.4265	0.0726	0.0837	0.6132
	$\mathbf{u}$	0.1195	**0.1177**	0.4749	0.1178	0.1343	0.6550
SHIFT-Wing surf.	p	0.00356	0.00361	0.0288	0.00352	**0.00330**	0.0191
	$\boldsymbol{\tau}_w$	0.0793	0.0791	0.1202	0.0790	**0.0788**	0.1453
SHIFT-Wing vol.	p	0.0101	**0.00939**	0.0782	0.00971	0.0102	0.1318
	$\mathbf{u}$	**0.2041**	0.2069	0.3239	0.2078	0.2124	0.4490
SHIFT-SUV surf.	p	0.0444	0.0461	0.1859	**0.0442**	0.0465	0.3757
	$\boldsymbol{\tau}_w$	**0.2933**	0.2946	0.3649	0.2937	0.2954	0.5754
SHIFT-SUV vol.	p	0.0537	0.0546	0.2181	**0.0501**	0.0595	0.3436
	$\mathbf{u}$	0.0729	0.0738	0.2049	**0.0685**	0.0801	0.4039

E LINEARNO IMPLEMENTATION AND LATENT WIDTH

For the UNTIED-LN variant in Figure 2(a), we use the released layer implementation of Hu et al. (2026) within the same model framework as Transolver. The lifting MLP, pre-layer normalization, residual connections, and output head are unchanged, so the comparison isolates the replacement of the attention layer. This experiment uses one seed on SHIFT-SUV surface.

Normalization. Both layers aggregate point features into tokens and map the tokens back to points, but construct their aggregation weights differently (Figure E.1(a)). Transolver applies a softmax over the G slices at each point and uses the resulting weights in both directions. Each aggregated token is then divided by the sum of its weights over points. LinearNO uses separate weights for slicing and deslicing. Its deslice weights q are normalized over latent channels, whereas its slice weights k are normalized over points. The released implementation explicitly stores both tensors, each of shape $[B, H, N, M]$, where M is the latent width per head. Their memory footprint therefore grows linearly with M. For Transolver, FlashSlice avoids materializing the corresponding point-to-slice weights (§5).

Latent width. LinearNO sets $M = \mathrm{key_ratio} \times d_{\mathrm{head}}$. With our head dimension $d_{\mathrm{head}} = 32$, the published default $\mathrm{key_ratio} = 4$ gives $M = 128$, compared with Transolver's $G = 32$ slices. We use $\mathrm{key_ratio} = 1$ in the main comparison to match these widths and run the default setting as an additional comparison.

Runtime and validation error. On SHIFT-SUV surface, using one node with four GH200 GPUs, $\mathrm{key_ratio} = 1$ runs at 1.88 iterations per second and completes 50k steps in $7\,\mathrm{h}\,25\,\mathrm{m}$. The default setting runs at 3.59 seconds per iteration, approximately 6.8 times slower. At that rate, 50k steps would take approximately 50 hours. These timings describe the implementations evaluated here.

The default-width run reaches $11{,}531$ steps within one 12-hour allocation. Both runs use the same 50k-step cosine schedule, allowing comparison at the same steps and learning rates. Across eleven shared evaluation points, the ratio of validation error at $\mathrm{key_ratio} = 4$ to that at $\mathrm{key_ratio} = 1$ averages 0.997, with a range of 0.95–1.02. Table E.1 lists selected checkpoints.

The two widths therefore give similar validation errors over the observed training interval, while the narrower setting is substantially faster. We use $\mathrm{key_ratio} = 1$ for the width-matched comparison; the partial default-width run does not establish how the two settings compare after 50k steps.

Table E.1: **LinearNO latent width on SHIFT-SUV surface.** Validation relative L^1 errors at selected shared checkpoints. Both runs use a 50k-step cosine schedule. The ratio is the error at $\mathrm{key_ratio} = 4$ divided by that at $\mathrm{key_ratio} = 1$.

step	1k	3k	5k	7k	9k	11k
$\mathrm{key_ratio} = 1$	0.4952	0.3005	0.2577	0.2210	0.2032	0.1927
$\mathrm{key_ratio} = 4$	0.4714	0.2973	0.2543	0.2232	0.2049	0.1938
ratio	0.95	0.99	0.99	1.01	1.01	1.01

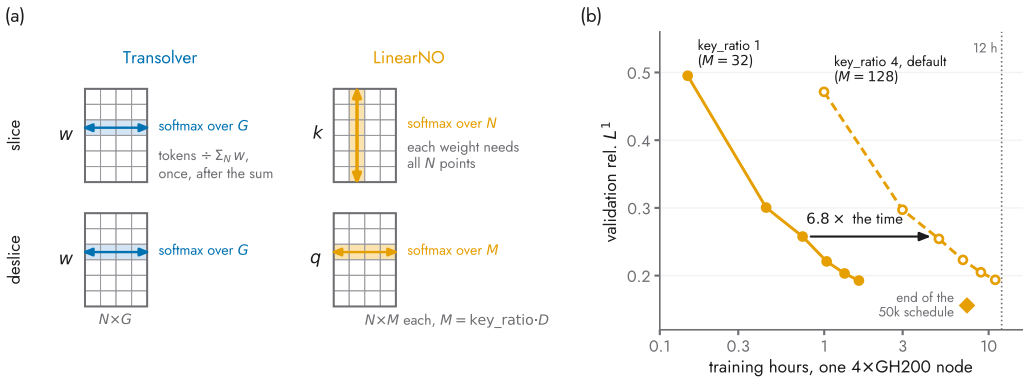

Figure E.1: **LinearNO normalization and latent width.** (a) Slice and deslice weights in Transolver and LinearNO,with their normalization axes. LinearNO uses latent width $M = \text{key_ratio} \times d_{\text{head}}$. (b) Validation errors on SHIFT-SUV surface at shared training steps, plotted against runtime estimated from measured iteration times on a four-GH200 node. The settings $\text{key_ratio} = 1$ and 4 give $M = 32$ and 128, respectively. The diamond marks the completion of 50k steps for $\text{key_ratio} = 1$.

F FLASHSLICE: IMPLEMENTATION AND EVALUATION

This appendix describes the kernels, numerical validation, and performance measurements supporting §5. Experiments use a node with four NVIDIA GH200 GPUs (95 GB usable HBM per GPU), PyTorch 2.5, and Triton 3.0 (Ansel et al., 2024; Tillet et al., 2019). Unless stated otherwise, measurements use one GPU with $C = 256$, $H = 8$, $G = 32$, $L = 8$, and batch size 1.

Each comparison is measured within the same job using fixed seeds and at least ten timed iterations after warm-up. We report median times; observed min–max spreads are below 2%. Point counts are rounded in the tables. The main sweep uses powers of two from 2^{18} to 2^{24}, with additional intermediate sizes near the memory limit.

F.1 KERNEL DESIGN

Single-tile kernels. For each head, the slice operation computes

$$w_{ig} = \text{softmax}_G\left(\frac{x_{\text{mid},i}W^\top + b}{\tau}\right)_g, \qquad z_g^{\text{num}} = \sum_i w_{ig} f_i, \qquad s_g = \sum_i w_{ig}.$$

The normalized tokens are $z_g = z_g^{\text{num}}/(s_g + \varepsilon)$. Deslicing recomputes the weights and maps the mixed tokens back to points.

When the slice axis fits within a tile, the softmax is computed locally for each point. The weights remain in registers rather than being written to HBM, and are recomputed during backward. Reductions use fp32 partial buffers followed by a deterministic tree reduction without atomics. The kernels read the projections in their native (N, H, D) layout, avoiding the eager implementation's layout copies.

Input projections remain separate GEMMs, and the token transformation remains in PyTorch. In a forward-only comparison over 36 tile configurations, fusing the projections achieved only 0.18–0.75 times the throughput of separate cuBLAS projections followed by the kernel.

The single-tile implementation supports power-of-two D and G in $[16, 128]$, with value width equal to D. Other supported shapes use the G-blocked kernels; $D > 256$ falls back to eager execution with a warning. Tile settings are tuned separately for each G. The forward and backward kernels are registered as PyTorch custom operators with shape inference and autograd support, allowing the tested model to compile as a single graph.

G-blocked kernels. For larger or irregular slice counts, the kernels process the slice axis in blocks of G_B. A statistics pass computes the maximum logit m_n and the shifted exponential sum l_n for

each point and head. Subsequent kernels reconstruct weights blockwise as

$$w_{ng} = \frac{\exp(a_{ng} - m_n)}{l_n},$$

where a_{ng} is the temperature-scaled logit.

Slice programs own a block of slots and reduce over points, producing partial token numerators and weight sums. Column normalization is applied after these partials have been combined. Deslice programs own a block of points and reduce over slots. When statistics are unavailable, deslicing uses an online softmax reduction and saves (m, l) for reuse. Operations with the same membership weights can share these statistics.

Backward uses separate point-owned and slot-owned kernels. The point-owned kernel makes two passes over slots to compute the softmax correction and input gradients. The slot-owned kernel accumulates projection, bias, and token gradients over points. This requires more logit recomputation than the single-tile implementation. The blocked family therefore uses its own tile settings, with G_B selected from 16–64 according to the dot-product tile size.

F.2 Numerical validation

Precision modes and acceptance criteria. Table F.1 lists the supported dot-product modes. All modes accumulate in fp32. The default `ieee` mode uses fp32 FMA operations; the other modes trade numerical accuracy for tensor-core throughput.

We compare outputs and parameter gradients against an fp64 eager reference, using the corresponding eager error to set the acceptance threshold. For the default fp32 mode, the threshold is 1.25 times the eager fp32 error. It is relaxed to 3 times when the eager reference error exceeds 10^{-4}. The bf16-dot modes use a threshold of 2 times the eager-autocast error, and the TF32 modes use 10 times the eager fp32 error. For the blocked kernels, the operator-level temperature gradient uses a 2-times threshold; the layer-level tests at the paper configurations retain the 1.25-times threshold. Repeated fused runs must also agree bitwise in these tests.

Table F.1: **Dot-precision levels of the kernels.** Accumulation is fp32 throughout; the gate is the admissible error against an fp64 reference, relative to the corresponding eager computation. `ieee` is Triton's FMA path and needs no tensor cores.

level	logits dot	value dots	inputs	gate
`ieee` (default)	fp32 FMA	fp32 FMA	fp32 or bf16	1.25× eager fp32
`tf32`	fp32 FMA	tf32 tensor cores	fp32 or bf16	10×
`bf16v`	fp32 FMA	bf16 tensor cores	bf16	2× eager bf16 autocast
`bf16`	bf16 tensor cores	bf16 tensor cores	bf16	2× eager bf16 autocast
`tf32x3`	three tf32 products	three tf32 products	fp32	10×; measured 3–7× eager fp32

Temperature gradients. The temperature gradient is particularly sensitive to rounding in the softmax backward:

$$\frac{\partial L}{\partial \tau} = -\sum_{n,g} \frac{\partial L}{\partial a_{ng}} \frac{a_{ng}}{\tau}.$$

In exact arithmetic, $\sum_g \partial L / \partial a_{ng} = 0$ for each point. Rounding errors in this cancellation can affect the temperature gradient.

We therefore save (m, l) directly, construct the softmax correction and logit gradients from the same rounded products, and use correctly rounded reciprocals for $1/l_n$ and $1/\tau$. In our tests, approximate division increased the temperature gradient error to 8–14 times the eager error.

Layer and optimizer tests. With TF32 disabled, the tested single-tile configurations and ablation variants have fused output and parameter-gradient errors at or below those of eager fp32, typically 5×10^{-7} compared with 1–4 $\times 10^{-6}$. Starting from identical parameters, a 60-step optimizer comparison gives a relative loss difference of at most 2×10^{-6}.

Table F.2: **Complete single-GPU sweep behind Table 2** (one job, eager PyTorch, no compilation, no checkpointing). Each cell is median time in ms with peak memory in GB; "OOM" = does not fit in 95 GB. "ours" is the exact-fp32-dot tier in the fp32 block and the bf16 tensor-core tier under autocast.

	inference: ms (GB)		training step: ms (GB)	
N	eager	ours	eager	ours
fp32				
262k	58.4 (2.6)	35.3 (2.1)	171 (28.0)	126 (24.0)
1M	229 (10.1)	138 (8.1)	OOM	OOM
2M	475 (20.2)	280 (16.2)	OOM	OOM
4M	980 (40.3)	566 (32.3)	OOM	OOM
8.4M	1944 (80.5)	1160 (64.5)	OOM	OOM
12.6M	OOM	OOM	OOM	OOM
bf16 autocast				
262k	45.6 (2.0)	29.6 (1.5)	143 (20.4)	93.1 (15.4)
1M	174 (7.6)	113 (5.6)	585 (81.3)	345 (61.3)
2M	349 (15.2)	226 (11.2)	OOM	OOM
4M	700 (30.3)	456 (22.3)	OOM	OOM
8.4M	1420 (60.5)	932 (44.5)	OOM	OOM
12.6M	OOM	1411 (66.7)	OOM	OOM
16.8M	OOM	OOM	OOM	OOM

Under bf16 autocast, the output errors are 2.5×10^{-3} for fused and 2.6×10^{-3} for eager, with gradient errors within 5% of each other. Table 2 uses fp32 dots for the fp32 measurements and bf16 tensor-core dots for the autocast measurements.

F.3 Single-GPU performance

Point-count sweep. Table F.2 reports inference and training measurements against uncompiled eager PyTorch, without activation checkpointing. The fused implementation reduces time and peak memory by avoiding materialized slice weights and layout copies.

Additional measurements near the memory limit increase the largest tested inference size from 8.4M to 10.5M points in fp32 and from 10.5M to 14.7M under bf16 autocast. For training, the corresponding sizes increase from 786k to 917k points in fp32 and from 1M to 1.57M in bf16.

Precision affects the speedup. Under autocast, retaining fp32 dots gives 1.20–1.31 times the eager training throughput at 262k and 1M points, compared with 1.53–1.70 times using bf16 dots. At 262k points with fp32 inputs, the optional TF32 mode gives a 1.45-times speedup, compared with approximately 1.35 times for fp32 dots.

Slice count and depth. Table F.3 varies G and L at $N = 262$k points. From $G = 16$ to 128, fused bf16 peak memory remains approximately 15.4 GB, while eager memory increases from 17.9 to 36.9 GB. Over the same range, the training speedup increases from 1.31 to 2.18 times.

Peak memory grows approximately linearly with depth for both implementations, with a smaller slope for the fused model. At $G = 32$, the tested configurations fit up to 32 layers in fp32 and 48 in bf16, compared with 16 and 32 for eager. Figure F.1 shows the corresponding step times.

Compiled baselines. Compilation substantially improves the baseline, so Table F.4 also compares against `torch.compile`. The compiled fused configuration with fast dots is 1.24–1.26 times faster per training step than the compiled baseline in the tested settings. These are configuration-level comparisons: the fused rows use TF32 dots for fp32 inputs and bf16 tensor-core dots for bf16 inputs. The corresponding inference speedups are 1.4–1.5.

Table F.3: **Structural scaling**: peak memory of a training step at N=262k as the two configuration knobs move, each block measured in one job. "OOM" = does not fit in 95 GB. Fused peak memory changes little over the tested range of G; both grow linearly in depth, with different slopes.

	fp32, peak GB		bf16, peak GB		bf16 step
	eager	ours	eager	ours	speedup
slice count G (depth L=8)					
G=16	26.0	24.0	17.9	**15.4**	1.31×
G=32	28.0	24.0	20.4	**15.4**	1.54×
G=64	32.0	24.0	25.4	**15.4**	1.83×
G=128	41.7	24.0	36.9	**15.4**	2.18×
depth L (G=32)					
L=4	15.0	13.0	10.9	8.4	1.52×
L=8	28.0	24.0	20.4	15.4	1.51×
L=16	54.1	46.1	39.5	29.5	1.42×
L=32	OOM	90.2	77.6	57.6	1.50×
L=48	OOM	OOM	OOM	85.7	—

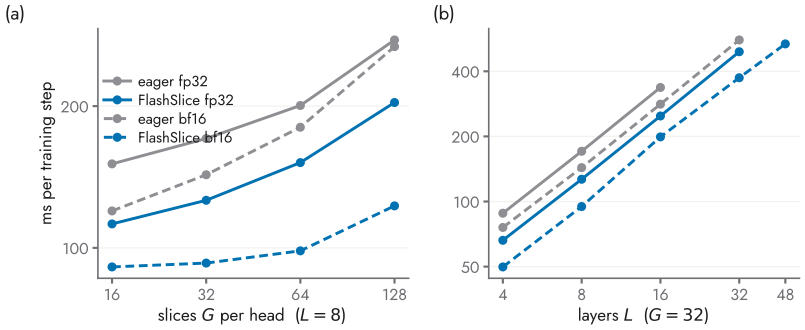

Figure F.1: **Step time of the sweeps of Figure 3(b, c).** Median time of a training step against (a) slice count and (b) depth (N=262k, eager PyTorch against the fused kernels, no compilation or checkpointing; log axes). The speedup grows with G, from 1.31× at G=16 to 2.18× at G=128 under bf16, and is a constant factor in depth.

Activation checkpointing. Table F.5 compares both implementations with non-reentrant activation checkpointing and compilation. The fused configurations are 1.31–1.34 times faster in bf16 and 1.39–1.42 times faster with fp32 inputs, with approximately 9% lower memory use.

In these tests, both implementations train up to 4M points per GPU in bf16. Thus, with checkpointing and compilation enabled, fusion reduces runtime and memory but does not increase the largest tested training size.

Large slice counts. Table F.6 evaluates one attention layer using the G-blocked kernels. At N = 262k, fused bf16 training memory remains about 2 GB from G = 48 to 1024. At G = 512 and N = 1M, eager runs out of memory while the fused layer uses 8.1 GB.

Runtime depends strongly on dot precision. For the reported large-G training configurations, bf16 tensor-core modes give speedups of 3.98–6.45 times, whereas fp32 FMA modes can be slower than eager. The `tf32x3` mode gives speedups of up to 1.92 times with fp32 inputs.

The kernels are not faster for every shape. At D = 256, the tested blocked configuration reaches 0.79 times eager throughput with 4% lower memory. At the paper configuration, G = 32, the single-tile implementation is faster than the blocked implementation, which requires additional statistics and recomputation.

Table F.4: **Single-GPU training step (ms) against compiled baselines**, medians of ≥ 10 iterations; each precision block is measured in one job, and anchors reproduce across jobs within 1%. "fast dot" = TF32 (fp32 inputs) or bf16 tensor-core (bf16 inputs) tier.

	fp32, 262k	bf16, 262k	bf16, 1M
eager	170.5	142.9	586
eager + compile	134.8	79.5	300
fused (exact fp32 dots)	128.4	118.9	448
fused + compile + fast dot	**106.6**	**63.9**	**241**
vs. strongest baseline	1.26×	1.24×	1.25×

Table F.5: **Checkpointed training (ms), one job.** `torch.compile` traverses non-reentrant activation checkpointing in this stack, halving the checkpointed baseline; the baseline uses checkpointing and compilation.

	bf16, 1M	bf16, 4M	fp32, 1M	fp32, 2M
ckpt	740	2999	921	1874
ckpt + compile	365	1479	696	1432
fused ckpt + compile + fast dot	**278**	**1106**	**500**	**1006**
vs. ckpt + compile	1.31×	1.34×	1.39×	1.42×

F.4 Multi-GPU execution and compilation

Point sharding. We shard one sample's points across four GPUs. In the forward pass, each GPU computes local token numerators and weight sums, which are all-reduced before normalization. At the paper configuration, this reduction contains approximately 33 KB per layer. Sharded and single-GPU forward results agree within fp32 reduction-order differences.

Table F.7 reports training at 16.8M points with checkpointing and bf16. The fused compiled configuration takes 1152 ms per step, compared with 3043 ms for the compiled eager pipeline, giving approximately 2.6 times the throughput. Peak memory decreases from 86.6 to 78.6 GB per GPU. Step timings include parameter-gradient synchronization.

Compilation has little effect on the eager sharded pipeline in this software stack. The custom-operator implementation composes with compilation and checkpointing in the tested configuration. The measured speedup therefore reflects the full pipeline, including its compilation behavior.

Implementation settings. All compiled measurements use static per-shape graphs. For Triton 3.0, the TF32 path uses one pipeline stage, and tensor-core configurations with $G = 16$ use four warps to avoid compilation failures encountered during tuning. These settings and the kernel tile tables are included in the implementation.

Table F.6: **One layer at large slice counts, eager against the G-blocked kernels** (H=8, D=32, one GH200, medians of ten; training step unless marked). "inputs / dots" is the activation dtype and the dot-precision level of Table F.1; "OOM" = does not fit in 95 GB.

inputs / dots	G	N	eager (ms)	ours (ms)	speedup	eager (GB)	ours (GB)	saved
bf16 / bf16	256	262k	47.5	11.9	3.98×	13.7	2.0	85%
bf16 / bf16	256	1M	293.7	45.5	6.45×	54.5	8.1	85%
bf16 / bf16	512	262k	90.6	20.0	4.53×	26.7	2.0	92%
bf16 / bf16	512	1M	OOM	80.3	—	—	8.1	—
bf16 / bf16	1024	262k	180.2	37.6	4.79×	52.7	2.1	96%
bf16 / bf16, inference	256	1M	39.4	13.3	2.96×	17.0	2.1	88%
bf16 / bf16, inference	256	4.2M	157.3	53.2	2.95×	68.0	8.3	88%
bf16 / bf16	48	262k	13.0	6.1	2.12×	3.1	2.0	35%
fp32 / tf32x3	256	262k	45.4	37.4	1.21×	14.8	2.0	86%
fp32 / tf32x3	256	1M	283.6	147.7	1.92×	59.0	8.1	86%
fp32 / tf32	256	262k	45.1	40.5	1.11×	14.8	2.0	86%
fp32 / ieee	256	262k	45.1	53.2	0.85×	14.8	2.0	86%
fp32 / ieee	256	1M	284.2	209.6	1.36×	59.0	8.1	86%
bf16 / tf32	256	262k	47.5	40.0	1.19×	13.7	2.0	85%
bf16 / ieee	256	262k	47.4	64.0	0.74×	13.7	2.0	85%
bf16 / ieee	256	1M	294.3	253.4	1.16×	54.5	8.1	85%

Table F.7: **One node, 4×GH200, 16.8M points, bf16**, all configurations in one job; DDP-equivalent step including the gradient all-reduce. Compilation gives little improvement on the eager sharded pipeline; the operator form composes.

	step (ms)	Mpts/s	GB/GPU
ckpt + sharding	3048	5.5	86.5
ckpt + sharding + compile	3043	5.5	86.6
fused ckpt + compile + bf16 dot	**1152**	**14.6**	78.6