
LTS: A Lightweight Persistent-Latent Reformulation of Transolver for Transient Impact Simulation

Chia Feng Liao*
National Sun Yat-sen
University

Lin Chou*
National Sun Yat-sen
University

Shih-Chi Wang
National Chung Hsing
University

Abstract

Physics surrogates accelerate engineering analysis, but Transolver [1]-based models repeatedly route between full-resolution meshes and latent tokens in every Physics-Attention block, incurring substantial cost. We adopt single-pass latent projection within the Transolver backbone and obtain LTS, a Lightweight Transolver Surrogate that constructs physics-aware latent tokens once, processes them persistently through all intermediate blocks, reconstructs node features only at the output, and applies lightweight component and global adaptation after decoding. On DropTest, a fixed-topology industrial phone-drop benchmark, LTS reduces train-step latency by **84.6%**, peak allocated training memory by **82.5%**, and inference latency across all 99 independently queried target states by **86.9%** relative to a depth-matched Transolver. It also lowers stress relative L_2 error from 3.54% to 2.85%, while displacement relative L_2 error increases from 0.47% to 0.64%. Routing-frequency experiments show that accuracy gains from re-routing are non-monotonic, whereas computational cost increases substantially. These results characterize the accuracy–efficiency trade-off of persistent latent processing in transient impact simulation.

1 Introduction

High-fidelity computer-aided engineering (CAE) simulations often require large-scale meshes and expensive iterative solvers, making repeated evaluation costly in industrial design and optimization [2, 3]. Deep learning surrogates, including neural operators [4], can substantially reduce this cost, but industrial CAE problems are commonly discretized on large unstructured meshes, where efficient global modeling remains challenging.

Transformers provide global interaction modeling, but standard self-attention over N mesh nodes incurs $O(N^2)$ complexity. Transolver [1] addresses this through Physics-Attention: each block softly maps the N nodes to $M \ll N$ learned physics-aware slices, performs self-attention among the resulting tokens, and maps them back to the mesh.

However, this slice–attend–deslice cycle is repeated in every Physics-Attention block. Across L blocks, Transolver repeatedly recomputes the $N \times M$ mesh–latent routing and reconstructs full-resolution node features, coupling mesh-level computation and activations to network depth despite performing attention only on M tokens.

To address this inefficiency, we reformulate the Transolver backbone with persistent latent routing and obtain LTS.² Let K denote the number of mesh–latent routing cycles; LTS sets $K = 1$ regardless

*Equal contribution.

²Code and experiment artifacts are available at <https://github.com/choulin/LTS>.

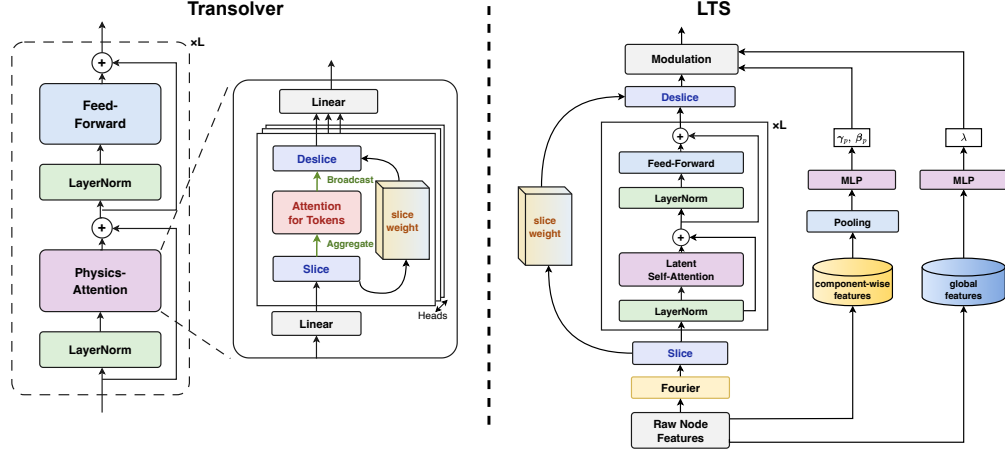


Figure 1: Comparison between Transolver and LTS. Transolver repeatedly performs slicing and deslicing in every block, whereas LTS keeps intermediate computation in the latent space and optionally uses component adaptation to modulate the reconstructed node features.

of network depth, performing physics-aware slicing once at the input, all intermediate self-attention and FFN computation in the M -token latent space, and deslicing once at the output.

This efficiency comes with a representational trade-off. In Transolver, re-slicing updated node features at every block allows routing assignments to be progressively revised, whereas LTS must rely on a single initial assignment throughout the forward pass. Consequently, improving the information available to this initial routing becomes particularly important.

The main contributions of this work are summarized as follows.

- Within our fixed-parameter shared-router variants, increasing the routing count substantially raises computational cost while yielding non-monotonic mean accuracy.
- We adapt single-pass latent projection to the Transolver backbone, obtaining LTS, which removes repeated mesh–latent routing, keeps intermediate computation in latent space, and uses lightweight component adaptation to preserve region-specific information.
- The full LTS system, including Fourier features and output-side adaptation, achieves lower mean stress error than Transolver, although the persistent-latent backbone alone is less accurate.

2 Related Work

Transolver and persistent latent processing. Transolver [1] maps mesh nodes to learned physics-aware slices, processes the resulting tokens, and maps them back to the mesh in every block. Single-pass latent processing has previously been explored by UPT [5] and LNO [6], with LNO also reporting memory and training-speed improvements over Transolver. LinearNO [7] reformulates each Physics-Attention block as linear attention while retaining point routing across layers; LTS instead reduces the number of mesh–latent routing cycles across depth.

Building on this principle, LTS retains Transolver’s learned slicing mechanism but isolates routing frequency: we vary the number of mesh–latent routing cycles K while keeping the projection mechanism fixed. LTS corresponds to $K = 1$, enabling a controlled study of the accuracy–efficiency effect of repeated physics-aware re-slicing. Fourier features enrich the one-time input routing, while output-side component adaptation and displacement scaling provide regional context and trajectory-level amplitude adaptation. Further comparisons are provided in Appendix A.

Table 1: Main results on DropTest under a unified evaluation and single-H200 profiling protocol.

Model	Accuracy		Efficiency			
	Disp. Rel. $L_2 \downarrow$ (%)	Stress Rel. $L_2 \downarrow$ (%)	Train-step Latency $\downarrow$ (ms/step)	Peak Train Alloc. Mem. $\downarrow$ (GiB)	Inference Latency $\downarrow$ (ms/traj.)	Params. (M)
Transolver	<u>0.47 ± 0.02</u>	3.54 ± 0.17	193.12 ± 0.08	28.15	4320.38 ± 1.11	8.97
GeoTransolver	0.59 ± 0.24	<u>1.91 ± 0.14</u>	362.14 ± 0.18	45.10	10453.15 ± 3.93	27.58
GeoTS-FLARE	0.53 ± 0.09	1.81 ± 0.08	276.43 ± 0.31	29.83	9334.00 ± 29.64	26.33
Transolver++	0.51 ± 0.06	4.22 ± 0.14	<u>26.91 ± 0.10</u>	2.03	793.78 ± 0.96	1.74
Transolver-3	0.38 ± 0.01	3.58 ± 0.09	177.19 ± 0.17	<u>3.69</u>	4463.01 ± 11.21	<u>7.60</u>
LTS (Base)	0.91 ± 0.07	7.81 ± 0.27	24.06 ± 0.16	4.84	455.60 ± 4.23	8.02
LTS (Full)	0.64 ± 0.07	2.85 ± 0.69	29.76 ± 0.21	4.93	<u>567.18 ± 1.72</u>	8.34
vs. Transolver	$\uparrow 0.17$ pp	$\downarrow 0.69$ pp	$\downarrow 84.6\%$	$\downarrow 82.5\%$	$\downarrow 86.9\%$	$\downarrow 7.0\%$

Notes. Accuracy and 99-query latency are mean $\pm$ sample s.d. over seeds $\{7, 17, 27\}$. Seed-7 H200 training latency uses 50 timed steps after 20 warm-up steps, peak memory is the maximum allocated during profiling. Base disables CA, Fourier, and D-Scale; Full enables all three. Only CA uses component labels. Detailed settings are in Appendix B.1.

3 Methodology

We consider N mesh points with coordinates $\mathbf{G}$, node features $\mathbf{X}_{\text{in}}$, global impact conditions $\mathbf{g}$, and model query τ_q , and predict nodal displacement $\hat{\mathbf{D}} \in \mathbb{R}^{N \times 3}$ and von Mises stress $\hat{\mathbf{S}} \in \mathbb{R}^{N \times 1}$. LTS performs physics-aware tokenization once, processes all intermediate blocks in the latent space, and reconstructs node features only at the output, avoiding the repeated mesh-latent routing of Transolver. Since this fixed routing cannot revise initial slice assignments, we augment LTS with Fourier feature encoding [8], component adaptation, and a trajectory-conditioned displacement scale (D-Scale), all without introducing additional routing cycles.

Fourier feature encoding. Let $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ denote the normalized spatial coordinates. For a normalized scalar s with bandwidth B , we define

$$\Phi(s; B) = [\sin(\pi 2^b s), \cos(\pi 2^b s)]_{b=0}^{B-1}. \quad (1)$$

Each trajectory is sampled at time steps $r \in \{0, \dots, 99\}$, where $r = 0$ is the reference state and $r = 1, \dots, 99$ are prediction targets. The dataloader reindexes the targets as $q = r - 1 \in \{0, \dots, 98\}$ and supplies the model with

$$\tau_q = \frac{q}{99} = \frac{r-1}{99}. \quad (2)$$

Thus, τ_q is a normalized query-slot index rather than physical simulation time. We use 10 frequency bands for each spatial coordinate and 8 for τ_q . The resulting encodings $\Phi(\tilde{x}; 10)$, $\Phi(\tilde{y}; 10)$, $\Phi(\tilde{z}; 10)$, and $\Phi(\tau_q; 8)$ are concatenated, projected by a small MLP, and added to the point features to form the initial node feature matrix $\mathbf{X}_0$ used for one-time slicing.

Persistent-latent LTS. Following Transolver, the assignment matrix, slice tokens, latent processing, and final reconstruction are

$$\begin{aligned} \mathbf{W} &= \text{Softmax}_M(\text{Project}(\mathbf{X}_0)) \in \mathbb{R}^{N \times M}, \\ \mathbf{z}_j &= \frac{\sum_{i=1}^N w_{ij} \mathbf{x}_{0,i}}{\sum_{i=1}^N w_{ij}}, \quad \mathbf{Z}_L = \text{Block}_L \circ \dots \circ \text{Block}_1(\mathbf{Z}_0), \\ \mathbf{H} &= \mathbf{W} \mathbf{Z}_L, \end{aligned} \quad (3)$$

where $\mathbf{X}_0, \mathbf{H} \in \mathbb{R}^{N \times d}$ and $\mathbf{Z}_0, \mathbf{Z}_L \in \mathbb{R}^{M \times d}$. The softmax is taken over the M slices independently for each node. The assignment matrix $\mathbf{W}$ is cached.

Output-side adaptation. After deslicing, component identity is again well defined. For DropTest, node i has component label r_i from PART_ID. We pool the pre-slicing features of component p and apply a FiLM-style affine modulation [9]:

$$\begin{aligned} \mathbf{c}_p &= \text{MLP}([\text{mean}_{i \in S_p} \mathbf{x}_{0,i}; \max_{i \in S_p} \mathbf{x}_{0,i}]), \\ [\gamma_p; \beta_p] &= \text{Linear}(\mathbf{c}_p), \quad \mathbf{h}'_i = (1 + \gamma_{r_i}) \odot \mathbf{h}_i + \beta_{r_i}. \end{aligned} \quad (4)$$

Table 2: Effect of the mesh-latent routing count K on DropTest.

K	Disp. Rel. $L_2 \downarrow$ (%)	Stress Rel. $L_2 \downarrow$ (%)	99-query Inference Latency $\downarrow$ (ms/traj.)
1	0.64 ± 0.07	2.85 ± 0.69	567.18 ± 1.72
2	0.62 ± 0.02	<u>2.50 ± 0.24</u>	<u>856.14 ± 10.09</u>
3	0.69 ± 0.07	2.70 ± 0.56	1148.69 ± 21.74
4	0.65 ± 0.01	2.74 ± 0.31	1421.19 ± 1.44
6	0.67 ± 0.12	2.40 ± 0.05	1985.36 ± 19.86
12	<u>0.63 ± 0.06</u>	2.80 ± 0.17	3689.43 ± 30.29

Notes. Full settings and twelve latent blocks remain fixed across K . Blocks are split near-uniformly across cycles (Appendix B), each deslicing and recomputing soft assignments with shared routing parameters. $K = 12$ matches Transolver’s routing frequency, not its parameterization or auxiliary modules.

The modulation is applied once before the shared output head, with its final projection zero-initialized to preserve an identity mapping at initialization.

We additionally apply a trajectory-conditioned displacement scale (D-Scale) after the output head. The head first produces the pre-scaling displacement and stress predictions, and the final outputs are

$$\begin{aligned} [\tilde{\mathbf{D}}, \tilde{\mathbf{S}}] &= \text{Head}(\mathbf{H}'), \\ \lambda(\mathbf{g}, \tau_q) &= \mathbf{1} + \text{MLP}([\mathbf{g}; \tau_q]) \in \mathbb{R}^3, \\ \hat{\mathbf{D}} &= \tilde{\mathbf{D}} \odot \lambda(\mathbf{g}, \tau_q), \quad \hat{\mathbf{S}} = \tilde{\mathbf{S}}. \end{aligned} \tag{5}$$

Here, $\lambda(\mathbf{g}, \tau_q)$ is the D-Scale factor and is broadcast across all N nodes. D-Scale therefore adjusts the trajectory-level amplitude of the three displacement channels while leaving the stress prediction unchanged.

4 Experiments

Dataset, baselines, and evaluation. DropTest-192 comprises 192 trajectories on a fixed 76,065-node mesh in a $2^5 \times 6$ full-factorial design. We use a material-grouped 132/30/30 train/validation/test split and predict nodal displacement and von Mises stress at 99 target states from reference geometry, query τ_q , and eight material/impact parameters. We compare LTS with Transolver, GeoTransolver [10], GeoTS-FLARE [11], Transolver++ [12], and Transolver-3 [13]. As LTS is a reformulation of Transolver, our evaluation focuses on its accuracy–efficiency trade-off relative to Transolver and related architectures. A broader comparison with other neural operator architectures, including LNO[6], is left for future work. External baselines retain their released backbone configurations, with only task-specific interfaces adapted to DropTest. All models share the same training and evaluation protocol, and LTS uses the official Transolver volume configuration as its reference scale; full settings are in Appendix B. Each relative L_2 norm accumulates all 99 targets within a trajectory and is then macro-averaged over 30 test trajectories; values are mean $\pm$ sample s.d. over three seeds.

Main results. Table 1 summarizes accuracy and systems efficiency. Relative to Transolver, Full reduces train-step latency, peak training memory, and 99-query inference latency by 84.6%, 82.5%, and 86.9%, respectively, with 7.0% fewer parameters. Stress error drops from 3.54% to 2.85%, whereas displacement error rises from 0.47% to 0.64%. The no-label Fourier + D-Scale variant obtains 2.85% stress error, essentially matching Full.

Per-state absolute error and efficiency. Figure 2 complements trajectory-level relative error with physical RMSE. All seven models have higher RMSE at $r = 99$ than at $r = 1$, despite lower per-state relative L_2 : GT RMS grows $96.1 \times / 122.5 \times$ for displacement/stress. Relative error decline therefore does not imply lower absolute error for these independent queries. Panel (c) shows Full’s low-latency, competitive-stress-accuracy regime.

Ablation study. Increasing K does not consistently improve accuracy (Table 2). Compared with $K = 1$, $K = 2$ reduces both mean errors (stress by 12.3%) but increases latency by 51% (567 to 856 ms/trajectory). At $K = 12$, latency rises to $6.5 \times$ that of $K = 1$ without consistent accuracy gains, supporting $K = 1$ as an efficiency-oriented default.

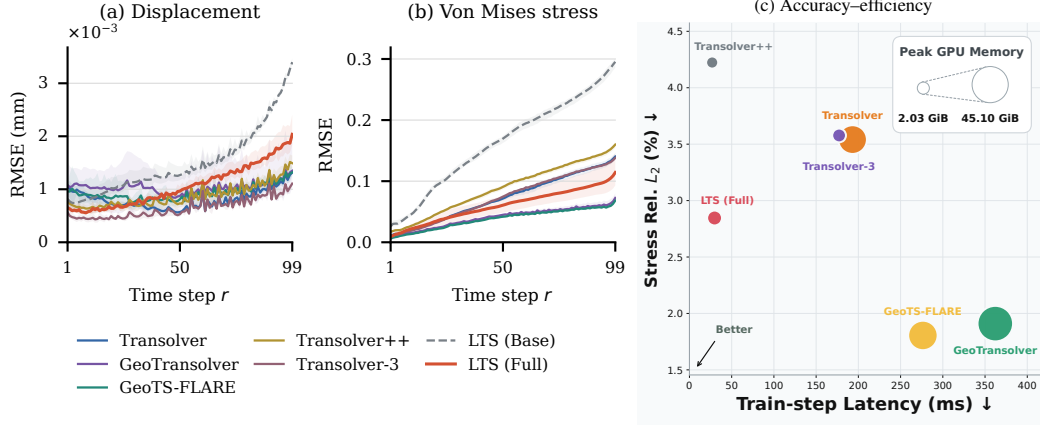


Figure 2: DropTest error and accuracy-efficiency. (a,b) Per-trajectory physical RMSE for XYZ displacement and scalar stress, averaged over 30 cases and seeds 7, 17, 27. Bands show seed ranges. (c) Trajectory-macro stress relative L_2 versus training-step latency for five baselines and Full. Bubble area denotes peak allocated training memory.

Table 3: LTS component ablation on DropTest.

	Configuration							
	1	2	3	4	5	6	7	8
CA		✓			✓	✓		✓
Fourier			✓		✓		✓	✓
D-Scale				✓		✓	✓	✓
	0.91	0.65	0.94	0.99	0.64	0.80	0.81	0.64
Disp. L_2 (%) ↓	± 0.07	± 0.05	± 0.10	± 0.03	± 0.04	± 0.05	± 0.03	± 0.07
	<i>baseline</i>	-28.2%	+3.4%	+9.1%	-29.3%	-11.7%	-11.4%	-29.7%
	7.81	5.26	7.41	6.31	4.10	5.03	2.85	2.85
Stress L_2 (%) ↓	± 0.27	± 0.44	± 1.10	± 0.34	± 0.18	± 0.67	± 0.74	± 0.69
	<i>baseline</i>	-32.6%	-5.2%	-19.3%	-47.5%	-35.7%	-63.6%	-63.6%

Notes. Each cell reports the mean, standard deviation over seeds {7, 17, 27}, and relative change from bare LTS (config. 1). Negative changes indicate improvement.

Table 3 evaluates all eight combinations of CA, Fourier, and D-Scale (Eq. (5)) under the common protocol. Configurations 1 and 8 reuse the Base and Full runs in Table 1; Full also corresponds to $K = 1$ in Table 2. CA provides the largest individual improvement, reducing displacement/stress errors by 28.2%/32.6%. Fourier and D-Scale alone improve stress but worsen displacement, whereas their combination reduces stress error to 2.85%. Adding CA to this pair further reduces displacement error from 0.81% to 0.64%. The gains depend on the component combination: adding D-Scale to CA alone increases displacement error from 0.65% to 0.80%. At the reported precision, Full combines the displacement accuracy of CA + Fourier with the stress accuracy of Fourier + D-Scale, supporting its use when both targets are considered. Separate DrivAerML results are in Appendix C.

5 Conclusion and Limitations

LTS exposes an efficiency-accuracy trade-off on fixed-topology DropTest by reducing repeated mesh-latent routing. Among the tested routing counts, $K = 2$ yields the lowest mean displacement error and improves both errors over $K = 1$, at higher latency. However, we have not established a criterion for selecting K across tasks. Our evaluation covers one topology, and CA requires part labels. Future work will study adaptive re-slicing, label-free grouping, and cross-topology evaluation.

6 Acknowledgement

We thank the National Center for High-Performance Computing, National Institutes of Applied Research (NIAR), Taiwan, for providing the necessary computational resources and platform services for this research.

References

- [1] H. Wu, H. Luo, H. Wang, J. Wang, and M. Long, “Transolver: A fast transformer solver for PDEs on general geometries,” in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024, pp. 53 681–53 705.
- [2] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.
- [3] N. Umetani and B. Bickel, “Learning three-dimensional flow for interactive aerodynamic design,” *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.
- [4] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Neural Operator: Learning maps between function spaces with applications to PDEs,” *Journal of Machine Learning Research*, vol. 24, no. 89, pp. 1–97, 2023.
- [5] B. Alkin, A. Fürst, S. L. Schmid, L. Gruber, M. Holzleitner, and J. Brandstetter, “Universal physics transformers: A framework for efficiently scaling neural operators,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024.
- [6] T. Wang and C. Wang, “Latent neural operator for solving forward and inverse PDE problems,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024.
- [7] W. Hu, S. Liu, P. Qiao, Z. Sun, and Y. Dou, “Transolver is a linear transformer: Revisiting Physics-Attention through the lens of linear attention,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 40, 2026, arXiv:2511.06294.
- [8] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, “Fourier features let networks learn high frequency functions in low dimensional domains,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.
- [9] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville, “FiLM: Visual reasoning with a general conditioning layer,” *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 32, no. 1, 2018.
- [10] C. Adams, R. Ranade, R. Cherukuri, and S. Choudhry, “Geotransolver: Learning physics on irregular domains using multi-scale geometry aware physics attention transformer,” *arXiv preprint arXiv:2512.20399*, 2025.
- [11] D. Akhare, M. A. Nabian, C. Adams, S. Chavare, and S. Choudhry, “High-fidelity industrial crash dynamics prediction via geometry-aware operator learning with memory-efficient low-rank attention,” *arXiv preprint arXiv:2605.27758*, 2026.
- [12] H. Luo, H. Wu, H. Zhou, L. Xing, Y. Di, J. Wang, and M. Long, “Transolver++: An accurate neural solver for PDEs on million-scale geometries,” in *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 267. PMLR, 2025, pp. 41 432–41 449.
- [13] H. Zhou, H. Wu, H. Shangguan, Y. Ma, H. Weng, J. Wang, and M. Long, “Transolver-3: Scaling up transformer solvers to industrial-scale geometries,” in *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026, arXiv:2602.04940.
- [14] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators,” *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021.
- [15] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier Neural Operator for parametric partial differential equations,” in *International Conference on Learning Representations (ICLR)*, 2021.
- [16] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, “Learning mesh-based simulation with graph networks,” in *International Conference on Learning Representations (ICLR)*, 2021.

- [17] Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu, “GNOT: A general neural operator transformer for operator learning,” in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 12 556–12 569.
- [18] V. Puri, A. Joglekar, S. D. G. Bandreddi, K. Ferguson, Y. hsuan Chen, Y. J. Zhang, and L. B. Kara, “FLARE: Fast low-rank attention routing engine,” *arXiv preprint arXiv:2508.12594*, 2025.
- [19] J. Lee, Y. Lee, J. Kim, A. R. Kosiosek, S. Choi, and Y. W. Teh, “Set transformer: A framework for attention-based permutation-invariant neural networks,” in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 3744–3753.
- [20] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, “Perceiver: General perception with iterative attention,” in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 4651–4664.
- [21] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep hierarchical feature learning on point sets in a metric space,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5099–5108.
- [22] H. Zhao, L. Jiang, J. Jia, P. H. S. Torr, and V. Koltun, “Point transformer,” in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 16 259–16 268.
- [23] Z. Li, N. B. Kovachki, C. Choy, B. Li, J. Kossaifi, S. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, and A. Anandkumar, “Geometry-informed neural operator for large-scale 3d PDEs,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.
- [24] S. Wen, A. Kumbhat, L. Lingsch, S. Mousavi, Y. Zhao, P. Chandrashekar, and S. 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.
- [25] X. Guo, W. Li, and F. Iorio, “Convolutional neural networks for steady flow approximation,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 481–490.
- [26] Y. Zhu and N. Zabaras, “Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification,” *Journal of Computational Physics*, vol. 366, pp. 415–447, 2018.
- [27] M. Elrefaie, D. Shu, M. Klenk, and F. Ahmed, “CarCrashNet: A large-scale dataset and hierarchical neural solver for data-driven structural crash simulation,” *arXiv preprint arXiv:2605.07098*, 2026.
- [28] N. Ashton, C. Mockett, M. Fuchs, L. Fliessbach, H. Hetmann, T. Knacke, N. Schonwald, V. Skaperdas, G. Fotiadis, A. Walle, B. Hupertz, and D. Maddix, “DrivAerML: High-fidelity computational fluid dynamics dataset for road-car external aerodynamics,” *arXiv preprint arXiv:2408.11969*, 2024. [Online]. Available: <https://arxiv.org/abs/2408.11969>

A Extended Context

This appendix provides broader literature context and a compact dataset summary omitted from the five-page main text.

A.1 Point-to-Token Projection

Neural operators are increasingly used for PDE surrogate modeling on large-scale meshes [2, 4], following convolutional encoder–decoders on regular grids [25, 26], operator learning [14, 15], and mesh-based graph solvers [16]. Attention-based architectures suit irregular point sets, since they do not require a fixed grid [17], but applying self-attention to all N points costs $O(N^2)$. A common remedy is to project the points onto $M \ll N$ tokens, attend there, and map the result back. We refer to these two steps as projection and unprojection.

Existing projection mechanisms fall roughly into three categories. Geometric local pooling selects a well-spread set of centroids by farthest point sampling and aggregates each neighborhood by ball query [21] or k NN [22]; because aggregation only reaches nearby points, exchanging information between distant regions requires stacking many such layers. Grid-based projection instead maps the point cloud onto a regular latent grid and mixes globally in a single step, through spectral convolution [23] or ViT-style patch attention [24], but in three dimensions the token count grows cubically with resolution, so the resolution needed for thin shells or boundary layers conflicts with an affordable token budget. Learned assignment sidesteps this trade-off by scoring points against a fixed set of learnable vectors, with no intermediate grid, as in the cross-attention compression of Set Transformer [19] and Perceiver [20]. Transolver’s slice mechanism falls into this last category and is the direct basis for LTS.

A.2 Additional Details on Physics-Attention

Physics-Attention [1] proceeds in three steps: slicing groups the N nodes into M learned physics-aware slices, self-attention is applied among the resulting tokens, and deslicing reuses the same assignment weights in reverse. Grouping points by physical characteristics rather than by spatial proximity is argued to recover the underlying physical states that the discretization obscures, so that long-range dependencies induced by complex geometries can be modeled directly while attention stays linear in N . We refer to these M tokens as slice tokens; later works in the Transolver family call the same quantity physical states. The entire slice–deslice cycle is re-executed at every Physics-Attention block, so the cost of forming and dissolving the M tokens is paid again at every layer and accumulates with depth.

A.3 Transolver Variants

Several follow-up studies have proposed improvements to Transolver. Transolver++ [12] targets what it calls eidetic states, physical states that remain distinguishable once the mesh becomes very large. It introduces Ada-Temp, letting each point compute its own temperature from its own features to improve grouping quality, and Rep-Slice, a reparameterization that keeps the slice weight distribution from collapsing. Both are paired with a multi-GPU parallelization framework that scales to meshes with millions of points. Transolver-3 [13] reorders the slice–deslice computation using the associativity of matrix multiplication, and caches the slice tokens of each layer during inference. GeoTransolver [10] introduces Geometry-Aware Latent Embeddings (GALE). A context projector takes multi-scale ball-query features, global parameters, and boundary conditions, and projects them into the slice token space as a shared context, which is computed once and reused at every layer. Slice tokens attend to each other as before, but also cross-attend to this shared context, and the two results are combined as a convex combination weighted by a learnable per-layer scalar gate. GeoTransolver with FLARE (GeoTS-FLARE) [11] further replaces the Physics-Attention self-attention backend of GeoTransolver with the low-rank attention routing of FLARE, the Fast Low-rank Attention Routing Engine [18], so that global communication passes through a low-rank latent bottleneck instead of slicing. The geometry context pathway is left unchanged. LinearNO [7] shows that Physics-Attention is a special case of linear attention, unties its shared query/key projection, and removes slice attention entirely, reducing the per-block cost from $O(NM + M^2)$ to $O(NM)$ with 40.0% fewer parameters and 36.2% fewer FLOPs than Transolver.

Since LinearNO is the closest efficiency-oriented work, we state the distinction explicitly. It redesigns the operator inside a block, whereas LTS changes how often that block routes between the mesh and the latent space. LinearNO blocks are stacked and each still routes all N points, so $K = L$; its savings come from removing slice attention rather than from reducing K , which makes the two directions composable. We do not report it as a DropTest baseline: it also changes the projection parameterization, by untying the two projections and removing slice attention, so it cannot be used to isolate K with the projection mechanism held fixed, and it leaves the routing schedule at $K = L$, placing it on the same side of our comparison as $K = 12$ in Table 2.

These improvements target grouping quality, computational efficiency, geometric awareness, attention design, and operator formulation, respectively. In all of these works, however, point-to-token projection is retained at every layer, an aspect that is orthogonal to the improvements they set out to make. To the best of our knowledge, within the Transolver family, whether learned physics-aware slicing must be applied at every layer has not been studied in isolation.

A.4 Single-Pass Projection in Latent-Based Neural Operators

Single-pass projection, in which the mesh is projected into the latent space and unprojected back exactly once for the entire forward pass, has been explored by several prior works. UPT [5] samples the point cloud at the start of a rollout, aggregates it, and compresses it into a fixed number of latent tokens. The following timesteps then evolve entirely within this fixed latent, without re-slicing. This design also employs additional inverse encoding and decoding losses that force the encoder, approximator, and decoder to take on clearly separated roles. LNO [6] similarly transforms the data only at the first and last layers, with every intermediate layer operating solely on a fixed set of latent tokens. Both works provide evidence that projecting only once is feasible, at least within the settings they consider. LNO further reports an average 30% reduction in parameter count, a 50% reduction in GPU memory, and a $1.8\times$ training speedup over Transolver, and attributes these gains to transforming the data only once rather than switching between the latent and mesh space at every layer.

LNO includes an ablation, presented under the heading of decoupling, in which its Physics-Cross-Attention is replaced with Transolver’s Physics-Attention, so that queries and keys are again computed from both positions and physics quantities. The replacement is less accurate than LNO on all six forward benchmarks. Both variants, however, project exactly once, so the comparison isolates the slicing rule and not the number of times it is applied; the routing frequency is never varied for a fixed mechanism. Cost is not reported for either variant, even though systems cost is what motivates single-pass projection in the first place. Our routing-frequency ablation in Table 2 varies K directly on an industrial-scale mesh and reports cost and accuracy together.

A.5 Component-Aware Conditioning

Recent crash surrogate models have demonstrated the value of finite-element part information. CrashSolver, introduced with CarCrashNet [27], organizes the vehicle mesh into semantic structural components, applies shared local component encoders, mixes component summaries with a global Transformer, and exchanges information through a component-interface graph, reporting small but consistent benefits from dense component label conditioning. It uses the component hierarchy to organize physical interaction modeling throughout the network. Our objective is complementary: rather than partitioning the backbone into component-specific streams, LTS retains a single shared physics-aware latent representation and uses component context only after deslicing, so that component adaptation acts as a lightweight conditional decoder while preserving the persistent-latent backbone.

Where such conditioning is injected is itself a design choice. Within the Transolver family, GeoTransolver injects its geometry context inside the token stream, where slice tokens cross-attend to it at every layer. Conditioning applied outside the backbone, directly to the reconstructed node features, is well established in other settings [9] but has not been examined in this family, even though the reconstruction boundary is precisely where per-node component identity becomes available again. LTS adopts this latter placement in the output-side adaptation described in the main methodology.

A.6 DropTest Dataset Summary

For reference, Table 4 summarizes the DropTest dataset used in the main experiments; implementation details follow in Appendix B.1.

DropTest-192 contains 192 OpenRadioss trajectories from a deterministic $2^5 \times 6$ full-factorial design over five Young’s-modulus multipliers and six rigid-wall orientations. All other conditions are fixed. Each trajectory uses the same 76,065-node mesh and 100 states (one reference and 99 targets). Table 4 summarizes the dataset and DOE.

Table 4: DropTest-192 dataset and design-of-experiments setup.

Setting	Value
Solver	OpenRadioss 2026
DOE	$2^5 \times 6$ full factorial
Material scales	$5 \times \{0.8, 1.2\}$
Wall orientations	$(-10, 0, 0), (0, -10, 0), (0, 0, 0), (0, 0, 10), (0, 10, 0), (10, 0, 0)$ deg
Trajectories	192
Processed mesh	76,065 nodes
Simulation states	100 (state $r = 0$ reference; states $r = 1, \dots, 99$ targets)
Time-step indices	$r = 0, \dots, 99$
Model query index	$q = r - 1 = 0, \dots, 98; \tau_q = q/99$
Split	132 / 30 / 30
Split rule	Material-grouped
Global variables	5 material + 3 orientation

B Additional Experimental Details and Analyses

B.1 Implementation and Training Protocol

Table 5: Model architectures and common training protocol on DropTest.

Parameter	Transolver	Transolver++	Transolver-3	Geo Transolver	GeoTS- FLARE	LTS
<i>Architecture</i>						
Hidden Width	256	256	256	256	256	256
Layers	12	4	16	12	12	12
Physics Slices	256	32	64	256	256	256
<i>Common Training Protocol</i>						
Dataset Split	132 training / 30 validation / 30 held-out test trajectories					
Training	25 epochs; Adam; batch size = 1; BF16 autocast					
Learning Rate	$1 \times 10^{-4} \rightarrow 3 \times 10^{-7}$, CosineAnnealingLR ($T_{\max} = 25$), no warmup					
Training Objective	Channel-mean MSE in the training-normalized output space					
Seeds / Hardware	{7, 17, 27}; one NVIDIA H200 GPU per model and seed					

Notes. External baselines retain their released backbone configurations. LTS uses the Transolver-volume-scale persistent-latent backbone with Fourier encoding and output-side component/displacement adaptation; its fixed development configuration is described below.

Implementation and training. Table 5 summarizes model configurations and the common training protocol; Table 4 summarizes DropTest. Inputs are normalized initial coordinates, eight global variables, and model query τ_q ; outputs are displacement and von Mises stress. Normalization uses training data only, and checkpoints are selected by the equal-weight validation average of trajectory-macro displacement and stress relative L_2 errors. LTS uses a fixed 17-component partition transferred from a topology-identical annotated reference mesh after verifying node ordering and topology, since canonical DropTest files omit PART_ID.

Dataset split. The 192 trajectories form 32 material groups, each paired with the same six rigid-wall orientations. Groups are indexed by the five binary Young’s-modulus settings for material IDs $\{1, 4, 5, 8, 9\}$, with 0/1 denoting multipliers 0.8/1.2. Test groups are $\{00000, 00111, 10100, 11001, 11110\}$ and validation groups are $\{00101, 01010, 01011, 10000, 11111\}$; the remaining 22 groups are used for training. The split is deterministic and contains no material-group overlap. Material IDs 1, 4, 5, 8, and 9 correspond respectively to the rear cover, battery, LCD glass, top/bottom PCBs, and IC/connector/camera/microchip components.

Held-out evaluation. Predictions are transformed back to physical units before evaluation. For trajectory j and field f , we compute

$$e_{j,f} = \frac{\left(\sum_{r=1}^{99} \sum_{n=1}^N \left\| \hat{\mathbf{y}}_{j,n,r}^f - \mathbf{y}_{j,n,r}^f \right\|_2^2 \right)^{1/2}}{\max \left[\left(\sum_{r=1}^{99} \sum_{n=1}^N \left\| \mathbf{y}_{j,n,r}^f \right\|_2^2 \right)^{1/2}, 10^{-12} \right]}, \quad (6)$$

where $N = 76,065$. The displacement norm jointly includes all three Cartesian channels, while stress is scalar. The test metric is the macro-average $\frac{1}{30} \sum_{j=1}^{30} e_{j,f}$, and reported accuracy is the mean $\pm$ sample s.d. across seeds $\{7, 17, 27\}$. Thus, all 99 target states are accumulated within each trajectory before the relative norm is taken; per-state relative errors are not averaged. Tables 1–3, 6, and 7 display two-decimal results; rankings and relative changes use unrounded values.

Per-state absolute-error metrics. For each trajectory and target time step, displacement RMSE is $\sqrt{\text{SSE}_D/(3N)}$ over scalar XYZ entries and stress RMSE is $\sqrt{\text{SSE}_S/N}$. Displacement is measured in mm; stress RMSE is evaluated on the unnormalized stress field. The curves first average the 30 per-trajectory RMSE values, then the three seed means; shading is the observed seed range. GT RMS uses the same divisors on squared target norms. Its case mean grows from 0.0031798 to 0.3056901 mm for displacement and increases by $122.5\times$ for stress between $r = 1$ and $r = 99$. The identity $\text{Rel}L_2 = \text{RMSE}/\text{GT RMS}$ holds separately for each trajectory and state; the mean of these ratios generally differs from the ratio of their means.

Efficiency profiling. All measurements use one NVIDIA H200 GPU with batch size one and BF16 autocast. Inputs are resident on the GPU before measurement, and data loading and host-to-device transfer are excluded. Train-step latency is measured using CUDA events after 20 warm-up iterations and over 50 timed iterations. For every model, one explicit Python cyclic-GC collection is performed before timing, and cyclic GC is disabled throughout the timed window.

The timed region begins immediately before the forward pass and ends after the fused Adam update. It includes the BF16 forward pass, target normalization, normalized-loss computation, backward pass, and `optimizer.step`. The call to `optimizer.zero_grad(set_to_none=True)` occurs before the start event and is excluded. The ending CUDA event is synchronized before its elapsed time is read.

Peak allocated memory is measured over the same profiling window using `torch.cuda.reset_peak_memory_stats()` before measurement and `torch.cuda.max_memory_allocated()` afterward. Train-step latency is reported as mean $\pm$ sample s.d. over the 50 seed-7 iterations, whereas peak memory is the single maximum observed during the window.

Inference uses `model.eval()`, `torch.no_grad()`, and BF16 autocast. The 99 target queries are evaluated sequentially, with CUDA synchronized around each trajectory-level timing interval. Each seed is averaged over the 30 held-out trajectories, and inference latency is reported as mean $\pm$ sample s.d. across the three seed-level averages. No model uses `torch.compile`. The profiling environment uses PyTorch 2.13.0, CUDA 13.0, and cuDNN 9.20.

Fixed LTS development configuration. The LTS configuration reported in the formal comparisons was frozen after preliminary architecture development. Fourier bandwidth $B = 10$ and displacement-rescaling MLP width 64 were chosen as heuristic capacity settings rather than as outcomes of a

systematic hyperparameter sweep; the Fourier projector width is 256, matching the LTS backbone width. The inclusion of Fourier encoding, component adaptation, and displacement rescaling was informed by preliminary validation experiments. After this development stage, these settings were fixed for the reported LTS (Full) model. The main results, systems profiling, and qualitative analysis use this configuration unchanged; the routing-frequency and component-ablation studies vary only the factor being studied while retaining the remaining development settings. Model checkpoints are selected using the common validation criterion described above. The held-out test set was not used for architecture, hyperparameter, or checkpoint selection.

Routing-frequency implementation. For $K = 1, 2, 3, 4, 6, 12$, the twelve latent blocks are partitioned as $[12]$, $[6, 6]$, $[4, 4, 4]$, $[3, 3, 3, 3]$, $[2, 2, 2, 2, 2, 2]$, and $[1, \dots, 1]$, respectively.

B.2 Efficiency and Qualitative Analysis

Figure 3 provides complementary qualitative and CUDA-time analyses on DropTest. Both panels use the same fixed LTS (Full) configuration reported in the main experiments. The accuracy–efficiency bubble comparison is shown in the main text in Figure 2(c).

To visualize a case of typical aggregate difficulty rather than one particularly favorable to any individual model, we use a deterministic median-based selection rule. For each of the 30 held-out trajectories under seed 7, we compute the mean of the per-case macro relative L_2 errors from Transolver, GeoTransolver, GeoTS-FLARE, Transolver++, Transolver-3, and LTS (Full). We then select the trajectory whose six-model mean error is closest to the median across the 30 cases. `run0152` and `run0154` are exactly equidistant from the median, and a deterministic case-ID tie-break selects `run0152`. This selection uses held-out quantitative errors and is therefore post hoc, but it does not use visual inspection or the relative performance difference between LTS and any particular baseline. The final time step $r = 99$ was fixed in advance.

Both absolute-error maps share a common color range, enabling a direct visual comparison between Transolver and LTS (Full). Accordingly, the stress-error panel is intended as a descriptive visualization of a median-difficulty held-out case rather than as independent evidence of aggregate model performance.

The CUDA-time panel complements the main-text accuracy–efficiency comparison with an instrumented breakdown of training computation.

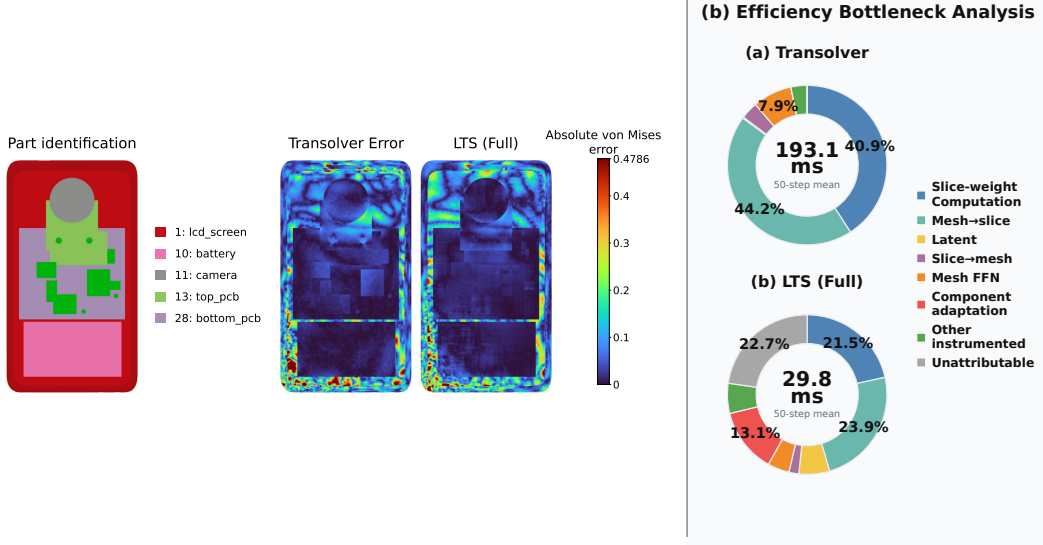


Figure 3: Additional qualitative and CUDA-time analyses on DropTest. Left: component identification and held-out absolute von Mises stress-error maps for Transolver and LTS (Full) on run0152 at time step $r = 99$, corresponding to query slot $q = 98$ and model query $\tau_q = 98/99$. Both error maps use the same color range. The trajectory is selected using the median six-model mean per-case error under seed 7. Right: instrumented CUDA-time breakdown for Transolver and LTS (Full). The accuracy–efficiency bubble panel is moved to Figure 2(c).

CUDA-time breakdown. CUDA kernels are assigned to non-overlapping instrumented regions. Routing-related regions include slice-weight computation, mesh-to-slice aggregation, and slice-to-mesh reconstruction; latent processing covers slice-token attention and feed-forward blocks, while component adaptation covers the output-side conditioned refinement. Backward kernels are associated with the corresponding forward region using PyTorch autograd sequence identifiers; unmatched kernels are grouped separately. Results are averaged over five profiler repetitions. Routing-related operations account for 88.5% of the dedicated CUDA-event train-step time in Transolver and 47.5% in LTS (Full), while component adaptation accounts for 13.1% in LTS (Full).

C Additional Steady-State Evaluation on DrivAerML

We additionally report DrivAerML [28] Volume and Surface results in Tables 6 and 7. These steady-state experiments use a separately configured LTS backbone; they are not the depth-matched DropTest routing ablation. LTS (Base) retains the legacy eight-layer, 512-slice backbone without CA. LTS (Full) adds CA, 10-band coordinate Fourier features, and a static input-conditioned target gate. The gate uses input mean/RMS instead of DropTest’s time-dependent envelope; no additional dataset field or test signal is used.

Both tables report accuracy as mean $\pm$ sample s.d. over seeds $\{7, 17, 29\}$, with relative L_2 errors in percent. Seed-7 systems profiling uses one H200, BF16, batch size one, 100,000 points, 20 warm-up steps, and 100 timed optimizer steps. Training latency is reported as the timed-step mean $\pm$ sample s.d. The table notes specify the distinct evaluation protocols.

Table 6: DrivAerML Volume results.

Model	Macro Rel. L_2 (%)↓	Velocity Rel. L_2 (%)↓	Pressure Rel. L_2 (%)↓	ν_t Rel. L_2 (%)↓	Train-step (ms)↓	Peak train alloc. (GiB)↓	Params. (M)↓
Transolver	10.43 ± 0.07	16.68 ± 0.01	5.85 ± 0.03	8.76 ± 0.17	251.00 ± 0.18	36.90	8.96
GeoTransolver	10.12 ± 0.04	16.42 ± 0.02	5.26 ± 0.03	8.69 ± 0.10	483.69 ± 0.37	59.13	27.52
GeoTS-FLARE	10.06 ± 0.02	16.36 ± 0.01	5.15 ± 0.01	8.66 ± 0.06	377.37 ± 0.39	39.13	26.27
Transolver++	12.26 ± 0.04	18.45 ± 0.04	7.88 ± 0.05	10.46 ± 0.04	29.38 ± 0.15	4.57	1.74
Transolver-3	10.50 ± 0.03	16.74 ± 0.02	5.90 ± 0.02	8.84 ± 0.09	419.77 ± 29.10	3.84	7.60
LTS (Base)	15.77 ± 0.31	21.67 ± 0.37	11.49 ± 0.37	14.16 ± 0.22	47.59 ± 0.25	11.98	5.64
LTS (Full)	11.68 ± 0.13	15.67 ± 0.14	7.84 ± 0.12	11.52 ± 0.16	60.11 ± 0.41	12.24	5.96
vs. Transolver	↑ 1.25 pp	↓ 1.01 pp	↑ 1.99 pp	↑ 2.76 pp	↓ 76.1%	↓ 66.8%	↓ 33.5%

Notes. Training uses 436 cases; evaluation uses 48 held-out cases at the fixed final epoch 200. Macro error equally averages velocity, pressure, and turbulent-viscosity relative L_2 errors. The final row compares LTS (Full) against Transolver: accuracy differences are in percentage points (pp); efficiency and parameter changes are relative percentages.

Table 7: DrivAerML Surface results.

Model	Area-weighted Rel. L_2 (%)↓	Unweighted Rel. L_2 (%)↓	Train-step (ms)↓	Peak train alloc. (GiB)↓	Inference (ms/case)↓	Params. (M)↓
Transolver	37.29 ± 2.40	42.00 ± 4.44	169.59 ± 0.16	25.78	39.61 ± 0.03	6.02
GeoTransolver	33.69 ± 1.81	37.00 ± 3.05	272.59 ± 0.18	38.91	62.59 ± 0.03	9.01
GeoTS-FLARE	8.32 ± 1.61	11.98 ± 1.96	151.77 ± 0.18	16.05	40.55 ± 0.02	8.90
Transolver++	51.45 ± 4.82	54.81 ± 3.76	29.84 ± 0.03	4.57	10.58 ± 0.01	1.74
Transolver-3	38.12 ± 1.02	43.86 ± 2.24	635.47 ± 37.29	5.42	99.19 ± 0.08	11.33
LTS (Base)	40.78 ± 5.15	50.56 ± 5.44	42.40 ± 0.06	11.98	9.82 ± 0.02	5.64
LTS (Full)	22.52 ± 0.90	29.10 ± 2.67	49.71 ± 0.09	12.09	13.87 ± 0.05	5.95
vs. Transolver	↓ 14.77 pp	↓ 12.90 pp	↓ 70.7%	↓ 53.1%	↓ 65.0%	↓ 1.2%

Notes. Errors are case-macro averages over 42 held-out cases; checkpoints minimize validation area-weighted relative L_2 error. The final row compares LTS (Full) against Transolver: accuracy differences are in percentage points (pp); efficiency and parameter changes are relative percentages.