A. Implementation

A.1. Balltree Partitioning

Ball trees. A ball tree is a hierarchical spatial partitioning data structure that recursively decomposes a point set into nested metric balls. Given a set of points $\mathcal{P} \subset \mathbb{R}^D$, each node u in the tree stores a subset $\mathcal{P}_u \subseteq \mathcal{P}$ together with a center $\mathbf{c}_u \in \mathbb{R}^D$ and radius $r_u > 0$ such that

$$\mathcal{P}_u \subseteq B(\mathbf{c}_u, r_u) := \{\mathbf{x} \in \mathbb{R}^D : \|\mathbf{x} - \mathbf{c}_u\|_2 \leq r_u\}.$$

The root node contains all points. Internal nodes recursively split their point set into two (approximately) balanced children, each associated with a tighter enclosing ball, until a prescribed leaf capacity is reached.

Construction. In our implementation we build a balanced binary ball tree from the point coordinates using the public `balltree-erwin` [47]. Starting from the full set $\mathcal{P}$, each internal node u is constructed by:

1. choosing a splitting direction based on a pair of distant points in $\mathcal{P}_u$;
2. partitioning $\mathcal{P}_u$ into two subsets of roughly equal size along this direction;
3. fitting enclosing balls for each child subset by setting the center to the mean and the radius to the maximal distance to that mean.

This produces a tree of depth $O(\log N)$ with $O(N)$ nodes, and the overall construction cost is $O(N \log N)$.

Leaf ordering and patch extraction. To obtain patches for PMSA, we traverse the leaves of the ball tree in depth-first order and record the points in that leaf order. This induces a permutation π of $\{1, \ldots, N\}$ such that nearby indices in the permuted sequence correspond to spatially close points. We then form patches by taking contiguous blocks of length L in this ordered sequence:

$$\mathcal{P}_k = \left\{\mathbf{p}_{\pi((k-1)L+1)}, \ldots, \mathbf{p}_{\pi(kL)}\right\}, \qquad k = 1, \ldots, K,$$

with padding applied if N is not divisible by L. In this view, each patch can be interpreted as a collection of leaves and small subtrees that occupy a localized region of the domain. Once the permutation π is fixed, all subsequent transformer blocks operate on the reordered sequence and can use simple linear splitting into contiguous patches.

Reuse across blocks. We emphasize that the ball tree is constructed *once* per input sample, based solely on the initial coordinates. The induced permutation and patch layout are reused across all MSPT blocks. This avoids the cost of rebuilding trees at each layer while preserving spatial locality throughout the network. For inputs that already come with a natural ordering on a regular grid, we skip the ball-tree construction and directly apply linear partitioning, which can be viewed as a degenerate case of ball-tree partitioning.

Patch locality diagnostic. To assess whether depth-first leaf traversal introduces occasional non-local "jumps" that mix distant points within a patch, we measure the spatial dispersion of each extracted patch after applying the ball-tree-induced permutation. For a patch $\mathcal{P}_k$ with points $\{x_i\}_{i \in \mathcal{P}_k}$, we compute

$$s_k = \frac{1}{|\mathcal{P}_k|} \sum_{i \in \mathcal{P}_k} \|x_i - \bar{x}_k\|_2^2, \qquad \bar{x}_k = \frac{1}{|\mathcal{P}_k|} \sum_{i \in \mathcal{P}_k} x_i,$$

(i.e., the trace of the within-patch coordinate covariance / mean squared radius), and plot a histogram of $\{s_k\}_{k=1}^K$. For example, in the Darcy benchmark (with $K = 300$ patches), Figure 6 shows that the patch-dispersion distribution under ball-tree DFS ordering is concentrated at low values with only a small outlier tail, indicating that the induced permutation yields predominantly spatially coherent patches in practice; in contrast, a random permutation produces substantially larger dispersion (Table 6). Quantitatively, Table 6 reports the median / p90 / p99 / max of s_k across benchmarks and shows substantially smaller dispersion under ball-tree ordering than under a random permutation (e.g., Elasticity: median 0.0065 vs. 0.2035; Darcy: median 0.0012 vs. 0.1672).

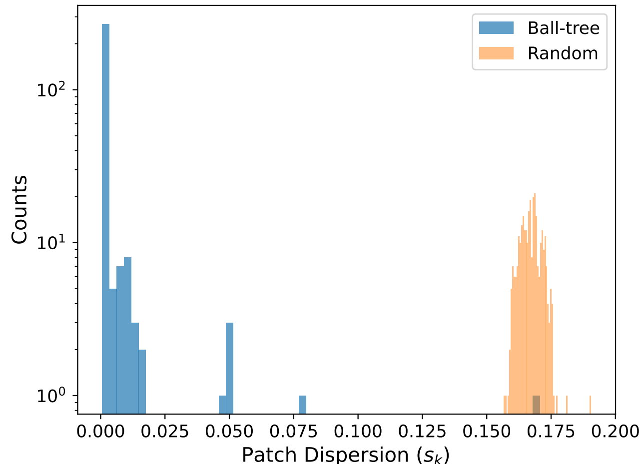

Figure 6. Histogram of per-patch spatial dispersion s_k on Darcy after ball-tree DFS ordering and contiguous patch extraction ($K = 300$; log-scaled y-axis). Ball-tree ordering yields predominantly compact patches (mass concentrated at low dispersion) with a small outlier tail, while a random permutation produces substantially larger dispersion (Table 6).

Table 6. Patch-dispersion statistics across PDE benchmarks (mean squared radius within each patch; lower is better). We report the median, 90th percentile (p90), 99th percentile (p99), and maximum over patches for the ball-tree DFS ordering versus a random permutation, using the same number of patches K as in our main experiments.

		Ball-tree ordering				Random ordering			
Benchmark	K	median	p90	p99	max	median	p90	p99	max
Elasticity	32	0.006523	0.0475	0.3238	0.3998	0.2035	0.2173	0.2274	0.2312
Plasticity	64	0.00435	0.01515	0.1136	0.1675	0.1728	0.1909	0.1991	0.2045
Airfoil	106	0.03324	133.3	435.9	450.6	77.3	118.3	154.3	155.2
Pipe	64	0.04324	0.1611	6.678	17.65	8.937	9.383	10.18	10.6
Navier–Stokes	32	0.006677	0.006677	0.006677	0.006677	0.171	0.1829	0.1866	0.1868
Darcy	300	0.001223	0.003719	0.05023	0.1706	0.1672	0.1728	0.1759	0.1905

A.2. Choosing the Number of Patches K

Beyond efficiency considerations, the optimal number of patches K depends on both the input resolution (N) and the nature of the underlying physical interactions. Problems dominated by local interactions (e.g., solid/material settings such as Elasticity and Plasticity) typically require less global coupling, whereas fluid dynamics settings (e.g., Airfoil, Pipe, Navier–Stokes, Darcy) often benefit from stronger long-range communication due to advective transport and global pressure coupling. In MSPT, K controls a trade-off between (i) local context per patch, $L = N/K$, and (ii) global communication capacity via pooled tokens, proportional to KQ supernodes. At the extremes, $K \to N$ (so $L \to 1$) approaches a global-attention regime similar in spirit to a full Transformer baseline, while $K = 1$ reduces to a single patch with a pooled global token (analogous to a [CLS]-style global summary).

A.3. Geometric Descriptors

Following Transolver [44], we form input tokens by concatenating positional or geometric descriptors with the available physical-state features. We use either raw coordinates or a distance-to-reference-grid descriptor $g_i = \texttt{get_grid}(\mathbf{pos}) \in \mathbb{R}^{\texttt{ref}^d}$ (with $d \in \{2, 3\}$ depending on whether the domain is 2D or 3D). Across all experiments, we use the same geometric descriptors and preprocessing as Transolver. Additionally, we apply Rotary Positional Embeddings (RoPE) to the query and key projections within each patch prior to the dot-product attention [36].

A.4. Benchmarks

The model is evaluated on eight benchmarks (see Table 1) covering three types of partial differential equations (PDEs): **Solid material** (Elasticity and Plasticity) [9], **Navier-Stokes equations for fluid** [30] (Airfoil, Pipe, Navier-Stokes, Shape-Net Car, AhmedML), and **Darcy flow**[16]. The specific details of each benchmark are provided below.

Elasticity This benchmark estimates internal stress in elastic materials from their structure, discretized into 972 points [24]. Each input is a 972×2 tensor for 2D positions, and the output is a 972×1 stress tensor. There are 1000 training samples with various structures and 200 test samples.

Plasticity This benchmark predicts the future deformation of plastic materials subjected to impact from an arbitrarily shaped die [24]. For each case, the input is the die shape, discretized into a structured mesh and represented as a 101×31 tensor. The output is the deformation at each mesh point over 20 future time steps, recorded as a $20 \times 101 \times 31 \times 4$ tensor, capturing deformation in four directions. The experimental protocol utilizes 900 training samples with diverse die shapes and 80 test samples.

Airfoil This task estimates the Mach number based on the airfoil shape, with the input discretized into a structured mesh of size 221×51 and the output representing the Mach number at each mesh point [24]. All shapes are derived from the NACA-0012 case provided by the National Advisory Committee for Aeronautics. The dataset comprises 1000 training samples with various airfoil designs and 200 samples for testing.

Pipe This benchmark estimates the horizontal fluid velocity based on the pipe structure [24]. Each case discretizes the pipe into a structured mesh of size 129×129. The input tensor, shaped $129 \times 129 \times 2$, encodes the positions of the discretized mesh points, while the output tensor, shaped $129 \times 129 \times 1$, represents the velocity at each point. The training set includes 1000 samples with varying pipe shapes, and 200 test samples are generated by modifying the pipe centerline.

Navier-Stokes This benchmark models incompressible and viscous flow on a unit torus, with constant fluid density and viscosity set to 10^{-5} [20]. The fluid field is discretized into a 64×64 regular grid. The task involves predicting the fluid state for the next 10 time steps based on observations from the previous 10 steps. The dataset consists of 1000 training samples with varying initial conditions and 200 samples for testing.

Darcy This benchmark models fluid flow through a porous medium [20]. The process is initially discretized into a 421×421 regular grid and subsequently downsampled to 85×85 resolution for the main experiments, following Wu et al. [44]. The model input is the structure of the porous medium, and the output is the fluid pressure at each grid point. The dataset includes 1000 training samples and 200 test samples, each with distinct medium structures.

Shape-Net Car This benchmark addresses the estimation of drag coefficients for moving cars, a critical factor in automotive design. A total of 889 samples with diverse car shapes are generated to simulate driving at 72 km/h [38], using car models from the ShapeNet "car" category [5]. The simulation discretizes the space into an unstructured mesh with 32,186 points, capturing both the surrounding air and surface pressure. Following the experimental setup in 3D-GeoCA [7], 789 samples are used for training and 100 for testing. Each input mesh is preprocessed to include mesh point positions, signed distance functions, and normal vectors. The model predicts velocity and pressure at each point, enabling subsequent calculation of the drag coefficient from the estimated physical fields.

AhmedML AhmedML [3] provides 500 parameterized geometries simulated with a hybrid RANS–LES solver (OpenFOAM v2212) for 80 convective time units on ~ 20 M-cell unstructured meshes. Each case includes the body surface mesh, full volumetric fields (velocity and pressure), boundary slices, geometry parameters, and time-averaged forces.

A.5. Metrics

Since our experiment consists of standard benchmarks and practical design tasks, we also include several design-oriented metrics in addition to the relative L_2 for physics fields.

Relative L_2 for physics fields Given the physics field $\mathbf{u}$ and the model-predicted field $\widehat{\mathbf{u}}$, the relative L_2 of the model prediction is calculated as follows:

$$\text{Relative L}_2 = \frac{\|\mathbf{u} - \widehat{\mathbf{u}}\|}{\|\mathbf{u}\|}. \tag{12}$$

Relative L_2 for drag coefficient For ShapeNet-Car, the drag coefficient is calculated based on the estimated physics fields, following the same implementation as Wu et al. [44]. For a unit-density fluid, the coefficient is defined as

$$C = \frac{2}{v^2 A}\left(\int_{\partial\Omega} p(\boldsymbol{\xi})\left(\widehat{n}(\boldsymbol{\xi})\cdot\widehat{i}(\boldsymbol{\xi})\right)\mathrm{d}\boldsymbol{\xi} + \int_{\partial\Omega}\tau(\boldsymbol{\xi})\cdot\widehat{i}(\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi}\right), \tag{13}$$

where v is the speed of the inlet flow, A is the reference area, $\partial\Omega$ is the object surface, p denotes the pressure, $\widehat{n}$ is the outward unit normal vector of the surface, $\widehat{i}$ is the direction of the inlet flow, and τ denotes wall shear stress on the surface. τ can be calculated from the air velocity near the surface [29], and is usually much smaller than the pressure term. For ShapeNet-Car, $\widehat{i}$ is set as $(-1, 0, 0)$ and A is the area of the smallest rectangle enclosing the front of the car. The relative L_2 is defined between the ground-truth drag coefficient and the coefficient calculated from the predicted velocity and pressure fields.

Spearman's rank correlation for drag coefficient Given M samples in the test set with ground-truth drag coefficients $T = \{T^1, \cdots, T^M\}$ and model-predicted coefficients $\widehat{T} = \{\widehat{T}^1, \cdots, \widehat{T}^M\}$, the Spearman correlation coefficient is defined as the Pearson correlation coefficient between the rank variables:

$$\rho = \frac{\mathrm{cov}\left(R(T), R(\widehat{T})\right)}{\sigma_{R(T)}\sigma_{R(\widehat{T})}}, \tag{14}$$

where R is the ranking function, cov denotes the covariance, and σ represents the standard deviation of the rank variables. This metric quantifies how well the model preserves the ordering of designs by drag, which is directly relevant for design optimization: higher ρ indicates that it is easier to identify good designs from the model's predictions [35].

A.6. Implementation Details

For the Standard PDE Benchmarks and ShapeNet-Car, we follow the training procedure outlined by Wu et al. [44], using the *Neural Solver Library*[7]; MSPT does not restrict the training objective and can incorporate additional physics-informed regularizers (e.g., PDE residual, divergence, or boundary-condition penalties), although we report results with the standard benchmark objectives for fair comparison. For AhmedML, we adopt a training protocol similar to AB-UPT [2], using 400 training samples, 50 validation samples, and 50 testing samples selected at random. To fit within a single A100 GPU (40 GB), each AhmedML mesh is partitioned into non-overlapping sub-meshes of 320k points (160k for surface and 160k for volume), enabling multiple blocks to be processed concurrently. For models trained with the LION optimizer [6], we use a peak learning rate of 5×10^{-5}, weight decay of 5×10^{-2}, and a linear warmup over the first 5% of training, followed by cosine decay to a final learning rate of 1×10^{-6}, as detailed in AB-UPT [2]. Models not trained with LION use the optimizer, and learning rate configurations implemented in the *Neural Solver Library*. The complete training and model configurations for all benchmarks are summarized in Table 7.

Table 7. Training and model configurations of MSPT. Here $\mathcal{L}_{\mathrm{v}}$ and $\mathcal{L}_{\mathrm{s}}$ represent the loss on volume and surface fields respectively. As for Darcy, we adopt an additional spatial gradient regularization term $\mathcal{L}_{\mathrm{g}}$ following ONO [45].

Benchmarks	Training Configuration (Shared in all baselines)					Model Configuration					
	Loss	Epochs	Initial LR	Optimizer	Batch Size	Layers L	Heads	Channels C	Patches K	Supernodes Q	Pooling
Elasticity	Relative L2	500	10^{-6}	LION [6]	1	8	8	128	32	1	Mean
Plasticity			10^{-3}	AdamW [26]	8			128	64		
Airfoil					4			128	106		
Pipe					4			128	64		
Navier–Stokes					2			256	32		
Darcy	$\mathcal{L}_{\mathrm{rL2}} + 0.1\mathcal{L}_{\mathrm{g}}$		10^{-6}	LION [6]	1			128	300		
Shape-Net Car	$\mathcal{L}_{\mathrm{v}} + 0.5\mathcal{L}_{\mathrm{s}}$	200	10^{-6}	LION [6]	1	12	8	256	170	1	Mean
Ahmed ML		100							400		

[7] https://github.com/thuml/Neural-Solver-Library

A.7. Efficiency Comparison with Baselines

Table 8. Model efficiency comparison in Elasticity (Relative L_2) and Shape-Net Car (ρ_D), where we select Transolver and four other Transformer-based baseline methods that can be applied to unstructured meshes. Baselines are taken from Wu et al. [44]. Efficiency is evaluated on inputs of different meshes during training. Running time is measured by the time to complete one epoch, which contains 10^3 iterations. "/" indicates the baseline will fail in this benchmark. For a like-for-like comparison, we set MSPT to $K = 32$ patches to match Transolver's $S = 32$ slices; unlike Transolver whose runtime increases with larger S, MSPT can reduce runtime by increasing K (i.e., using smaller patches).

Model	Input Mesh Size	GPU Memory	Running Time	Elasticity	Shape-Net Car
	N	(GB)	(s / epoch)	(972 mesh points)	(32,186 mesh points)
MSPT (Ours)	1024	0.17	46.6095	0.0048	0.9941
	2048	0.22	46.8275		
	4096	0.33	46.6800		
	8192	0.53	46.4544		
	16384	0.89	47.1659		
	32768	1.65	65.0738		
	100000	4.68	192.5923		
	200000	9.13	365.9022		
	500000	22.49	797.7500		
Transolver [44]	1024	0.13	36.5752	0.0064	0.9935
	2048	0.20	36.9748		
	4096	0.34	37.3330		
	8192	0.63	36.5046		
	16384	1.21	37.3509		
	32768	2.37	41.6690		
	100000	7.05	104.3364		
	200000	14.06	199.3195		
	500000	34.94	547.3848		
GNOT [14]	1024	0.85	54.282	0.0086	0.9833
	2048	1.07	55.939		
	4096	1.47	60.857		
	8192	2.33	67.170		
	16384	4.23	112.552		
	32768	7.46	209.923		
ONO [45]	1024	1.47	69.759	0.0118	/
	2048	1.75	76.245		
	4096	2.30	100.134		
	8192	3.47	149.598		
	16384	5.64	255.339		
	32768	10.09	462.459		
OFormer [22]	1024	0.63	28.147	0.0183	/
	2048	0.69	30.983		
	4096	0.80	31.113		
	8192	1.02	47.904		
	16384	1.67	91.671		
	32768	2.44	182.205		
Galerkin Transformer [4]	1024	0.62	26.507	0.0240	0.9764
	2048	0.66	26.503		
	4096	0.74	27.481		
	8192	0.91	37.098		
	16384	1.45	67.524		
	32768	2.05	129.872		

B. Additional Visualizations

In this section, we provide additional results comparing the performance of MSPT and Transolver on a set of benchmark PDE problems.

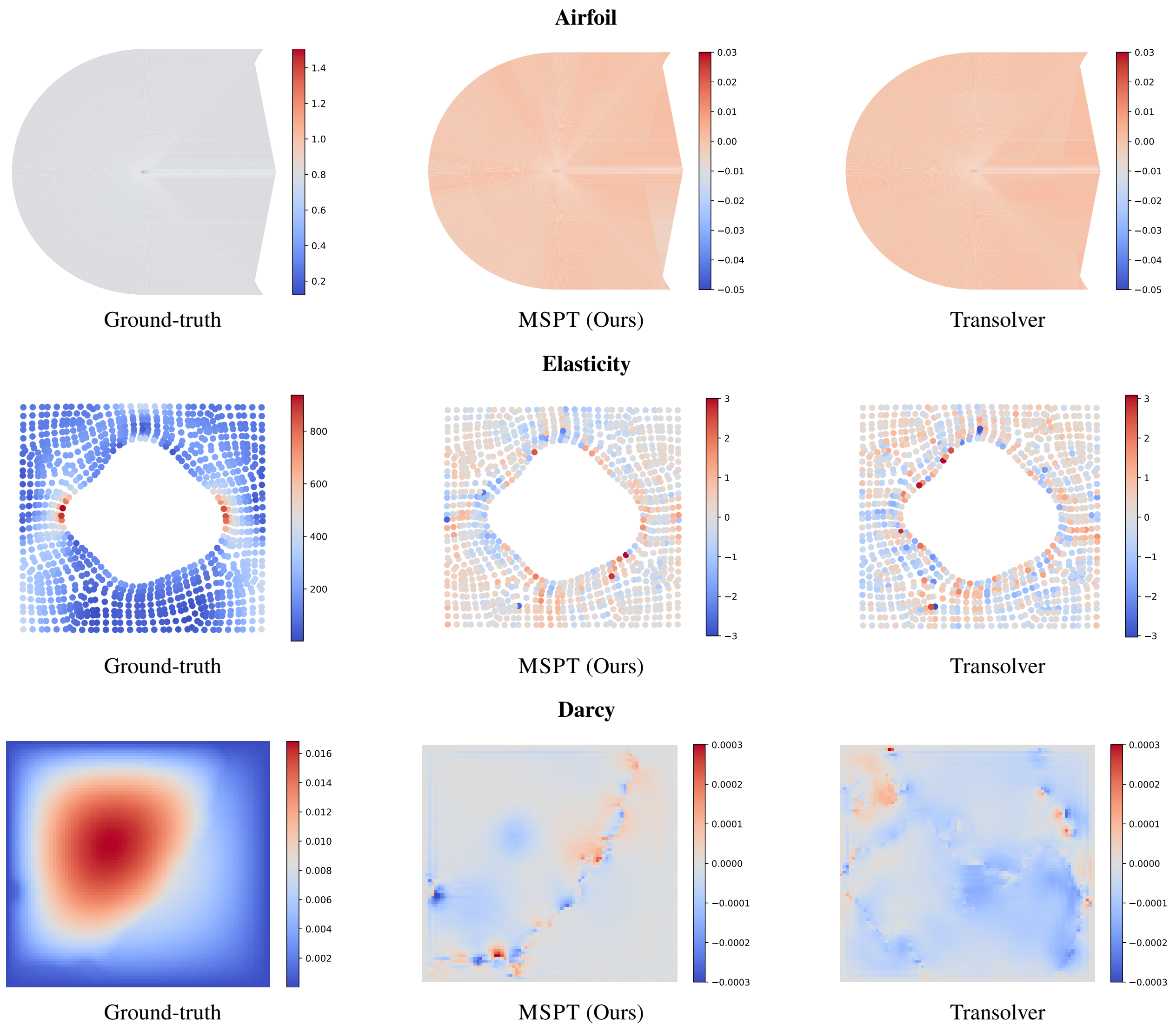

Figure 7. Visual comparison of per-point relative error maps across standard PDE benchmarks. Each row corresponds to one dataset (Airfoil, Elasticity, Darcy), and columns show the ground-truth field, MSPT, and Transolver.

C. Reproducibility Results

C.1. Transolver++

Transolver++ [27] is a parallelizable and more efficient successor to Transolver [44]. We trained Transolver++ using the official implementation[8] and adapted it to all geometries following the Transolver setup, with only minimal adjustments for structured meshes. Training was carried out using the Neural-Solver-Library framework[9], matching the experimental settings reported in Luo et al. [27].

As shown in Tables 9 and 10, Transolver++ (*repr.*) performs worse than both Transolver and MSPT. Similar reproduction differences were also reported by AB-UPT [2]. To avoid over-interpreting discrepancies, we do not include Transolver++ in the main benchmark comparisons and instead report its reproduced results here for completeness.

Table 9. **Reproducibility comparison on standard PDE benchmarks.** Relative L_2 errors are reported. "Transolver++ (repr.)" refers to our reproduction using the official implementation. "/" indicates inapplicability. All values are shown in units of $\times 10^{-2}$.

Model	Point Cloud	Structured Mesh			Regular Grid	
	Elasticity	Plasticity	Airfoil	Pipe	Navier Stokes	Darcy
Transolver [44]	0.64	0.12	0.53	0.33	9.00	0.57
Transolver++ [27]	0.52	0.11	0.48	0.27	7.19	0.49
Transolver++ (*repr.*) [27]	1.54	0.54	0.75	0.55	12.30	0.90
MSPT (Ours)	0.48	0.10	0.51	0.31	6.32	0.63

Table 10. **Reproducibility comparison on ShapeNet-Car.** Relative L_2 errors for volume and surface fields, and drag coefficient (C_D); Spearman ρ_D for ranking quality. "Transolver++ (repr.)" refers to our reproduction using the official implementation. All models are trained under identical experimental conditions. Values are shown in units of $\times 10^{-2}$.

Model	**ShapeNet-Car**			
	Volume $\downarrow$	Surf $\downarrow$	$(C_D)\downarrow$	$(\rho_D)\uparrow$
Transolver [44]	2.07	7.45	1.03	99.35
Transolver++ (*repr.*) [27]	2.37	8.37	1.41	99.24
MSPT (Ours)	1.89	7.41	0.98	99.41

C.2. AB-UPT

Since the official AB-UPT [2] training pipeline is not publicly available, we reproduced its results using the *Neural-Solver-Library* (following the same training procedure as Wu et al. [44]) for the ShapeNet-Car dataset, and our own training pipeline for AhmedML, following the model configurations and hyperparameters (e.g., optimizer, learning rate, batch size) reported in their paper. The reproduced results differ from the originally published values, which is consistent with the authors' note that their framework applies shared modifications across all models and that deviations can arise from training randomness and implementation details. In our experiments, similar discrepancies also appear when retraining other baselines under a unified pipeline, suggesting that part of the gap is due to differences in training frameworks rather than the architectures themselves. Accordingly, in the main text we report the original AB-UPT numbers for consistency with their publication, and provide our reproduced results and analysis here for completeness.

[8] https://github.com/thuml/Transolver_plus
[9] https://github.com/thuml/Neural-Solver-Library