Universiteit van Amsterdam

HAET

Hierarchical Attention Erwin Transolver for Large Scale Mesh Processing

Pedro M. P. Curvo
Mahdi Rahimi
Salvador Torpes

Index

Motivation and Problem

Motivation and Problem

Point Clouds and Meshes

Point Clouds/Meshes are essential for **Physical Simulations**

↓

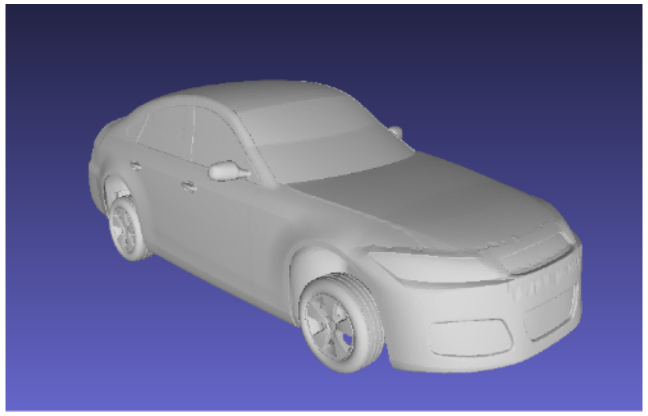
Real-world simulations involve **millions of irregular points**

↓

Processing must **preserve geometric structure and physical fields**

↓

Current PDE solvers **struggle** with such large inputs



Example from the
DrivAerNet benchmark

Motivation and Problem
Computational Challenges

- Transformer scale poorly: $O(N^2)$ **attention is impractical** at large N
- Possible solution: **Downsampling** = leads to **loss of physical detail** since points are strongly dependent on each other
- Models must:
  - Preserve local features
  - Capture long-range dependencies
  - Scale efficiently

Existing Solutions

Existing Solutions
General

Better efficiency in attention via geometric ball trees

↑

Erwin

Transolver++

↓

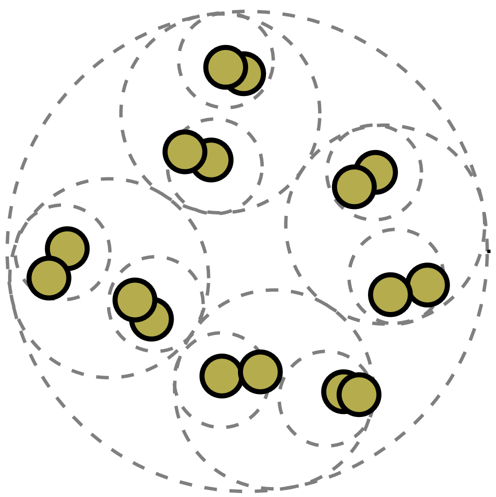
Physics aware tokenization of the input to reduce its dimension

Each solves part of the problem, **but not both**

Existing Solutions

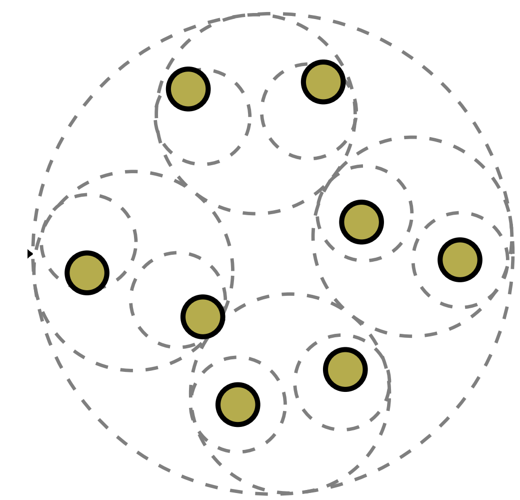
Erwin

Tree Structure



O(N) Complexity

Hierarchical Ball Attention



Erwin

[Point Cloud] → [Ball Groups] → [Hierarchical Processing]

Limitations

- Large-scale Challenges
- Geometric Context Loss
- Memory Management

Advantages

- Linear Scaling
- Efficient Processing
- Multi-scale Analysis

Existing Solutions
Erwin

Physical Aware slices

Million-point Processing

Adaptative Temperature

Transolver++

[Input] → [Slice] → [Attention] → [Deslice] → [Output]

Limitations

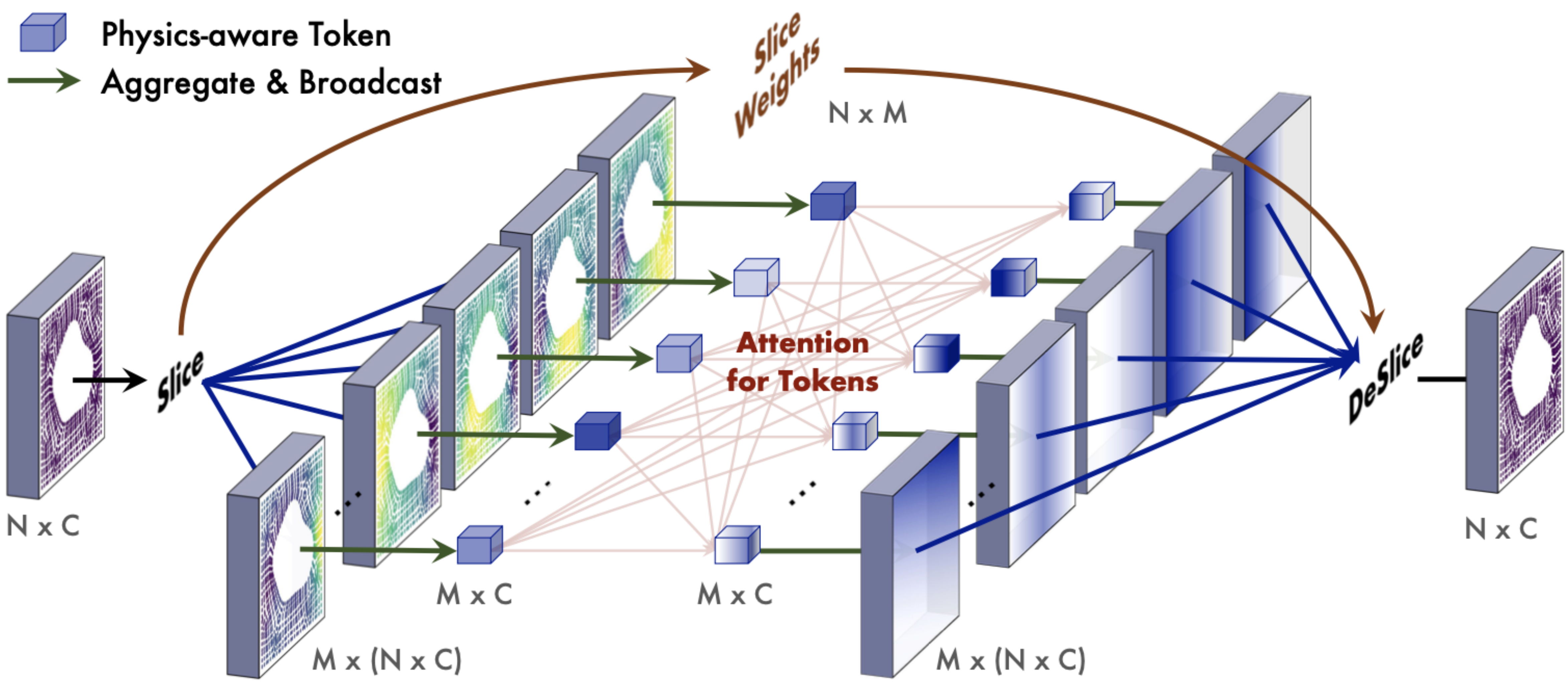
- Quadratic Scaling in attention
- Slice Limitations
- Memory Requirements

Advantages

- Large Scale Processing
- Physical Context
- Adaptive Learning

Existing Solutions

Transolver & Transolver++



Can we design a model that scales to millions of points with fewer bottlenecks, while remaining aware of the underlying physics?

Key Contributions

Key Contributions

Solve the Scaling Problem

ErwinFlash → **Speeding Up Erwin**

HAET* → **Hybrid Architecture**

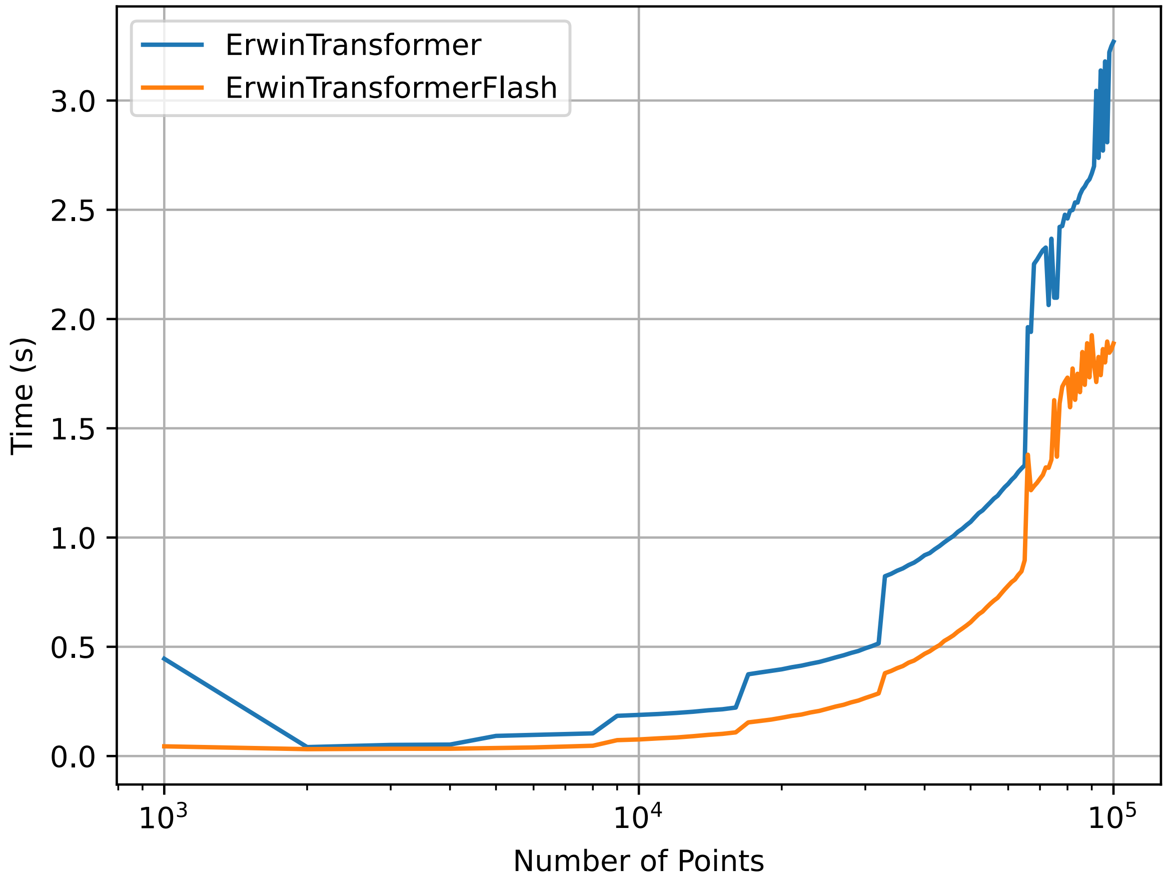
*Hierarchical Attention Erwin Transolver

ErwinFlash

ErwinFlash

Description and Results

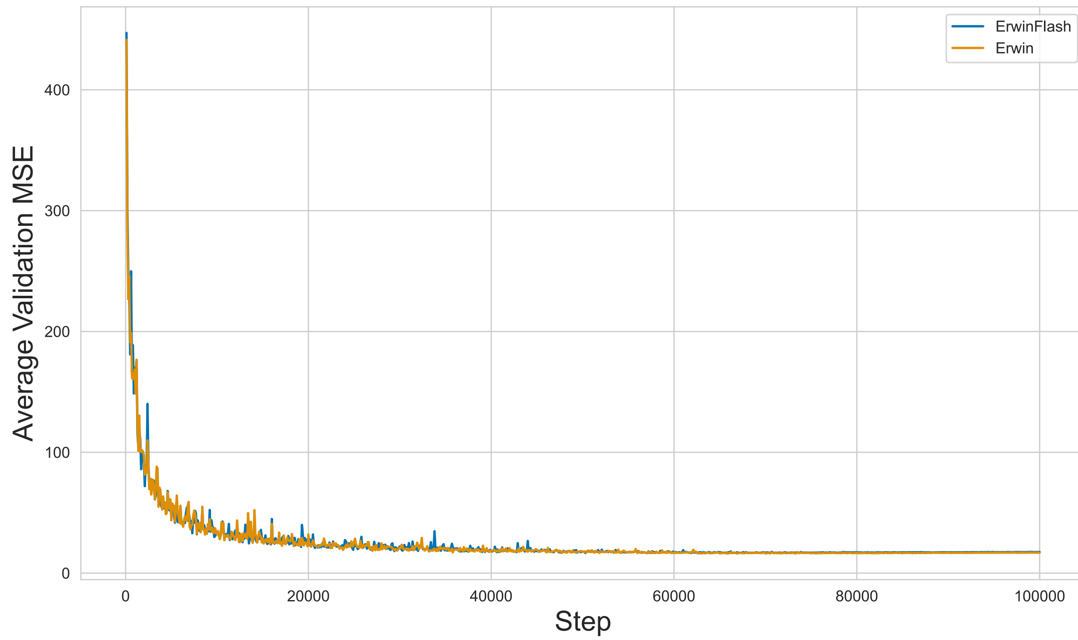
- **Key Optimizations:**
  - FlashAttention
  - Mixed-Precision Training
  - Fused CUDA Operations
- **Improvements:**
  - Speed: 1.9x over Erwin
  - Lower GPU memory usage
  - Identical training/validation loss curves
  - More stable GPU performance

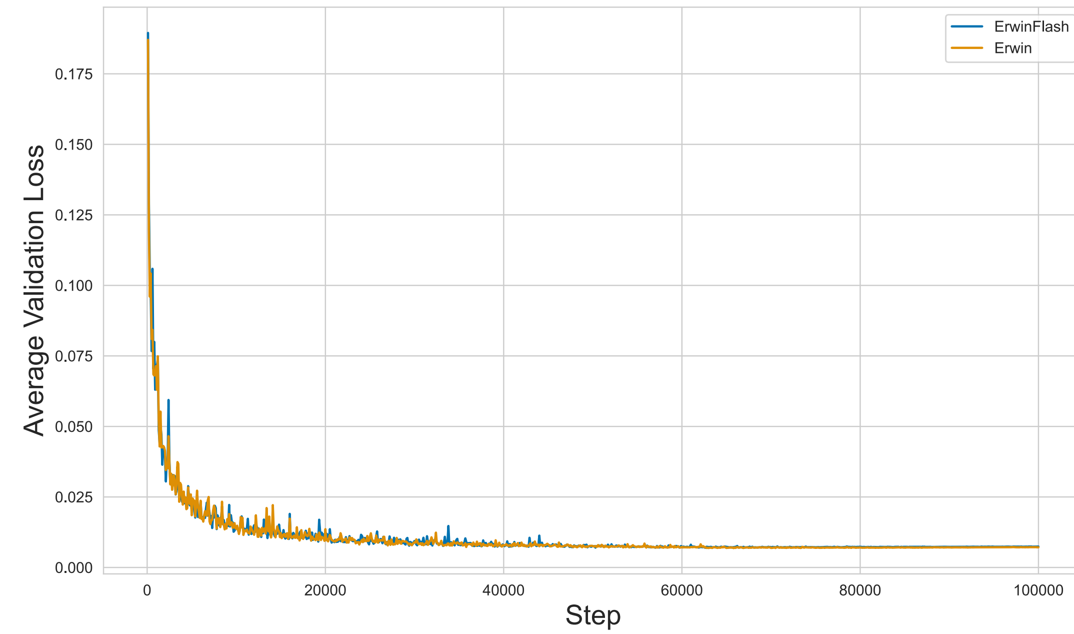


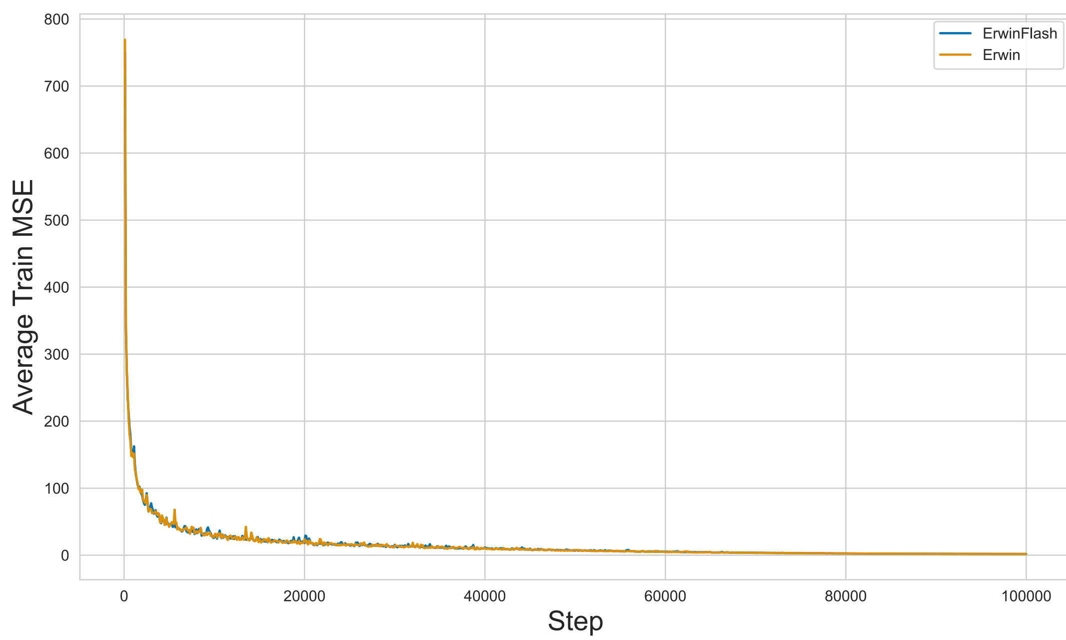
ErwinFlash Speed Improvement Results

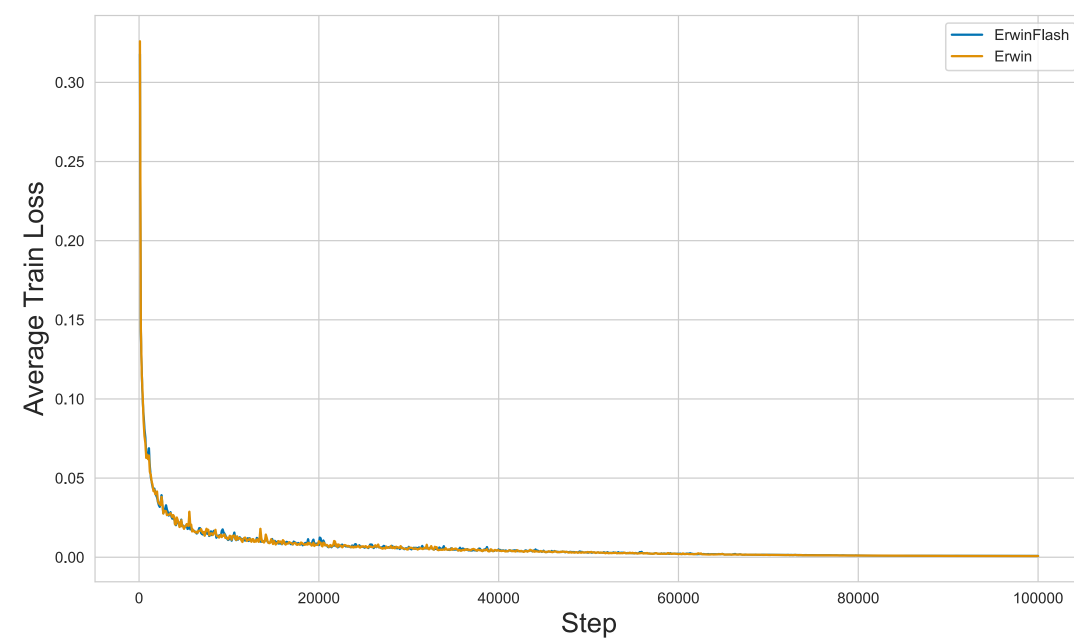
ErwinFlash

ShapeNet Car Results



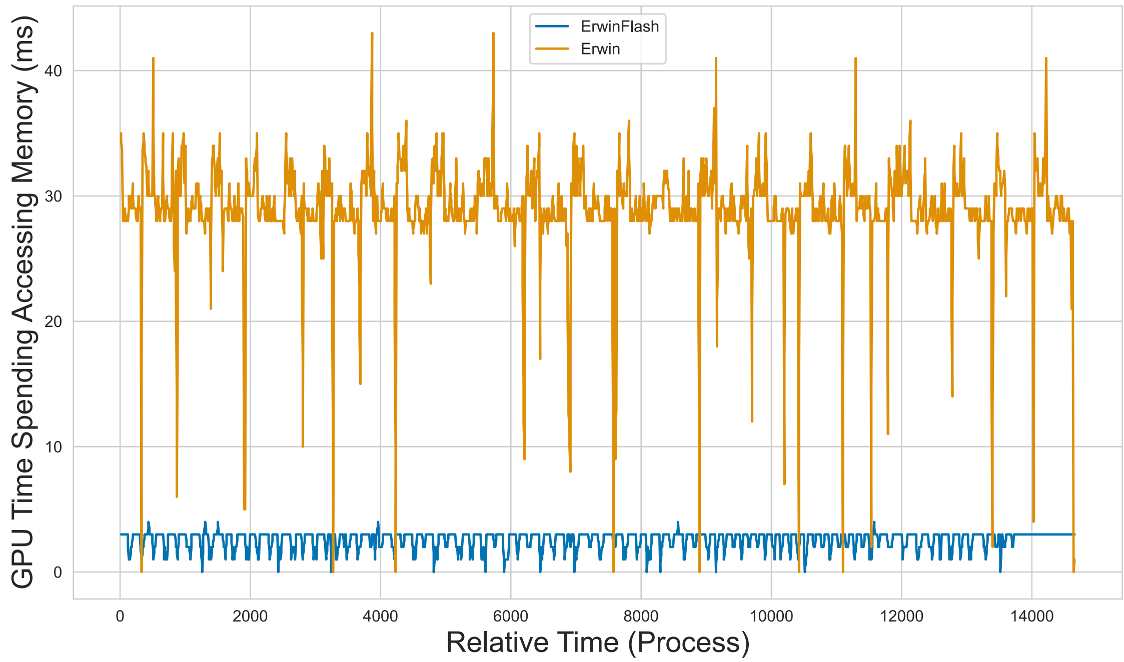


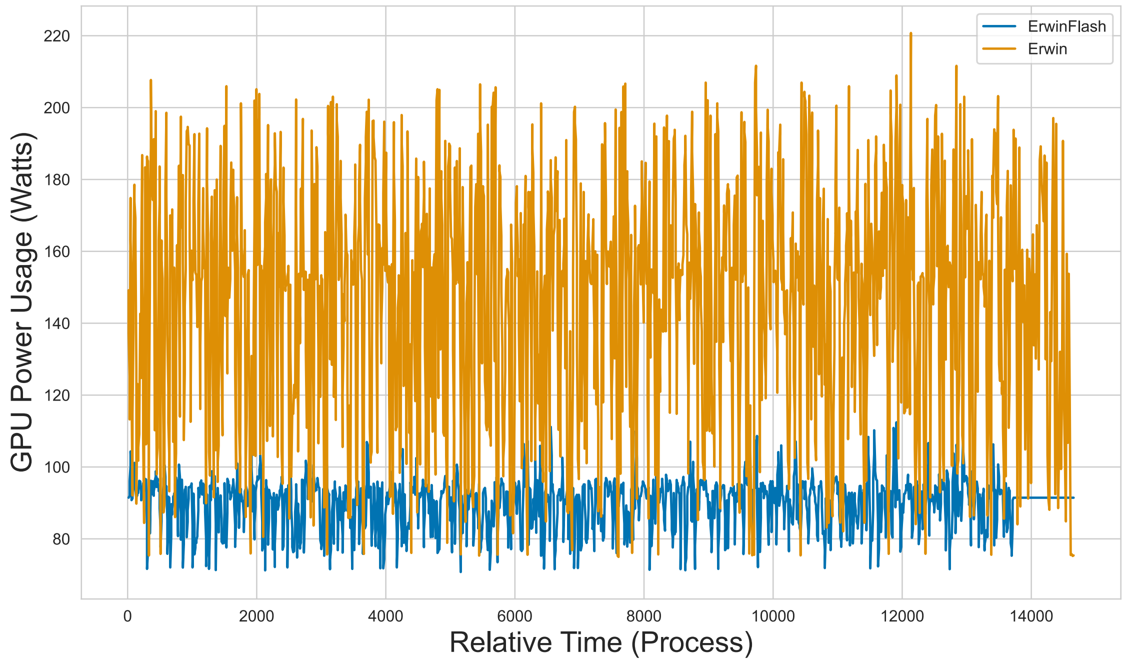


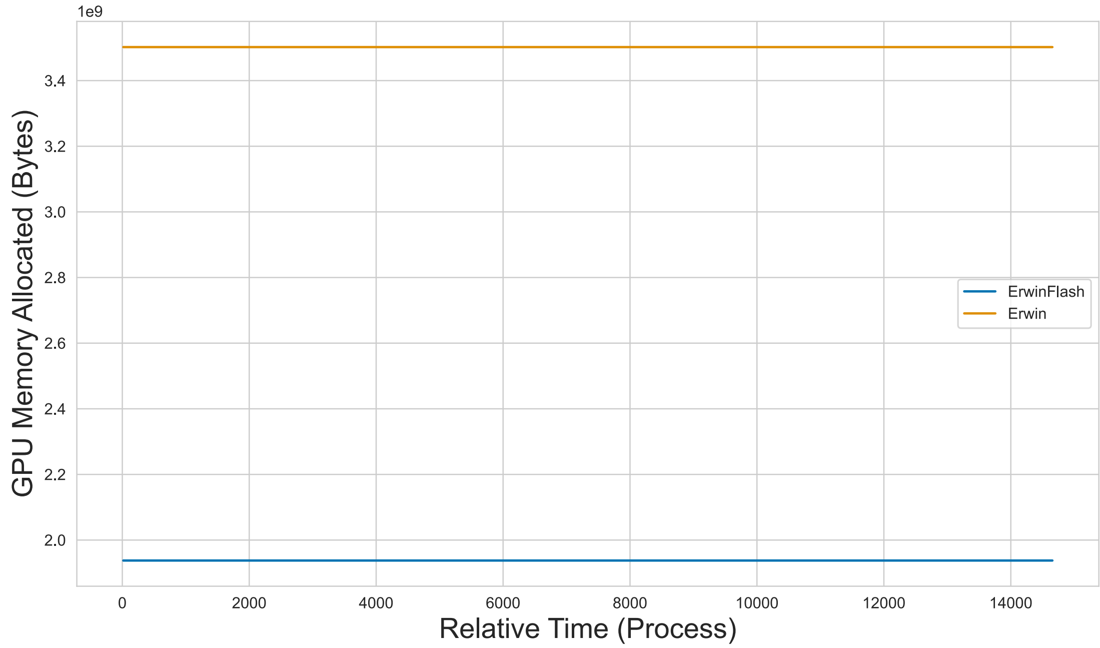


ErwinFlash

ShapeNet Car Results







HAET

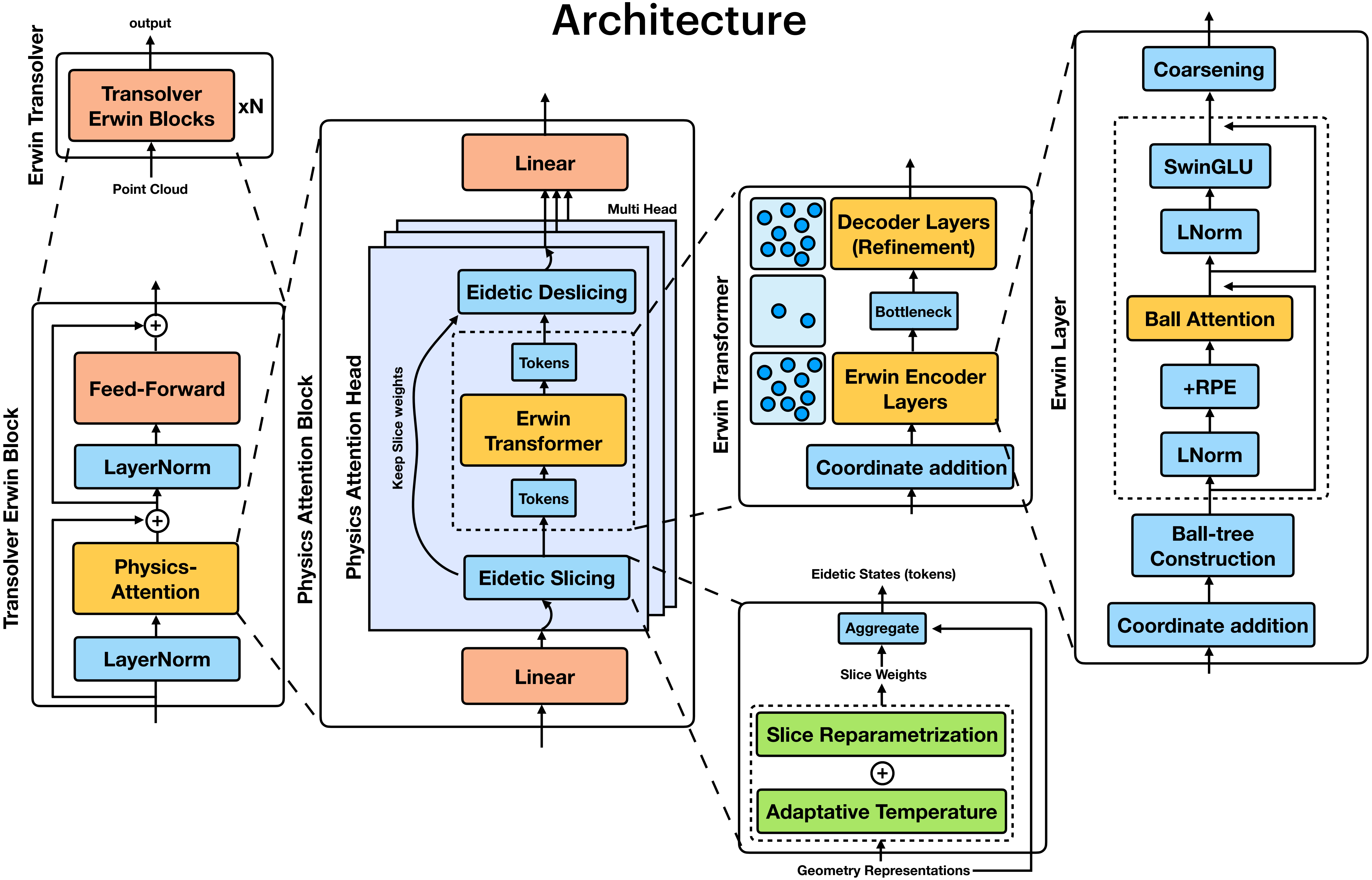
HAET

Hierarchical Attention Erwin Transolver

	Pros	Cons
Erwin	Scales Linearly	Cannot handle millions of input points
Transolver++	Can handle millions of input points “Downsampling” is physics-based	Attention between slices is still $O(N^2)$ Needs small number of slices
HAET	Can handle millions of points Hierarchical Attention between slices scales linearly Number of slices is not a bottleneck	Hierarchical attention within localized groups, i.e., Attention is not done among all slices

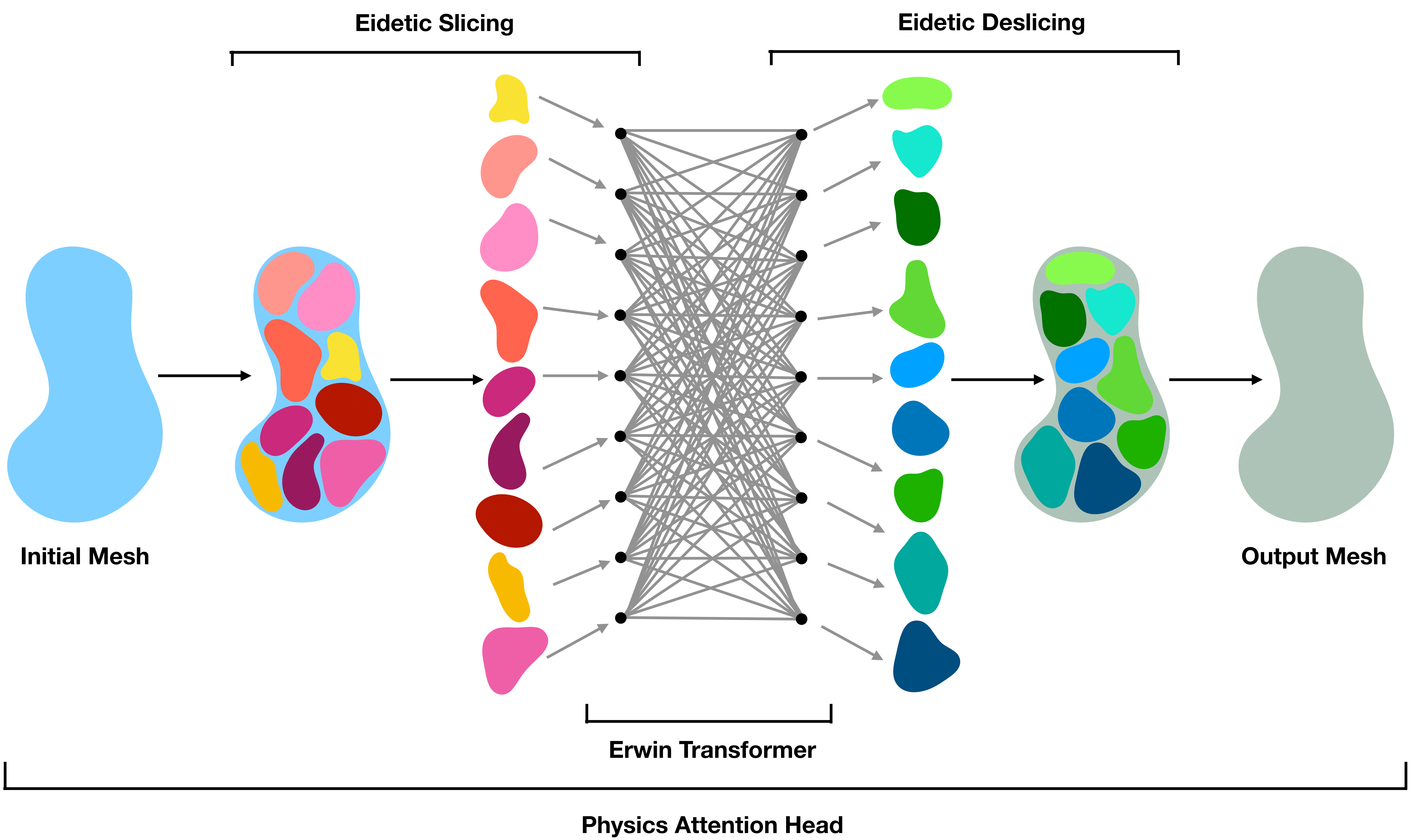
HAET

Architecture



HAET

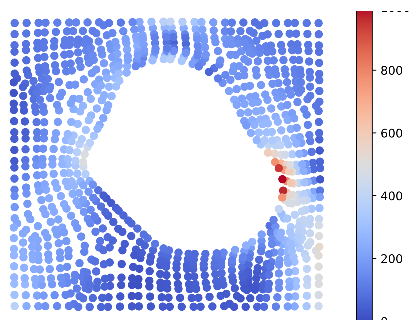
Physics Attention Head



HAET

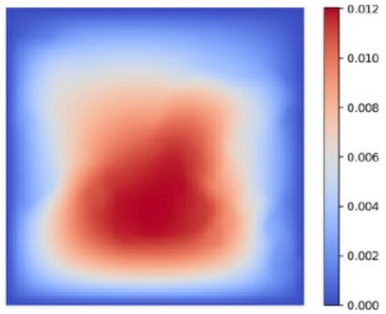
Standard Benchmarks

Elasticity



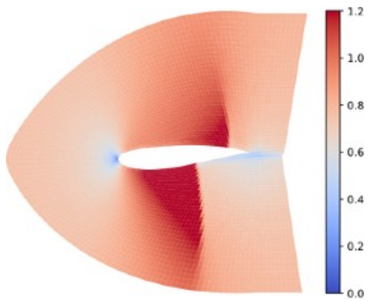
- Point Cloud
- 2D
- 972 Mesh Points

Darcy



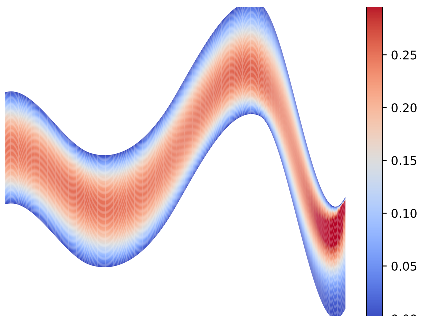
- Regular Grid
- 2D
- 7225 Mesh Points

Airfoil



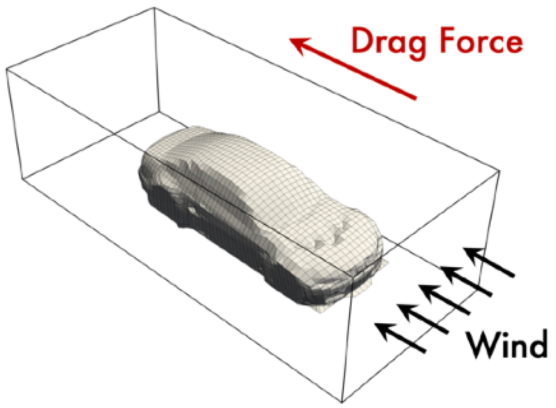
- Structured Mesh
- 2D
- 11271 Mesh

Pipe



- Structured Mesh
- 2D
- 16641 Mesh

ShapeNet Car Design



- Unstructured Mesh
- 3D
- 32186 Mesh Points

HAET

Results

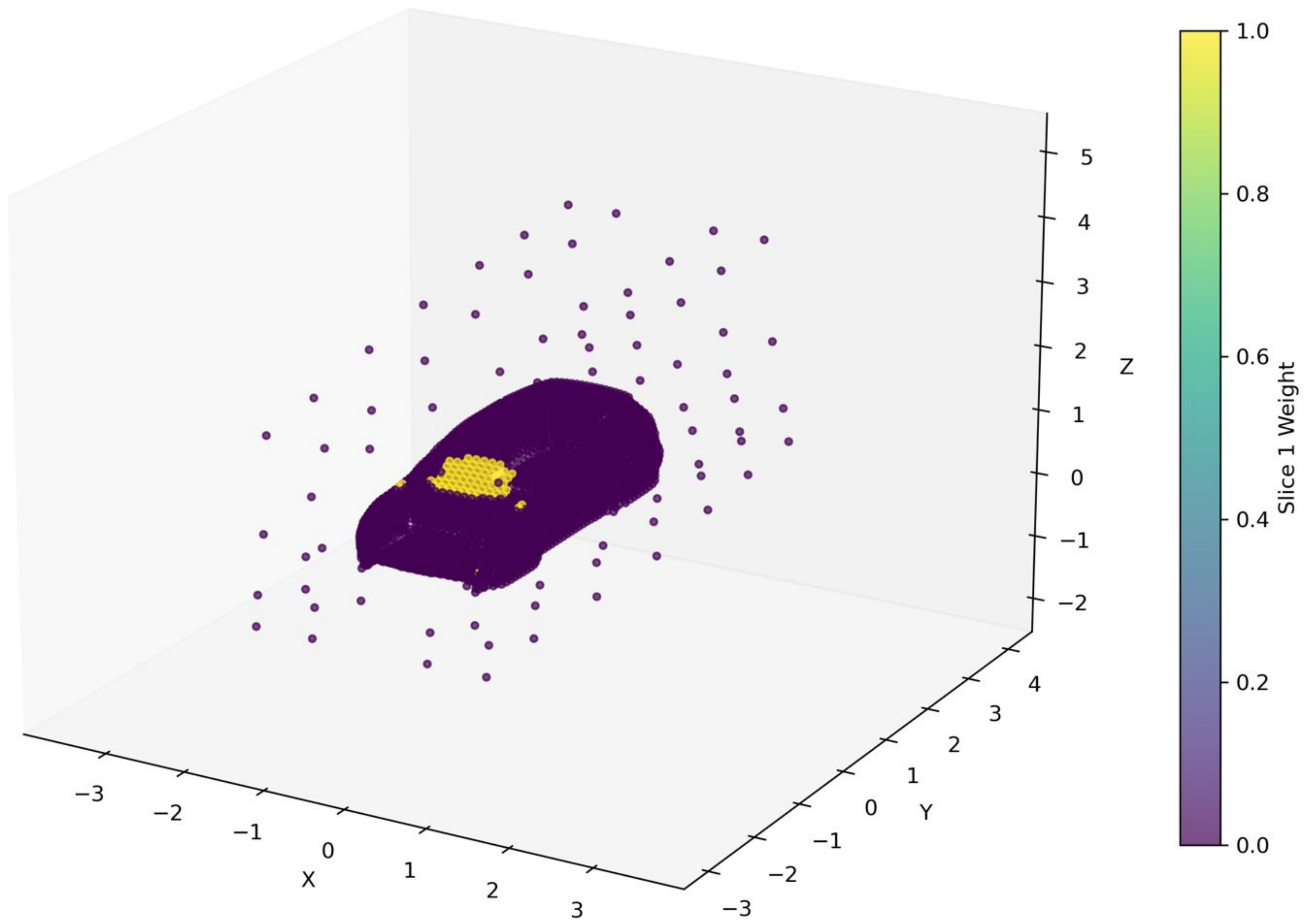
Model	Point Cloud	Structured Mesh			Regular Grid	
	Elasticity	Plasticity	Airfoil	Pipe	Navier–Stokes	Darcy
FNO	/	/	/	/	0.1556	0.0108
WMT	0.0359	0.0076	0.0075	0.0077	0.1541	0.0082
U-FNO	0.0239	0.0039	0.0269	0.0056	0.2231	0.0183
geo-FNO	0.0229	0.0074	0.0138	0.0067	0.1556	0.0108
U-NO	0.0258	0.0034	0.0078	0.0100	0.1713	0.0113
F-FNO	0.0263	0.0047	0.0078	0.0070	0.2322	0.0077
LSM	0.0218	0.0025	<u>0.0059</u>	0.0050	0.1535	0.0065
Galerkin	0.0240	0.0120	0.0118	0.0098	0.1401	0.0084
HT-Net	/	0.0333	0.0065	0.0059	0.1847	0.0079
OFormer	0.0183	<u>0.0017</u>	0.0183	0.0168	0.1705	0.0124
GNOT	<u>0.0086</u>	0.0336	0.0076	<u>0.0047</u>	0.1380	0.0105
FactFormer	/	0.0312	0.0071	0.0060	0.1214	0.0109
ONO	0.0118	0.0048	0.0061	0.0052	<u>0.1195</u>	0.0076
Transolver	**0.0064**	**0.0012**	**0.0053**	**0.0033**	**0.0900**	<u>0.0057</u>
HAET (Ours)	0.108	/	0.0085	0.0050	/	**0.0053**

Model	ShapeNet Car			
	Volume ↓	Surf↓	C_D ↓	ρ_D ↑
Simple MLP	0.0512	0.1304	0.0307	0.9496
GraphSAGE	0.0461	0.1050	0.0270	0.9695
PointNet	0.0494	0.1104	0.0298	0.9583
Graph U-Net	0.0471	0.1102	0.0226	0.9725
MeshGraphNet	0.0354	0.0781	0.0168	0.9840
GNO	0.0383	0.0815	0.0172	0.9834
Galerkin	0.0339	0.0878	0.0179	0.9764
Geo-FNO	0.1670	0.2378	0.0664	0.8280
GNOT	0.0329	0.0798	0.0178	0.9833
GINO	0.0386	0.0810	0.0184	0.9826
3D-GEOCA	0.0319	<u>0.0779</u>	<u>0.0159</u>	0.9842
Transolver	**0.0207**	**0.0745**	**0.0103**	**0.9935**
Erwin	0.0766	0.1335	0.1006	0.7681
HAET (Ours)	<u>0.0257</u>	0.0914	0.0174	<u>0.9865</u>

* PDE Benchmarks metrics: Relative L2

* ShapeNet metrics: Relative L2 of the surrounding (Volume) and surface (Surf) physics fields, drag coefficient, and their Spearman's rank correlations.

HAET
Results



HAET
Computational Efficiency

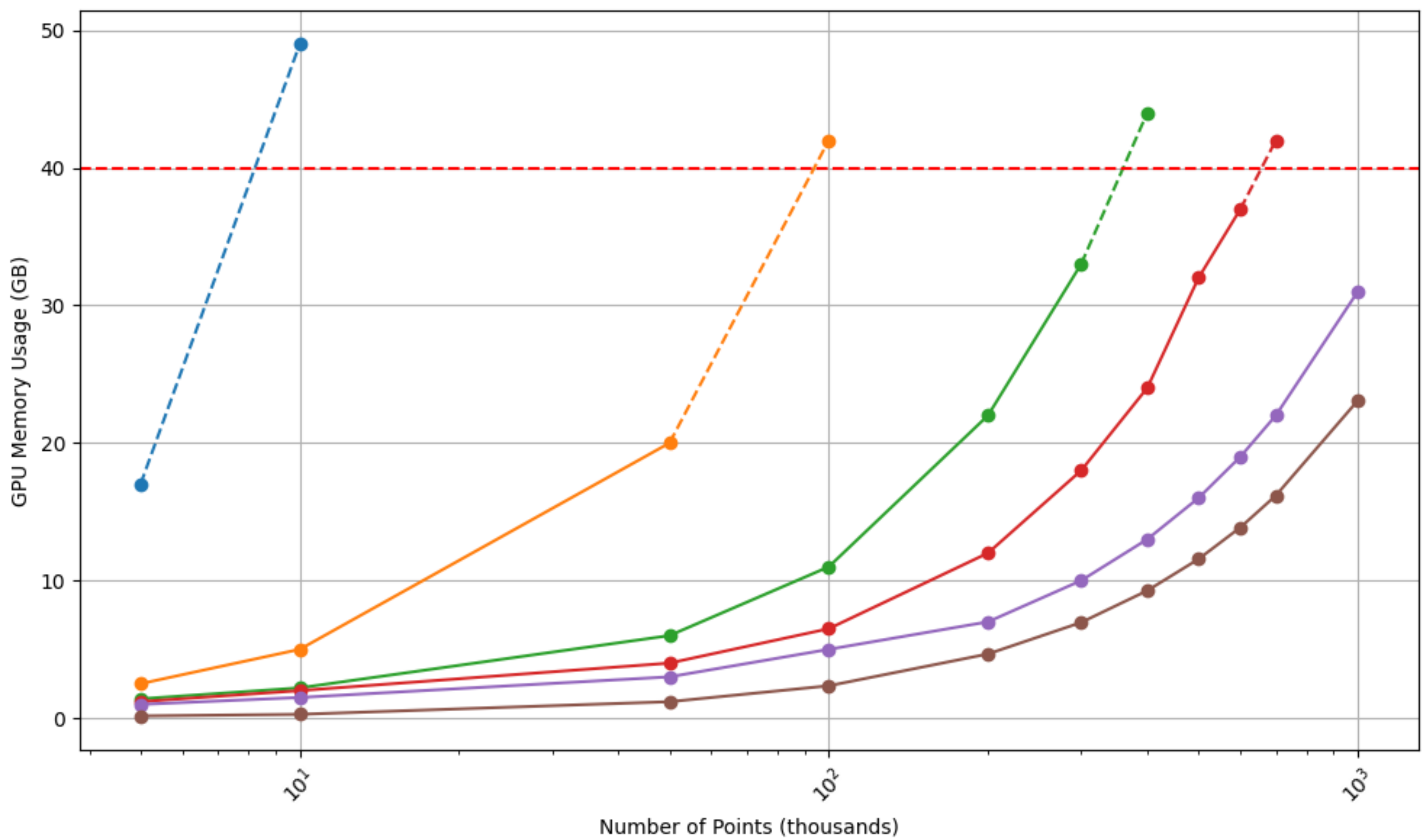


Figure 4: GPU memory usage across models as a function of input size. Models are color-coded as follows: —— **VanillaAttention**, —— **Galerkin**, —— **GNoT**, —— **Transolver**, —— **Transolver++**, —— **HAET**. Transolver++ and HAET demonstrate superior memory efficiency, remaining below the 40 GB threshold up to 1M points.

Conclusion and Future Work

- More efficient: less compute time, less memory
- Performance: close or better than SOTA models

- Future Work
  - Training on larger datasets that would require sharding both the model and the datasets
  - Trying a different task: Classification (e.g. Reduced-8 Dataset)
  - More ambitious: Same concept but using Diffusion instead of Attention

Thank You!

Pedro M. P. Curvo
Mahdi Rahimi
Salvador Torpes

Algorithm 1 Parallel Physics-Attention with Eidetic States

Input: Input features $\mathbf{x}^{(k)} \in \mathbb{R}^{N_k \times C}$ on the k-th GPU.
Output: Updated output features $\mathbf{x}'^{(k)} \in \mathbb{R}^{N_k \times C}$.
// drop $\mathbf{f}$ to save 50% memory.
Compute $\cancel{\mathbf{f}^{(k)}}, \mathbf{x}^{(k)} \leftarrow \text{Project}(\mathbf{x}^{(k)})$
Compute $\tau^{(k)} \leftarrow \tau_0 + \text{Ada-Temp}(\mathbf{x}^{(k)})$
Compute weights $\mathbf{w}^{(k)} \leftarrow \text{Rep-Slice}(\mathbf{x}^{(k)}, \tau^{(k)})$
Compute weights norm $\mathbf{w}_{\text{norm}}^{(k)} \leftarrow \sum_{i=1}^{N_k} \mathbf{w}_i^{(k)}$
Reduce slice norm $\mathbf{w}_{\text{norm}} \leftarrow \text{AllReduce}(\mathbf{w}_{\text{norm}}^{(k)})$ $\mathcal{O}(M)$
Compute eidetic states $\mathbf{s}^{(k)} \leftarrow \frac{\mathbf{w}^{(k)\top}\mathbf{x}^{(k)}\cancel{\mathbf{f}^{(k)}}}{\mathbf{w}_{\text{norm}}}$
Reduce eidetic states $\mathbf{s} \leftarrow \text{AllReduce}(\mathbf{s}^{(k)})$ $\mathcal{O}(MC)$
Update eidetic states $\mathbf{s}' \leftarrow \text{Attention}(\mathbf{s})$
Deslice back to $\mathbf{x}'^{(k)} \leftarrow \text{Deslice}(\mathbf{s}', \mathbf{w}^{(k)})$
Return $\mathbf{x}'^{(k)}$