Multiscale Neural Operators for Solving Time-Independent PDEs

Winfried Ripken[*]
Merantix Momentum

Lisa Coiffard[*]
Merantix Momentum

Felix Pieper[*]
Merantix Momentum

Sebastian Dziadzio
Tübingen AI Center

Abstract

Time-independent Partial Differential Equations (PDEs) on large meshes pose significant challenges for data-driven neural PDE solvers. We introduce a novel graph rewiring technique to tackle some of these challenges, such as aggregating information across scales and on irregular meshes. Our proposed approach bridges distant nodes, enhancing the global interaction capabilities of GNNs. Our experiments on three datasets reveal that GNN-based methods set new performance standards for time-independent PDEs on irregular meshes. Finally, we show that our graph rewiring strategy boosts the performance of baseline methods, achieving state-of-the-art results in one of the tasks.

1 Introduction

Partial Differential Equations (PDEs) describe a wide range of physical systems, particularly in thermodynamics, electromagnetics, or fluid dynamics [6]. Traditional numerical solvers find functions that satisfy PDEs on a discretized solution space, i.e. using reference points on a mesh. Data-driven neural PDE solvers [2] reduce the computational overhead for obtaining solutions drastically, while generalizing to unseen boundary conditions and parameters [18, 17].

Neural operators face unique challenges when solving time-independent PDEs. As opposed to time-dependent PDEs, for which solutions are rolled out incrementally over time [4], the former predict solutions in a single step. Popular approaches to the time-independent setting include Fourier operators [18] and convolutional U-Nets [13, 24]. However, these methods process information on a structured grid, which in many real-world scenarios fails to capture the different levels of precision that may be required over the spatial domain. We illustrate this in Figure 1, where areas around the permanent magnets of the motor require higher mesh resolutions.

In this work, we evaluate graph neural networks (GNNs) and transformers on existing PDE meshes. As the former capture only local interactions, we introduce a novel method for rewiring graphs to connect distant nodes (Section 3)[1]. Our contributions can be summarized as follows:

- We benchmark existing GNN- and transformer-based methods on three challenging datasets: Darcy Flow [25], magnetic simulations of a synchronous electric motor [3] and magnetostatics simulations of current-carrying conductors [22].

- We propose a graph rewiring strategy that reaches state-of-the-art performance.

- We propose a novel architecture that combines transformers with message passing, improving transformer-only results.

---

[*]Equal contribution
[1]Our source code is publicly available: https://github.com/merantix-momentum/multiscale-pde-operators

(a) Node density via a Gaussian kernel-density estimate.

(b) Discretized magnetic field (about 30K nodes).

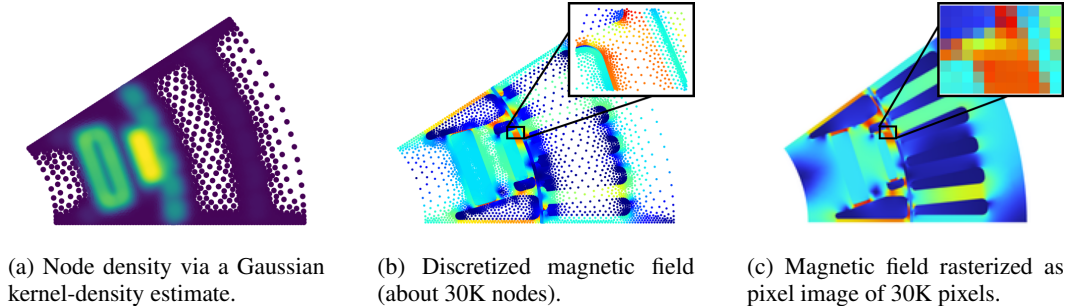(c) Magnetic field rasterized as pixel image of 30K pixels.

Figure 1: For changing mesh resolutions, rasterization either fails to capture local information or requires a vast number of pixels, increasing computational complexity. We show an example from the Motor dataset introduced by Botache et al. [3].

2 Related Work

Neural networks for processing graph data have gained traction recently under the term *geometric deep learning* [5]. Gilmer et al. [11] coin the term *Message Passing Neural Networks* (MPNNs) to denote computing messages as a form of information exchange between graph nodes.

GNN-based models. Pfaff et al. [23] introduce Mesh Graph Nets (**MGN**) to model physical systems by performing message passing on meshes. Their method consists of an encode-process-decode architecture: within the encoder, relative distances between nodes are added as edge attributes, to achieve spatial equivariance.

Hierarchical GNN models. Several works consider the limitations of GNNs in propagating information across long ranges within the graph. Dwivedi et al. [9]. Li et al. [17] are among the first to propose a hierarchical GNN operator for neural PDE solving. They create meshes of varying resolutions by randomly sub-sampling nodes, going from dense to coarse resolutions, and then back.

Similarly, Gao and Ji [10] propose a U-Net-based architecture (g-UNet), with adaptive pooling and unpooling operations based on a learnable transformation of node features. A major limitation of this approach is the high probability of ending up with isolated nodes in down-sampled graphs.

Cao et al. [7] propose the Bi-stride Multi-Scale GNN (**BSMS**): they conduct a breadth-first search (BFS) on the graph at each stage and pool every second front of the BFS result. To provably leave no isolated partitions, they perform adjacency matrix enhancement, i.e. adding edges to the graph by using the second power of the adjacency matrix before every pooling step.

Graph rewiring methods. Another family of methods investigates increasing graph connectivity between distant nodes. Gutteridge et al. [14] use a layer-dependent rewiring approach, such that edges are added over a maximum of k-hop neighbors at a distance $l + 1$ for each layer l.

Transformer-based methods can be considered a *global* form of rewiring, where every node is connected to others in the graph via attention. **Perceiver IO** [15] relies on cross-attention, similarly to other frameworks proposed for PDE solving [19].

3 Method: Creation of Hierarchical Tree Edges

We hypothesize that processing information across different scales is required to effectively solve time-independent PDEs. This can be achieved in various ways, e.g. by combining full-scale attention with local processing, or in a hierarchical manner, as in g-UNets (Section 2). To empirically validate our claim, we design a novel graph rewiring strategy that augments any message passing method with the capability to process information across scales. Most notably, our procedure for adding *hierarchical edges* to a graph depends only on node positions and is very lightweight, as it directly supplements edges to a graph instead of creating multiple graphs for different scales.
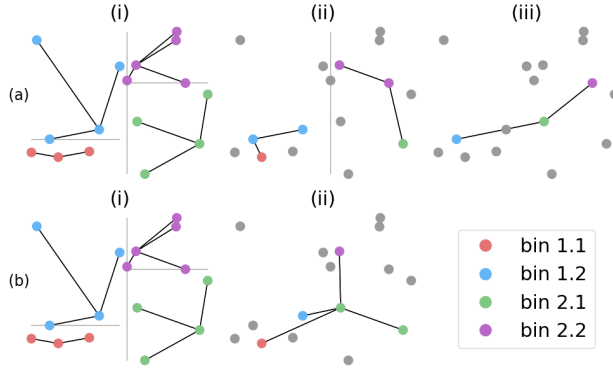
Figure 2: The construction of a binary tree for adding hierarchical edges in a graph, merging either (a) 2 bins, or (b) 4 bins at each level. Gray lines indicate split values; black lines indicate edges that are added to the computation graph. (a,b)(i) Creation of the binary tree: split along the y-axis to create two bins, then split these further along the x-axis. Connect all nodes in a bin to their respective center nodes. (a)(ii) Merge bins 1.1 and 1.2 into bin 1, bins 2.1 and 2.2 into bin 2, and add edges between the center nodes of the bins. (a)(iii) Merge remaining bins 1 and 2 and add edges between center nodes. (b)(ii) Merge all four bins and connect center nodes.

Creating *hierarchical tree edges* (see Appendix A for more details) involves recursively splitting graph nodes into bins according to their positions, over multiple levels (Figure 2). Our approach can be understood as a form of binary space partitioning [26] for graphs. Edges are added between the `center node` of each bin and that of its child bins. A `center node` is the node closest to the mean node position in a bin. The center nodes of the leaf bins are directly connected to all other nodes within the same bin (Figure 2(a,b)(i)). We introduce a separate hyperparameter that can skip levels in the binary tree to control the range of processing scales (Figure 2), similarly to choosing the size of the receptive field in a U-Net.

Our rewiring technique can be combined with any GNN. While it is similar to BSMS in creating coarser graph representations to encourage the exchange of information over long distances, our method, contrary to BSMS, does not require pooling or up-sampling operations.

4   Experiments

For our evaluation, we use three datasets: Darcy Flow, Motor, and Magnetostatics simulations (see Table 1, 2). For Darcy Flow, we further include the results by Takamoto et al. [25] for reference. For the Magnetostatics experiments, we include results from Li et al. [19] (a transformer-based method) and Lötzsch et al. [22] (a GNN method based on spectral convolutions).

Model	Mag. Shape	Magn. Sup.
ChebConv [8]	6×10^{-06}	1.72×10^{-04}
OFormer [19]	1.6×10^{-05}	-
MGN [23]	$2.40 \times 10^{-06} \ (1.12 \times 10^{-06})$	$8.88 \times 10^{-04} \ (4.09 \times 10^{-04})$
MGN + Tree (ours)	$\mathbf{1.31 \times 10^{-06}} \ (\mathbf{1.46 \times 10^{-07}})$	$3.31 \times 10^{-04} \ (3.55 \times 10^{-05})$
BSMS (2×MP) [10]	$4.25 \times 10^{-06} \ (4.27 \times 10^{-07})$	$\mathbf{2.27 \times 10^{-04}} \ (\mathbf{1.53 \times 10^{-05}})$
Perceiver IO [15]	$8.08 \times 10^{-05} \ (4.69 \times 10^{-05})$	$1.08 \times 10^{-03} \ (7.99 \times 10^{-04})$
Perceiver IO + GNN	$3.52 \times 10^{-06} \ (1.55 \times 10^{-06})$	$2.73 \times 10^{-04} \ (1.26 \times 10^{-04})$

Table 1: Test MSE for baselines and custom methods (MGN + Tree, Perceiver IO + GNN) on the Magnetostatics dataset. We average over 3 runs, standard deviation in brackets. Results for ChebConv [8] from Lötzsch et al. [22] and OFormer from Li et al. [19]. Best results in bold.

Our evaluations on Darcy Flow (Table 2) show that both GNN- and transformer-based methods achieve results that are in the same order of magnitude as convolution U-Nets and the Fourier

Model	Motor	Darcy Flow
U-Net [24]	$-$	$\mathbf{6.4 \times 10^{-03}}$
FNO [18]	$-$	1.2×10^{-02}
MGN [23]	$9.36 \times 10^{-02}\ (1.32 \times 10^{-02})$	$6.41 \times 10^{-02}\ (5.23 \times 10^{-04})$
MGN + Tree (ours)	$7.55 \times 10^{-02}\ (5.04 \times 10^{-03})$	$4.13 \times 10^{-02}\ (1.47 \times 10^{-02})$
BSMS (2×MP) [10]	$\mathbf{6.91 \times 10^{-02}}\ (\mathbf{2.78 \times 10^{-03}})$	$\mathbf{9.83 \times 10^{-03}}\ (\mathbf{8.18 \times 10^{-04}})$
Perceiver IO [15]	$1.67 \times 10^{-01}\ (2.18 \times 10^{-02})$	$1.41 \times 10^{-02}\ (1.16 \times 10^{-03})$
Perceiver IO + GNN	$9.22 \times 10^{-02}\ (8.01 \times 10^{-03})$	$1.31 \times 10^{-02}\ (2.19 \times 10^{-04})$

Table 2: Test RMSE of our baselines and custom methods (MGN + Tree, Perceiver IO + GNN) on the Motor and Darcy Flow datasets. All metrics are averaged over 3 runs with standard deviation in brackets. Results for U-Net and FNO from Takamoto et al. [25]. Best results are in bold; for Darcy Flow, we highlight both the best-performing method for regular grids (U-Net) and arbitrary grids (BSMS).

Neural Operator (FNO). Note that, as all other datasets have non-uniform grids, we cannot apply convolution-based methods nor the FNO, underlining the necessity for methods that can process arbitrary meshes.

Hierarchical processing. We evaluate our rewiring approach by augmenting the input mesh of MGN with hierarchical tree edges (MGN+Tree) as described in Section 3. Experiments show that adding our simple rewiring procedure improves upon MGN across all datasets. For the Magnetostatics Shape Generalization task, it achieves SOTA results, improving on the second-best performing method by a factor of 5 (Table 1).

For 3 out of 4 tasks, BSMS with 2 layers of message passing per scale achieves the best or second-best results, which supports our claim that hierarchical message passing is key to solving time-independent PDEs. The added advantage of introducing message passing layers at each level of the hierarchical operator in BSMS becomes significant for large graphs, such as the Darcy Flow dataset.

Comparison to full-scale attention. The results for Perceiver IO on the Darcy Flow dataset (Table 2) show that *global* rewiring via cross-attention can perform almost on par with BSMS. For irregular grids with changing resolutions, both in the Motor and Magnetostatics datasets (Table 2 & 1), Perceiver IO falls behind in performance. We also augment Perceiver IO with message passing layers to capture local variations more accurately (Perceiver + GNN), resulting in substantially better performance. This suggests that the standard Perceiver IO may fail to capture local variations in high-resolution areas.

Further negative results. We found that the graph rewiring method proposed in DRew [14] scales poorly to large graphs. Additionally, we integrated our hierarchical tree edges (MGN+Tree) into a U-Net architecture (inspired by [20]), which did not yield improvements to our presented results.

5 Summary & Future Work

In summary, all our experiments show that hierarchical information processing is key to solving time-independent PDEs on arbitrary meshes. Our results suggest that GNN-based and transformer-based methods can achieve similar performance to previously proposed techniques, which consider regular grids only. We believe that this finding is very relevant for practitioners, as irregular meshes are found frequently in real-world applications.

In the future, we hypothesize that our proposed Perceiver IO + GNN (Section 4) can achieve state-of-the-art results by improving the transformer's ability to distribute global information. Hyperparameter tuning per dataset could allow us to match the reported OFormer [19] results. We also plan to investigate our proposed combination of graph U-Nets and hierarchical tree edges, using a binary tree that adapts to the changing resolution of the mesh, instead of rasterization on a fixed grid. Finally, as extensive aggregation of information over coarser scales leads to oversquashing [1], we will study those effects in the context of hierarchical GNNs.

6   Acknowledgements

References

[1] U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2020.

[2] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[3] D. Botache, J. Decke, W. Ripken, A. Dornipati, F. Götz-Hahn, M. Ayeb, and B. Sick. Enhancing multi-objective optimization through machine learning-supported multiphysics simulation. *arXiv preprint arXiv:2309.13179*, 2023.

[4] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural pde solvers. In *International Conference on Learning Representations*, 2021.

[5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[6] Y. Cao and R. Li. A liquid plug moving in an annular pipe—flow analysis. *Physics of Fluids*, 30(9), 2018.

[7] Y. Cao, M. Chai, M. Li, and C. Jiang. Bi-stride multi-scale graph neural network for mesh-based physical simulation. In *International conference on machine learning*. PMLR, 2023.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[9] V. P. Dwivedi, L. Rampášek, M. Galkin, A. Parviz, G. Wolf, A. T. Luu, and D. Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

[10] H. Gao and S. Ji. Graph u-nets. In *International conference on machine learning*, pages 2083–2092. PMLR, 2019.

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[12] R. J. Gladstone, H. Rahmani, V. Suryakumar, H. Meidani, M. D'Elia, and A. Zareei. Gnn-based physics solver for time-independent pdes. *arXiv preprint arXiv:2303.15681*, 2023.

[13] J. K. Gupta and J. Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[14] B. Gutteridge, X. Dong, M. M. Bronstein, and F. Di Giovanni. Drew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pages 12252–12267. PMLR, 2023.

[15] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2021.

[16] D. P. Kingma and J. L. Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[17] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, and A. Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[18] Z. Li, N. B. Kovachki, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020.

[19] Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

[20] M. Lino, S. Fotiadis, A. A. Bharath, and C. Cantwell. Towards fast simulation of environmental fluid mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2205.02637*, 2022.

[21] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2016.

[22] W. Lötzsch, S. Ohler, and J. Otterbach. Learning the solution operator of boundary value problems using graph neural networks. In *ICML 2022 2nd AI for Science Workshop*, 2022.

[23] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

[24] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[25] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

[26] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 153–162, 1987.

A Details for the Creation of Tree Edges

In this section, we describe our procedure for adding hierarchical edges to the graph in more detail. Similarly to what is described in Gladstone et al. [12] as edge augmentation, we add edges to the computation graph (via rewiring). Instead of adding edges randomly between pairs of nodes, we strive to add them in a structured hierarchical way, such that information can flow more efficiently across the graph.

First, we use the following procedure to split the graph nodes approximately in half, relying on the position information for all graph nodes. Let n^i denote the position vector for node i. We identify the spatial dimension with the largest variance of node positions, i.e. we compute $\arg\max_{d \in \{x,y\}} \text{Var}_i(n_d^i) \to d$. We then split the nodes into two bins at the median of all node positions along that selected dimension: $s = \text{median}_i(n_d^i)$. We denote the node positions in the first bin as $\texttt{bin}_1 \leftarrow \{n_i | n_d^i \geq s\}$ and for the second bin as $\texttt{bin}_2 \leftarrow \{n_i | n_d^i < s\}$. If we apply this procedure recursively k times to create k levels, we end up with a binary tree represented by all split values. Those split values are shown as the gray lines in Figure 2.

We now describe how we construct additional edges for the computation graph from this structure. We define the `center node` as the node in the graph, that is closest to the mean of nodes in a `bin`, i.e. $c(\texttt{bin}) = \arg\min_{n_i \in \texttt{bin}} |n^i - \overline{n}|$ where $\overline{n} = \text{mean}(\texttt{bin})$. Given $\texttt{bin}_1$ and $\texttt{bin}_2$ resulting from a split s, we add edges from node $c(\texttt{bin}_1 \cup \texttt{bin}_2)$ to $c(\texttt{bin}_1)$ and $c(\texttt{bin}_2)$. Additionally, for the leaf

level, all nodes of a bin are connected to the bins' center node. Doing this connects the whole graph hierarchically as displayed in Figure 2.

We also allow skipping over levels in the binary tree and merging 2^{n+1} bins in a single step. For example, skipping every two levels would mean connecting four bins (or nodes) to a single node (Figure 2b) and so on.

For our experiments, we simply supplement all tree edges to the computation graph. Via the edge attributes (relative distances), the message passing operation is able to identify the hierarchical tree edges across larger distances.

B Dataset Details

Darcy Flow. We use the data made publicly available via PDEBench [25] for $\beta = 1$. The task is to predict flow through a porous medium given the viscosity of a fluid.

The Darcy flow dataset contains only uniform grids. It is challenging, because of a high grid resolution (about 16K graph nodes per sample) and therefore requires long-range message passing.

Motor Simulations. We use the data made publicly available by Botache et al. [3] for motor_pos = 1. Given motor topologies i.e. the arrangement of components (magnets, air cavities, coils, iron parts), the task is to predict the magnetic field. Even though the magnetic field inside the synchronous electric motor can change, it only depends on the applied current and the rotating angle, which we both keep constant in our case. We use Delaunay triangulation to obtain the meshes for our GNN-based methods.

The motor dataset has a very high resolution (ca. 30K graph nodes per sample) and highly irregular meshes and therefore is particularly challenging.

Magnetostatics Simulations. We use the data made publicly available by Lötzsch et al. [22]. The task is to predict the magnetic vector potential from a distribution of electric currents. We omit the prediction of the magnetic field. The magnetostatics PDE used is a 2D Poisson equation with Dirichlet boundary conditions.

The magnetostatics dataset contains irregular meshes of 5 different shapes. It is challenging because it explicitly requires the network to generalize (the two test sets are distinct from the training set).

C Hyperparameters + Training Details

For all methods, we use a node-wise MLP encoder and decoder before and after the operator, similar to Pfaff et al. [23]. For all experiments, the inputs are normalized with zero mean and unit standard deviation. All latent dimensions (node or edge embeddings, hidden layers of MLPs) are set to 64 if not specified otherwise.

We use the Adam optimizer [16] with default parameters. For training, we calculate the mean squared error loss over all predicted quantities and use a virtual batch size of 16 for all experiments. The training is conducted on an NVIDIA T4 with 16GB VRAM in a Google Kubernetes cluster. All experiments are repeated 3 times over 200 epochs each. We save checkpoints every 3 epochs and perform testing on the checkpoint with the lowest validation loss (MSE). We use cosine annealing [21] with $\eta_{min} = 0$ as the learning rate schedule.

MGN [23]: For Darcy Flow, we use the relative distances between nodes as edge attributes. As the problem is symmetric, we do not pass any absolute node positions, but rather the distance to the closest border in x- and y-direction. For the Motor data, we use relative distances between nodes as edge attributes and pass absolute positions as node attributes. We use 15 layers of message passing like in the original paper [23] and a latent dimension of 64 everywhere.

MGN + Tree (ours): As edge attributes are initialized as relative distances, the hierarchical edges we add in Section 3 can be differentiated from the rest. For both the Motor dataset and Darcy Flow, we split 10 times and connect 8 bins to one (skipping 2 levels for constructing the added edges). For

magnetostatics experiments, we split 4 times and connect all levels directly (2 bins are merged into one).

BSMS [7]: For BSMS, we use the relative distances between nodes as edge attributes, as proposed in the original paper [7]. Similar to before, we pass the distance to the border for Darcy Flow and absolute positions for the Motor dataset as node attributes. We downsample the graph 5 times using breadth-first search and use one or two layers of message passing for each scale.

Perceiver IO [15]: We always pass absolute positions as node attributes and, for Darcy Flow, additionally the distance to the border. If we have to batch graphs of different sizes together, we zero-pad the node attributes. We re-use the input vector as the query vector for the last cross-attention layer, which has only been transformed by the very first node-wise MLP encoder. We use 128 latent tokens. We also add a skip connection between the MLP encoder and decoder, essentially bridging the Perceiver IO model together and enabling local information to flow through. This greatly enhances the performance of Perceiver IO, especially for the Motor dataset.

Perceiver IO + GNN: We add 5 layers of message passing both before and after the Perceiver IO model. The output of the first message passing block is added to the output of the Perceiver IO model before feeding it to the second message passing block, adding a skip connection between the two blocks.

D   Additional Results

In Table 3, we record the training time in hours for all of our methods. In Table 4 we compare the performance of BSMS with 2 layers against only one layer of message passing. We observe an improvement for most of the datasets and therefore report our main results using 2 layers of message passing. As improvements are relatively minor, this parameter does not seem to be crucial for the success of BSMS. In Figure 3, Figure 4, Figure 5 and Figure 6, we visualize exemplary predictions for all methods and datasets.

Model	Motor	Darcy Flow	Magnetostatics
MGN	18.95	141.00	2.65
MGN + Tree	25.91	170.16	3.04
BSMS	22.92	161.81	**2.41**
Perceiver IO	**3.10**	**17.46**	5.25
Perceiver IO + GNN	15.46	104.18	6.27

Table 3: Average training time in hours, for our baselines and custom methods (MGN + Tree edges, Perceiver IO + GNN). All metrics are averaged over 3 runs on an Nvidia Tesla T4 GPU. For both magnetostatics tasks, we train only once, as only the test set changes. Perceiver IO is faster than GNN-based methods for datasets with large graphs, as we do not adapt the hyperparameters (e.g. number of latents).

Model	Motor	Darcy Flow	Mag. Shape	Magn. Sup.
BSMS (2xMP)	$\mathbf{6.91 \times 10^{-02}}$	9.83×10^{-03}	$\mathbf{1.84 \times 10^{-03}}$	$\mathbf{1.08 \times 10^{-02}}$
BSMS (1xMP)	7.29×10^{-02}	$\mathbf{9.48 \times 10^{-03}}$	2.38×10^{-03}	1.24×10^{-02}

Table 4: Ablation study of BSMS' number of message passing layers per scale. We report the mean RMSE scores averaged over 3 runs each.
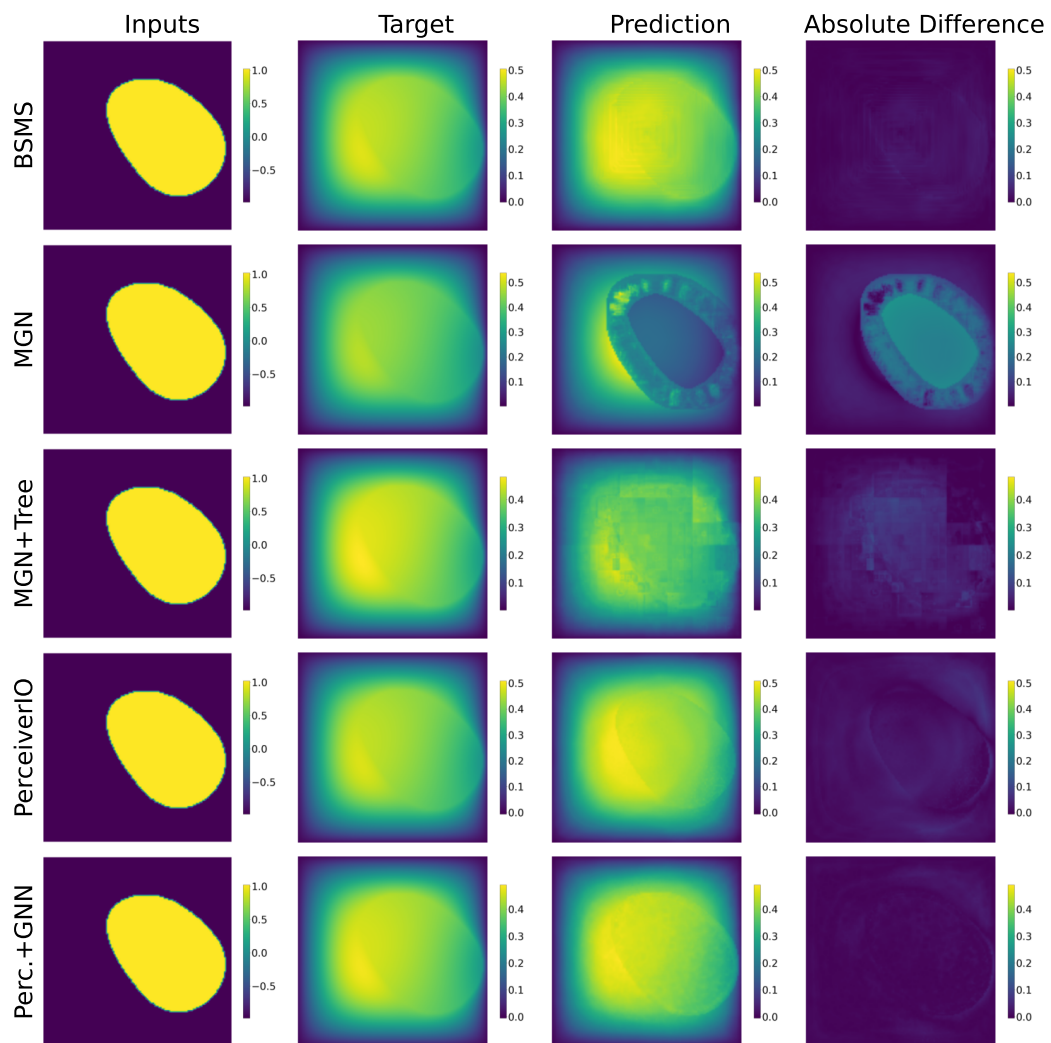
Figure 3: Example predictions for the Darcy Flow dataset [25]
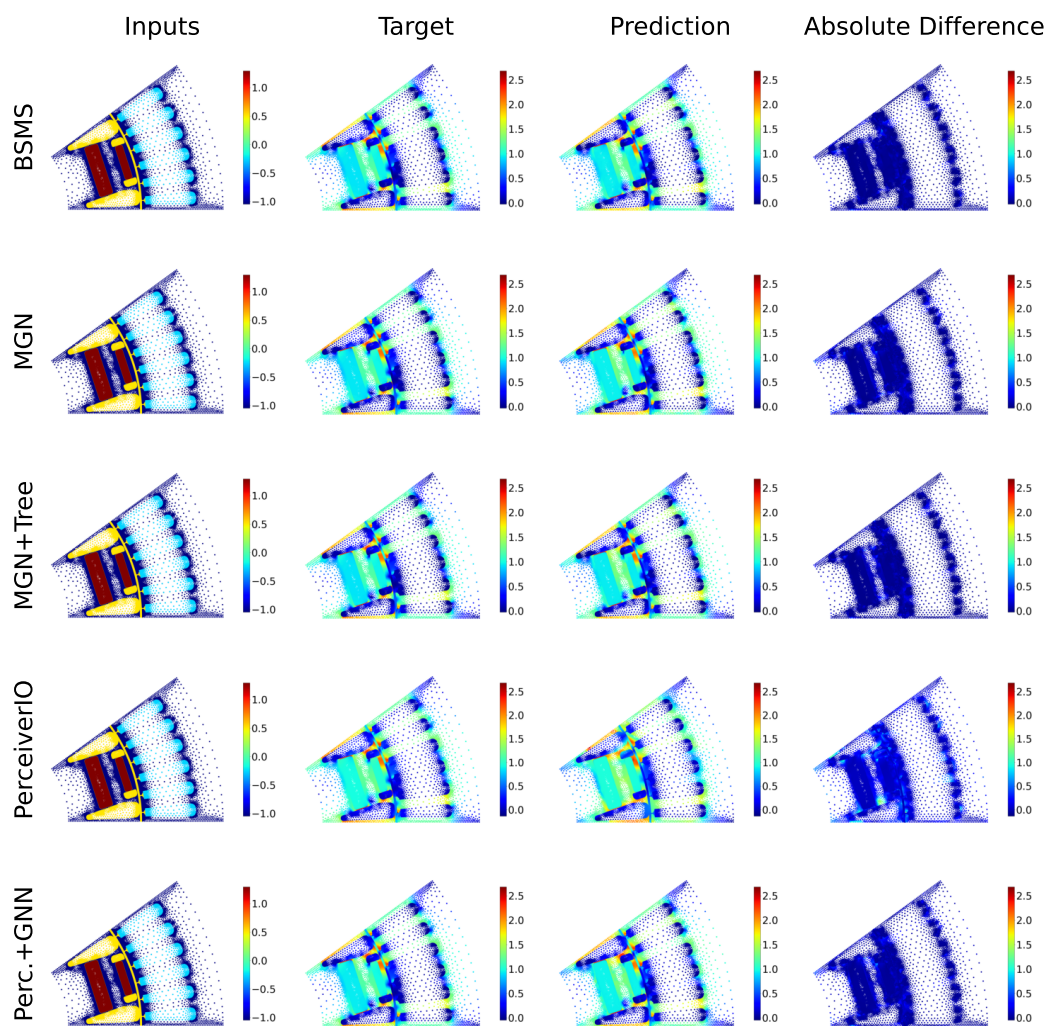
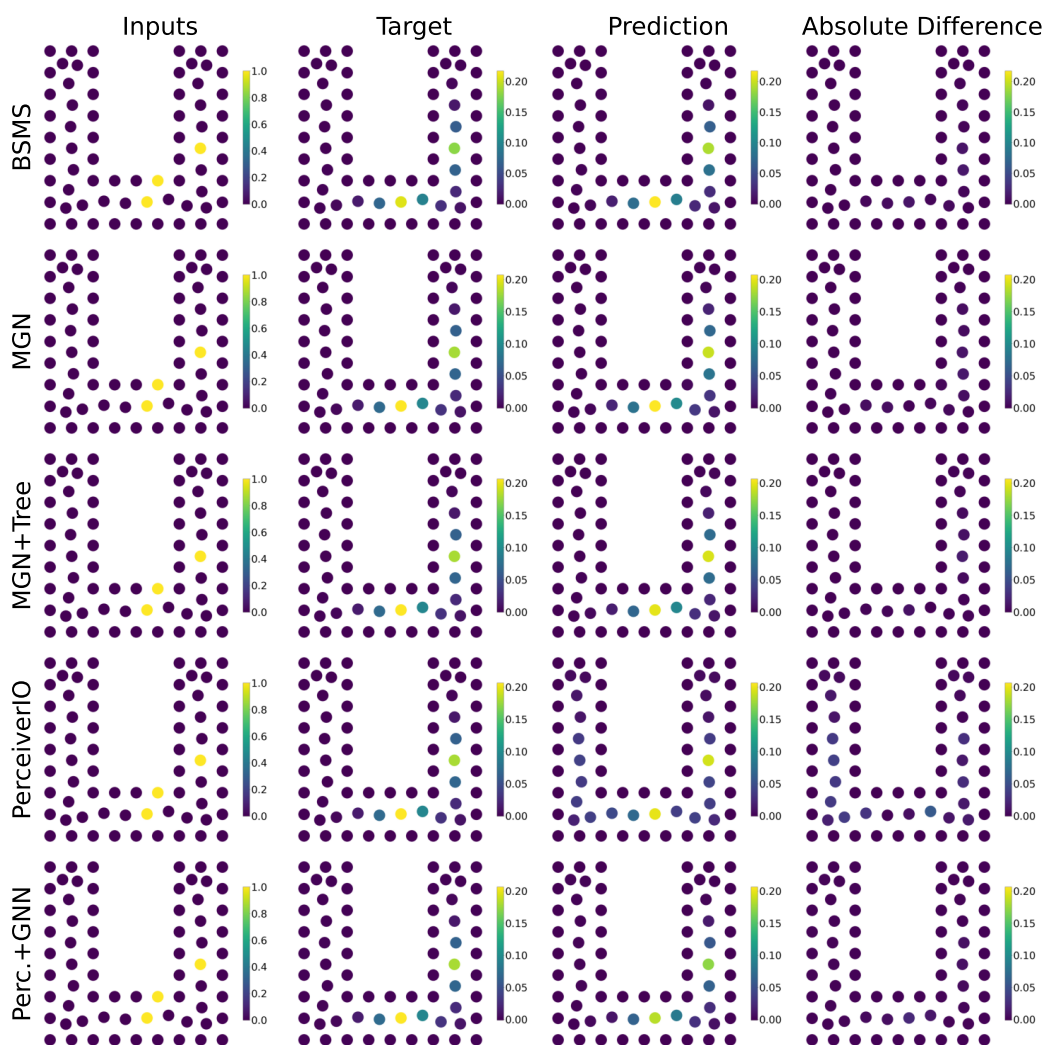Figure 4: Example predictions for the electric Motor dataset [3].

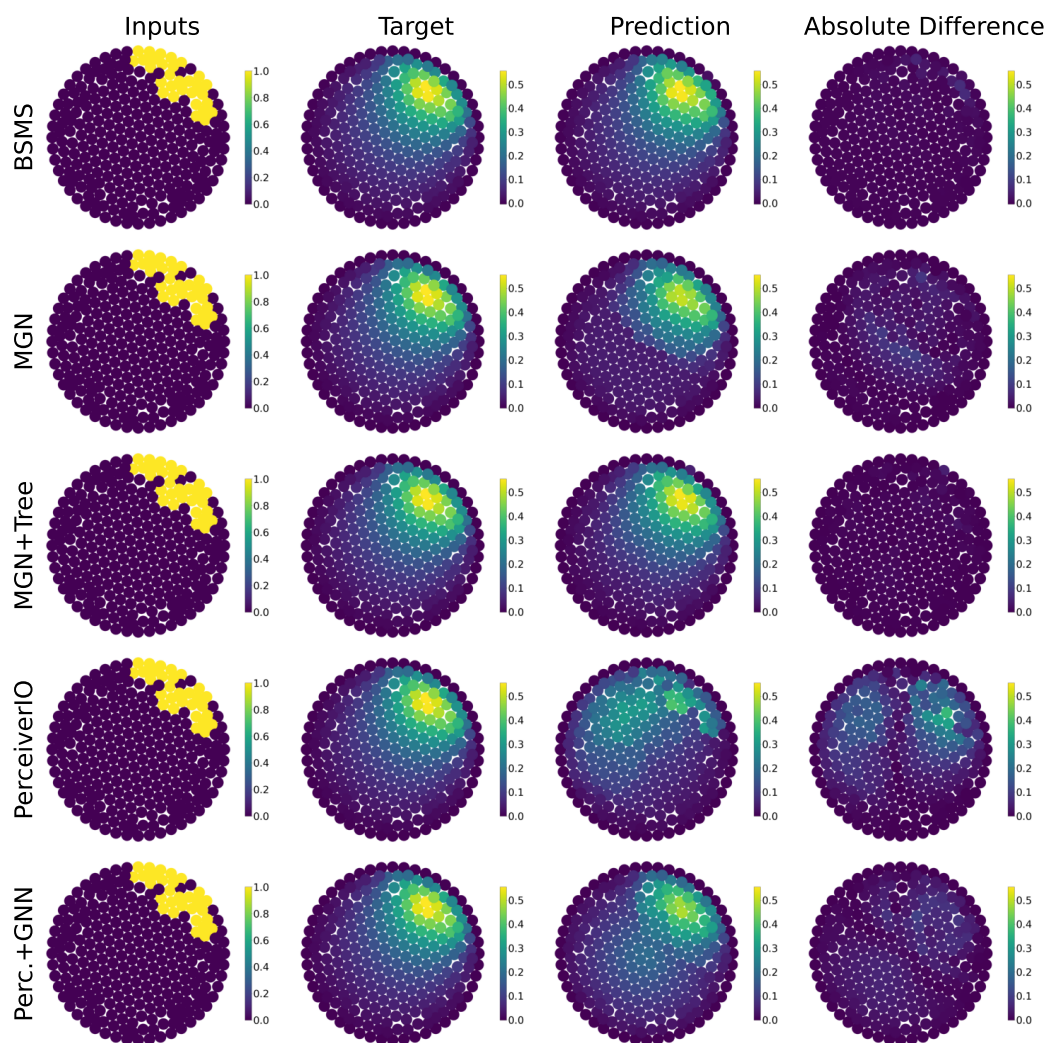Figure 5: Example predictions for magnetostatics data on the shape generalization test set [22].

Figure 6: Example predictions for magnetostatics data on the superpositions test set [22].