MAgNet: Mesh Agnostic Neural PDE Solver

Anonymous Author(s)
Affiliation
Address
email

Abstract

The computational complexity of classical numerical methods for solving Partial Differential Equations (PDE) scales significantly as the resolution increases. When it comes to climate predictions, fine spatio-temporal resolutions are required to resolve all turbulent scales in the fluid simulations. This makes the task of accurately resolving these scales computationally out of reach even with modern supercomputers. As a result, climate modelers solve these PDEs on grids that are too coarse (3km to 200km on each side), which hinders the accuracy and usefulness of the predictions. In this paper, we leverage the recent advances in Implicit Neural Representations (INR) to design a novel architecture that predicts the spatially continuous solution of a PDE given a spatial position query. By augmenting coordinate-based architectures with Graph Neural Networks (GNN), we enable zero-shot generalization to new non-uniform meshes and long-term predictions up to 250 frames ahead that are physically consistent. Our Mesh Agnostic Neural PDE Solver (MAgNet) is able to make accurate predictions across a variety of PDE simulation datasets and compares favorably with existing baselines. Moreover, MAgNet generalizes well to different meshes and resolutions up to four times those trained.

1 Introduction

Partial Differential Equations (PDEs) describe the continuous evolution of multiple variables, e.g. over time and/or space. They arise everywhere in physics, from quantum mechanics to heat transfer and have several engineering applications in fluid and solid mechanics. However, most PDEs can't be solved analytically, so it is necessary to resort to numerical methods. Since the introduction of computers, many numerical approximations were implemented, and new fields emerged such as Computational Fluid Mechanics (CFD) (Richardson and Lynch, 2007). The most famous numerical approximation scheme is the Finite Element Method (FEM) (Courant, 1943; Hrennikoff, 1941). In the FEM, the PDE is discretized along with its domain, and the problem is transformed into solving a set of matrix equations. However, the computational complexity scales significantly with the resolution. For climate predictions, this number can be quite significant if the desired error is to be reached, which renders its use impractical.

In this paper, we propose to learn the *continuous* solutions for spatio-temporal PDEs. Previous methods focused on either generating fixed resolution predictions or generating arbitrary resolution solutions on a fixed grid. PDE models based on Multi-Layer Perceptrons (MLPs) can generate solutions at any point of the domain (Dissanayake and Phan-Thien, 1994; Lagaris et al., 1998; Raissi et al., 2017a). However, without imposing a physics-motivated loss that constrains the predictions to follow the smoothness bias resulting from the PDE, MLPs become less competitive than CNN-based approaches especially when the PDE solutions have high-frequency information (Rahaman et al., 2018).

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

We leverage the recent advances in Implicit Neural Representations ((Tancik et al., 2020), (Chen et al., 2020), (Jiang et al., 2020)) and propose a general purpose model that can not only learn solutions to a PDE with a resolution it was trained on, but it can also perform zero-shot super-resolution on irregular meshes. The added advantage is that we propose a general framework where we can make predictions given any spatial position query for both grid-based architectures like CNNs and graph-based ones able to handle sensors and predictions at arbitrary spatial positions.

Contributions Our main contributions are in the context of machine learning for approximately but efficiently solving PDEs and can be summarized as follows:

- We propose a framework that enables grid-based and graph-based architectures to generate continuous-space PDE solutions given a spatial query at any position.
- We show experimentally that this approach can generalize to resolutions up to four times those seen during training in zero-shot super-resolution tasks.

2 Related Works

Current solvers can require a lot of computations to generate solutions on a fine spatio-temporal grid. When it comes to climate predictions, General Circulation Models (GCM) are typically used to make forecasts that span several decades over the whole planet (Phillips, 1956). These GCMs use PDEs to model the climate in the atmosphere-ocean-land system and to solve these PDEs, classical numerical solvers are used. However, the quality of predictions is bottlenecked by the grid resolution that is in turn constrained by the available amount of computing power. Deep learning has recently emerged as an alternative to these classical solvers in hopes of generating data-driven predictions faster and making approximations that do not just rely on lower resolution grids but also on the statistical regularities that underlie the family of PDEs being considered. Using deep learning also makes it possible to combine the information in actual sensor data with the physical assumptions embedded in the classical PDEs. All of this would enable practitioners to increase the actual resolution further for the same computational budget, which in turn improves the quality of the predictions.

Machine Learning for PDE solving Dissanayake and Phan-Thien (1994) published one of the first papers on PDE solving using neural networks. They parameterized the solutions to the Poisson and heat transfer equations using an MLP and studied the evolution of the error with the mesh size. Lagaris et al. (1998) used MLPs for solving PDEs and ordinary differential equations. They wrote the solution as a sum of two components where the first term satisfies boundary conditions and is not learnable, and the second is parameterized with an MLP and trained to satisfy the equations. In Raissi et al. (2017a) the authors also parameterized the solution to a PDE using an MLP that takes coordinates as input. With the help of automatic differentiation, they calculate the PDE residual and use its MSE loss along with an MSE loss on the boundary conditions. In follow-up work, Raissi et al. (2017b) also learn the parameters of the PDE (e.g. Reynolds number for Navier-Stokes equations).

The recently introduced Neural Operators framework (Kovachki et al., 2021; Li et al., 2020b,a) attempts to learn operators between spaces of functions. Li et al. (2021) use "Fourier Layers" to learn the solution to a PDE by framing the problem as learning an operator from the space of initial conditions to the space of the PDE solutions. Their model can learn the solution to PDEs that lie on a uniform grid while maintaining their performance in the zero-shot super-resolution setting. In the same spirit, Jiang et al. (2020) developed a model based on Implicit Neural Representations called "MeshFreeFlowNet" where they upsample existing PDE solutions to a higher resolution. They use 3D low-resolution space-time tensors as inputs to a 3DUnet in order to generate a feature map. Next, some points are sampled uniformly from the corresponding high-resolution tensors and fed to an MLP called ImNet (Chen and Zhang, 2018). They train their model using a PDE residual loss and are able to predict the flow field at any spatio-temporal coordinate. Their approach is closest to the one we propose here. The main difference is that we perform super-resolution on the spatial queries and forecast the solution to a PDE instead of only doing super-resolution on the existing sequence.

Brandstetter et al. (2022) use the message-passing paradigm ((Gilmer et al., 2017), (Watters et al., 2017), (Sanchez-Gonzalez et al., 2020)) to solve 1D PDEs. They are able to beat state-of-the-art Fourier Neural Operators (Li et al., 2021) and classical WENO5 solvers while introducing the "pushforward trick" that allows them to generate better long-term rollouts. Moreover, they present an

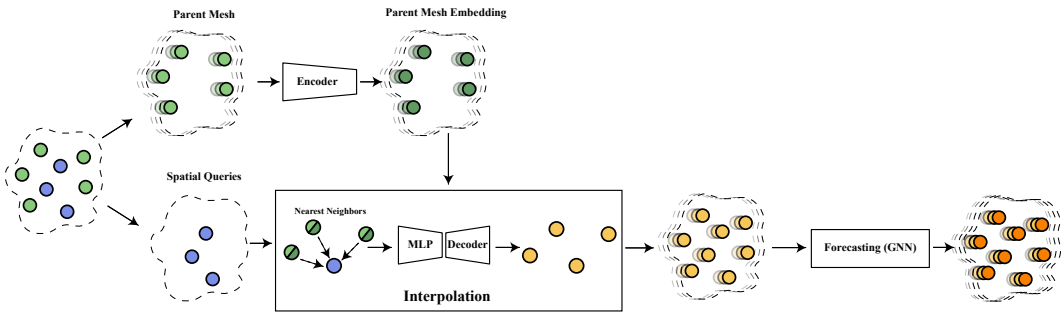


Figure 1: We illustrate the "Encode-Interpolate-Forecast" framework of MAgNet. The **parent mesh** is fed to the encoder to generate the **parent mesh embedding**. Next, we estimate the values at the **spatial queries** using the interpolation module that uses features from both the **parent mesh** points and the **parent mesh embedding** points closest to these queries. Finally, the **parent mesh** observations and interpolated values at **spatial queries** are gathered as nodes forming a **new graph** using nearest neighbors and the PDE solution is forecast for all nodes (therefore all spatial locations) into the **future** using the forecasting module.

added advantage over existing methods since they can learn PDE solutions at any mesh. However, they are not able to generalize to different resolutions.

Most machine learning approaches require data from a simulator in order to learn the required PDE solutions and that can be expensive depending on the PDE and the resolution. Wandel et al. (2020) alleviate that requirement by using a PDE loss.

Machine Learning for Turbulence Modeling Recent years have known a surge in machine learning-based models for modeling turbulence. Since it is expensive to resolve all relevant scales, some methods were developed that only solve large scales explicitly and separately model sub-grid scales (SGS). Recently, Novati et al. (2021) used multi-agent reinforcement learning to learn the dissipation coefficient of the Smagorinsky SGS model (Smagorinsky, 1963) using as reward the recovery of the statistical properties of Direct Numerical Simulations (DNS). Rasp et al. (2018) used MLPs to represent sub-grid processes in clouds and replace previous parametrization models in a global general circulation model. In the same fashion, Park and Choi (2021) used MLPs to learn DNS sub-grid scale (SGS) stresses using as input filtered flow variables in a turbulent channel flow. Brenowitz and Bretherton (2018) use MLPs to predicts the apparent sources of heat and moisture using coarse-grained data and use a multi-step loss to optimize their model.Wang et al. (2020) used one-layer CNNs to learn the spatial filter in LES methods and the temporal filter in RANS as well as the turbulent terms. A UNet (Ronneberger et al., 2015) is then used as a decoder to get the flow velocity. de Bezenac et al. (2017) predict future frames by deforming the input sequence according to the advection-diffusion equation. It yields good results for Sea-Surface Temperature predictions.

Stachenfeld et al. (2021) use the "encode-process-decode" (Sanchez-Gonzalez et al., 2018, 2020) paradigm along with dilated convolutional networks to capture turbulent dynamics seen in high-resolution solutions only by training on low spatial and temporal resolutions. Their approach beats existing neural PDE solvers in addition to the state-of-the-art Athena++ engine (Stone et al., 2020).

3 Methodology

We present the developed framework that leverages recent advances in Implicit Neural Representations (INR) (Jiang et al., 2020; Sitzmann et al., 2020; Chen et al., 2020; Tancik et al., 2020) and draws inspiration from mesh-free methods for PDE solving. We first start by giving a mathematical definition of a PDE. Next, we showcase the proposed "MAgNet" and derive two variants: A grid-based architecture and a graph-based one.

3.1 Preliminaries

We define PDE as follows, using D^k to denote k-th order derivatives:

Definition 3.1. Evans (2010) *Let U denote an open subset of $\mathbb{R}^n$ and $k \geq 1$ an integer. An expression of the form*:

$$\mathcal{L}(D^k\mathbf{u}(x), D^{k-1}\mathbf{u}(x), \ldots, \mathbf{u}(x), x) = \mathbf{0} \quad \forall x \in U \tag{1}$$

is called a k-th order system of PDEs, where $\mathcal{L} : \mathbb{R}^{mn^k} \times \mathbb{R}^{mn^{k-1}} \times \cdots \times \mathbb{R}^{mn} \times \mathbb{R}^m \times U \to \mathbb{R}^m$ is given and $\mathbf{u} : U \to \mathbb{R}^m, \mathbf{u} = (u^1, \ldots, u^m)$ is the unknown function to be characterized.

In this paper, we are interested in spatio-temporal PDEs. In this class of PDEs, the domain is $U = [0, +\infty] \times \mathcal{S}$ (time $\times$ space) where $\mathcal{S} \subset \mathbb{R}^n, \quad n \geq 1$ and, with D^k indicating differentiation wrt x, any such PDE can be formulated as:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathcal{L}(D^k\mathbf{u}(x), \ldots, \mathbf{u}(x), x, t) \forall t \geq 0, \forall x \in \mathcal{S}. \tag{2}$$

Numerical PDE simulations have enjoyed a great body of innovations especially where their use is paramount in industrial applications and research. Mesh-based methods like the FEM numerically compute the PDE solution on a predefined mesh. However, when there are regions in the PDE domain that present large discontinuities, the mesh needs to be modified and provided with many more points around that region in order to obtain acceptable approximations. Mesh-based methods typically solve this problem by re-meshing in what is called Adaptive Mesh Refinement (Berger and Oliger, 1984; Berger and Colella, 1989). However, this process can be quite expensive, which is why mesh-free methods have become an attractive option that goes around these limitations.

3.2 MAgNet: Mesh-Agnostic Neural PDE Solver

3.2.1 "Encode-Interpolate-Forecast" framework

Let $\{x_1, x_2, \ldots, x_T\} \in \mathbb{R}^{C \times N}$ denote a sequence of T frames that represents the ground-truth data coming from a PDE simulator or real-world observations. C denotes the number of physical channels, that is the number of physical variables involved in the PDE and N is the number of points in the mesh. These frames are defined on the *same* mesh, that is the mesh does not change in time. We call that mesh the *parent mesh* and denote its normalized coordinates of dimensionality n by $\{p_i\}_{1 \leq i \leq N} \in [-1, 1]^n$. Let $\{c_i\}_{1 \leq i \leq M} \in [-1, 1]^n$ denote a set of M coordinates representing the spatial queries. The task is to predict the solution for subsequent time steps both at: (i) all coordinates from the parent mesh $\{p_i\}_{1 \leq i \leq N}$, and (ii) coordinates from the spatial queries $\{c_i\}_{1 \leq i \leq M}$. At test time, the model can be queried at any spatially continuous coordinate within the PDE domain to provide an estimate of the PDE solution at those coordinates.

To perform the prediction, we first estimate the PDE solutions at the spatial queries for the first T frames and then use that to forecast the PDE solutions at the subsequent timesteps at the query locations. We do this through three stages (see Figure 1):

1. **Encoding**: The encoder takes as input the given PDE solution $\{x_t\}_{1 \leq t \leq T}$ at each point of the parent mesh $\{p_i\}_{1 \leq i \leq N}$ and generates a state-representation of original frames, which can be referred to as embeddings, and which we note $\{z_t\}_{1 \leq t \leq T}$. This representation will be used in the interpolation step to find the PDE solution at the spatial queries $\{c_i\}_{1 \leq i \leq M}$. Note that in this encoding step, we can generate one embedding for each frame such that we have T embeddings or summarize all the information in the T frames into one embedding. We will explain the methodology using T embeddings, as it is easier to grasp the time dimension in this formulation, but the implementation has been done using a summarized single embedding, as mentioned in section 3.2.2. We also note that the embedded mesh remains the same, i.e. we don't change it by upsampling or downsampling it.
2. **Interpolation**: We follow the same approach as Jiang et al. (2020) and Chen et al. (2020) by performing an interpolation in the feature space. Note that in case we generate one representation that summarizes all T frames into one, then $z_t = z$ for $t = 1, \ldots, T$. Let $\{t_k\}_{1 \leq k \leq T}$ denote the timesteps at which the x_t are generated.
   For each spatial query c_i, let $\mathcal{N}(c_i)$ denote the nearest points in the parent mesh p_j. We generate an interpolation of the features $z_k[c_i]$ at coordinates c_i and at timestep t_k as follows:

$$\forall k \in \{1, T\}, \forall i \in \{1, M\} : z_k[c_i] = \frac{\sum_{p_j \in \mathcal{N}(c_i)} w_j g_\theta(x_k[p_j], z_k[p_j], c_i - p_j, t_k)}{\sum_{p_j \in \mathcal{N}(c_i)} w_j} \tag{3}$$

Where $z_k[p_j]$ and $x_k[p_j]$ denote the embedding and input frame at position p_j and time t_k respectively. Moreover, w_j are interpolation weights and are positive and sum to one. Weights are chosen such that points closer to the spatial query have a higher contribution to the interpolated feature than points farther away from the spatial query. The g_θ is an MLP. To get the PDE solution $x_k[c_i]$ at coordinate c_i, we use a decoder d_θ which is an MLP here: $x_k[c_i] = d_\theta(z_k[c_i])$. In practice, the number of neighbors that we choose is 2^n where n is the dimensionality of the coordinates.

3. **Forecasting**: Now that we generated the PDE solution at the spatial queries c_i for all the past frames, we forecast the PDE solution at future time points at both spatial queries and the parent mesh coordinates. Let $\mathcal{G}$ denote the Nearest-Neighbors Graph (NNG) that has as nodes all the N locations in the parent mesh (at original coordinates $\{p_i\}_{1\leq i\leq N}$) as well as all the M query points (at locations $\{c_i\}_{1\leq i\leq M}$), with edges that include only the nearest neighbors of each node among the $N+M-1$ others. This corresponds to a new mesh represented by the graph $\mathcal{G}$. Let $\{c'_i\}_{1\leq i\leq M+N}$ denote the corresponding new coordinates. We generate the PDE solution for subsequent time steps on this graph auto-regressively using a decoder Δ_θ as follows:

$$x_{k+1}[c'_i] = x_k[c'_i] + (t_{k+1} - t_k)\Delta_\theta(x_k[c'_i], \ldots, x_1[c'_i]), \quad k = T, T+1, \ldots \tag{4}$$

We train MAgNet by using two losses:

- **Interpolation Loss**: This loss makes sure that the interpolated points match the ground-truth and is computed as follows:

$$L_{\text{interpolation}} = \frac{\sum_{i=1}^{M}\sum_{k=1}^{T} ||\hat{x}_k[c_i] - x_k[c_i]||_1}{T \times M} \tag{5}$$

  Where $\hat{x}_k[c_i]$ denotes the interpolated values generated by the model at the spatial queries.

- **Forecasting Loss**: This loss makes sure that the model predictions into the future are accurate. If H is the horizon of the predictions, then we can express the loss as follows:

$$L_{\text{forecasting}} = \frac{\sum_{i=1}^{M+N}\sum_{k=1}^{H} ||\hat{x}_{k+T}[c'_i] - x_{k+T}[c'_i]||_1}{H \times (M+N)} \tag{6}$$

  Where $\hat{x}_{k+T}[c'_i]$ denotes the forecasted values generated by the model at the graph $\mathcal{G}$ which combines both spatial queries and the parent mesh.

The final loss is then expressed as:

$$L = L_{\text{forecasting}} + L_{\text{interpolation}}.$$

3.2.2 Implementation Details

In the previous section, we described the general MAgNet framework. In this section, we present how we build the inputs to MAgNet as well as the architectural choices for the encoding, interpolation and forecasting modules and suggest two main architectures: MAgNet[CNN] and MAgNet[GNN].

Data pre-processing : We first consider a mesh that contains N' points ($N' \geq N$). We randomly sample N points from the mesh to form the parent mesh. During training, M spatial queries are randomly sampled from the $N' - N$ remaining points. We tried multiple values of M (that is the number of training spatial queries) to assess its impact on the performance of the method within a sensitivity study presented in in Section 4.3. The data pre-processing is illustrated in Figure 2 .

MAgNet[CNN] : In this architecture, we follow Chen et al. (2020) and adopt the EDSR architecture (Lim et al., 2017) as our CNN encoder. We concatenate all frames $\{x_t\}_{1\leq t\leq T}$ in the channel dimension and feed that to our encoder in order to generate a single representation z. For the forecasting module, we use the same GNN as in (Sanchez-Gonzalez et al., 2020). A key advantage of this architecture is that it effectively turns existing CNN architectures into mesh-agnostic ones by querying them at any spatially continuous point of the PDE domain at test time.

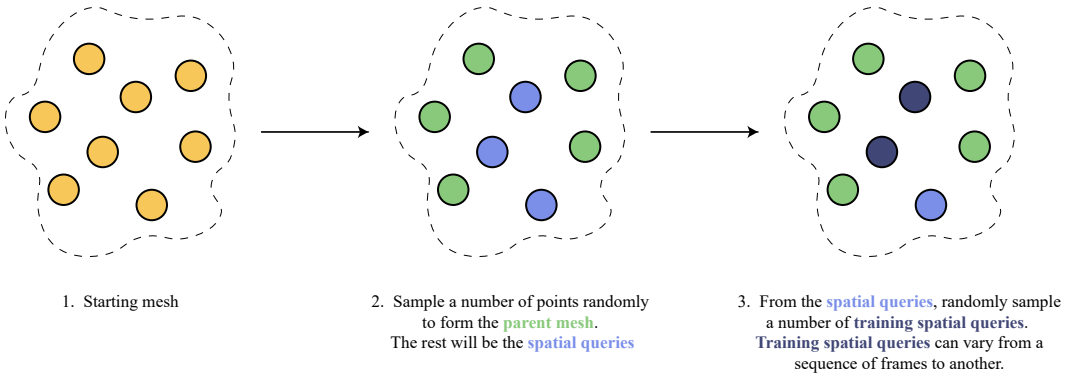


Figure 2: We illustrate the data pre-processing pipeline. We sample points randomly from the **starting mesh** to form the **parent mesh** and the remaining points form the **spatial queries**. Next, during training, we can sample from the **spatial queries** and form what we call "**training spatial queries**". The distinction is that the number of "**training spatial queries**" can be less than the total number of **spatial queries** and we investigate the impact of this number in Section 4.3.

MAgNet[GNN] : This model is similar to MAgNet[CNN] except that instead of using a CNN as an encoder, we use a GNN: the same architecture as in the forecasting module but each architecture having its separate set of parameters. This is better suited for encoding frames with irregular meshes. Similarly to MAgNet[CNN], we generate a single representation z that summarizes all the information from the frames $\{x_t\}_{1 \leq t \leq T}$.

4 Results

In this section, we evaluate MAgNet's performance against the following baselines:

Fourier Neural Operators (FNO) (Li et al., 2021) : Considered the state-of-the-art model in neural PDE solving, FNO casts the problem of PDE solving as learning an operator from the space of initial conditions to the space of the solutions. It is able to learn PDE solutions that lie on a uniform grid and can do zero-shot super resolution.

Message-Passing Neural PDE Solvers (MPNN) (Brandstetter et al., 2022) : Graph Neural Networks have been used to learn physical simulations with great success (Sanchez-Gonzalez et al., 2020). Recently, they have been used to learn solutions to PDEs (Brandstetter et al., 2022; Sanchez-Gonzalez et al., 2020). MPNN-based GNNs coupled with an autoregressive strategy demonstrate a superior performance to FNO and are able to make long rollouts with the help of the "pushforward-trick" that only propagates gradient of the last computed frame.

PDE Datasets We use three of MPNN's PDE simulations (Brandstetter et al., 2022) as our experimental testbed. In the same fashion, we are interested in the following family of PDEs:

$$\begin{cases} [\partial_t u + \partial_x(\alpha u^2 - \beta \partial_x u + \gamma \partial_{xx} u)](t,x) = \delta(t,x) \\ u(0,x) = \delta(0,x) \\ \delta(t,x) = \sum_{j=1}^{J} A_j \sin(\omega_j t + 2\pi l_j x / L + \phi_j) \end{cases} \tag{7}$$

Where $J = 5$, $L = 16$ and coefficients sampled uniformly in $A_j \in [-0.5, 0.5]$, $\omega_j \in [-.4, -0.4]$, '$j \in 1, 2, 3$, $\phi_j \in [0, 2\pi)$ and periodic boundary conditions following Brandstetter et al. (2022); Bar-Sinai et al. (2018). We denote the temporal resolution as n_t and and set it to $n_t = 250$ for the entire study. During testing, all the models are fed a history of $T = 25$ frames and produce a rollout of $n_t - T = 225$ frames in the future. Here, we present the three datasets we work with:

- **E1**: Burgers equation without diffusion $\alpha = 1, \beta = 0, \gamma = 0$

- **E2**: Burgers equation with variable diffusion $\alpha = 1, \beta = \eta, \gamma = 0$ where $\eta \in [0, 0.2]$
- **E3**: Mixed scenario where $\alpha \in [0, 3], \beta \in [0, 0.4]$ and $\gamma \in [0, 1]$.

All training sets contain 2048 simulations and test sets contain 128 simulations. All models are evaluated using the Mean Absolute Error (MAE) on the rolled out predictions averaged across time and space:

$$MAE = \frac{\sum_{t=1}^{n_t - T} \sum_{i=1}^{N} |x_t[c_i] - \hat{x}_t[c_i]|}{(n_t - T) \times N}. \tag{8}$$

We train models for 250 epochs with early stopping with a patience of 40 epochs. See Appendix A.2 and A.3 for more implementation details.

For all the subsequent sections, n_x and n'_x denote the training and testing set's resolutions respectively. The temporal resolution $n_t = 250$ remains unchanged for all experiments.

4.1 General Performance On Regular Meshes

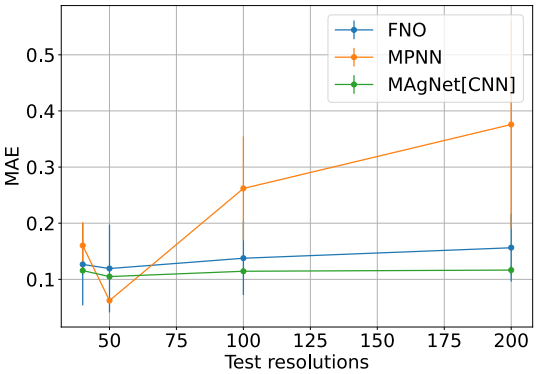
In this section, we compare MAgNet's performance on all three datasets. All models are trained on a resolution of $n_x = 50$ and the PDE solutions lie on a uniform grid. We test zero-shot super-resolution on $n'_x \in \{40, 50, 100, 200\}$. Results are summarized in Figure 3 and visualizations of the predictions can be found in Appendix A.1. MAgNet[CNN] outperforms both baselines on both **E1** and **E2** datasets yet is slightly outperformed by FNO on **E3** (Figure 3c). Nonetheless, MAgNet[CNN]'s predictive performance stays consistent up to $n'_x = 200$ while MPNN does not generalize well to resolutions not seen during training.

4.2 Zero-Shot Super-Resolution on Irregular Meshes

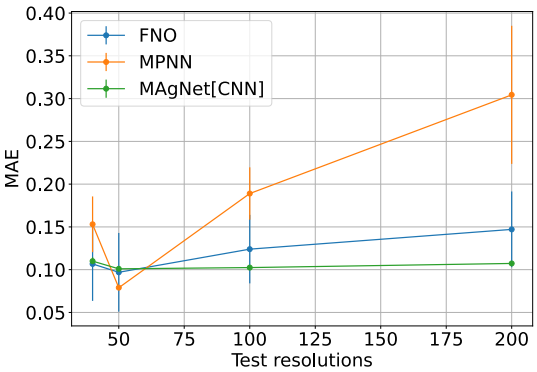
In this section, we study how MAgNets compare against the other baselines when it comes to making predictions on irregular meshes. In order to do so, we take simulations from the uniform-mesh **E1** dataset with a resolution of 100 and run the following steps:

1. Let $n_x \in \{30, 50, 70\}$.
2. For each simulation in the **E1** dataset, randomly sample the same subset of n_x points: the mesh remains for each single simulation in the **E1** dataset.

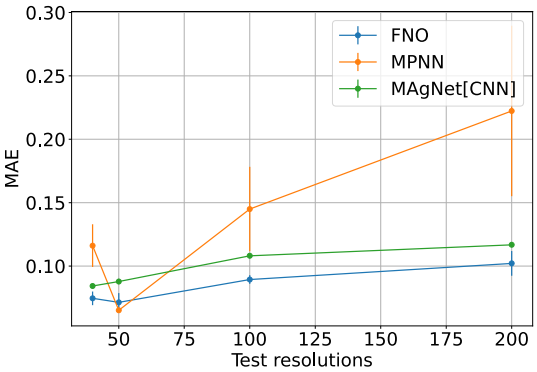
The procedure is the same for the test set but we instead take the original **E1** test set at a starting resolution of 200 and generate four test sets with irregular meshes for $n'_x \in \{40, 50, 100, 200\}$. This is different from the test set of the previous section albeit considering the same resolution since this one has irregular meshes. We summarize our findings in Table 1.



(a) Zero-shot super-resolution performance on **E1** test.



(b) Zero-shot super-resolution performance on **E2** test.



(c) Zero-shot super-resolution performance on **E3** test.

Figure 3: We present the models predictive performance on Zero-shot super-resolution, with a training spatial resolution of $n_x = 50$. MAgNet[CNN] outperforms baselines on both **E1** and **E2** test sets but lags behind FNO on **E3**. Error bars represent one standard deviation in both plots.

	$\mathbf{n_x = 30}$				$\mathbf{n_x = 50}$				$\mathbf{n_x = 70}$			
Model	$\mathbf{n'_x = 40}$	$\mathbf{n'_x = 50}$	$\mathbf{n'_x = 100}$	$\mathbf{n'_x = 200}$	$\mathbf{n'_x = 40}$	$\mathbf{n'_x = 50}$	$\mathbf{n'_x = 100}$	$\mathbf{n'_x = 200}$	$\mathbf{n'_x = 40}$	$\mathbf{n'_x = 50}$	$\mathbf{n'_x = 100}$	$\mathbf{n'_x = 200}$
FNO	0.2784	0.2471	0.2574	0.2501	0.3797	0.3324	0.3841	0.3821	0.2798	0.2341	0.2533	0.2605
MAgNet[CNN]	**0.2081**	0.1934	0.2063	0.2150	0.1869	**0.1630**	**0.1599**	0.1629	**0.2237**	0.1634	0.1385	0.1324
MPNN	0.2602	**0.1601**	0.3451	0.3667	0.3027	0.2521	0.3226	0.3243	0.2685	**0.1541**	0.3403	0.3570
MAgNet[GNN]	0.2422	0.2230	**0.1938**	**0.1902**	0.2302	**0.1659**	**0.1590**	**0.1404**	0.2400	**0.1599**	**0.1398**	**0.1070**

Table 1: We report the MAE per frame on the **E1** dataset. We train all four models on three different resolutions $n_x \in \{30, 50, 70\}$ and for each training resolution, we evaluate zero-shot super-resolution on irregular meshes for $n'_x \in \{40, 50, 100, 200\}$. We notice that even when we use a CNN encoder, MAgNet not only performs better than the existing baselines, but its performance stays consistent across different test resolutions. MAgNet with a CNN encoder beats MPNN even when using an encoder not suited for the task, which suggests MAgNet successfully turns existing CNN architectures into mesh-agnostic ones.

MAgNet[GNN] performs better than MAgNet[CNN] on irregular meshes which is expected since GNN encoders are better suited for this task. However, surprisingly, even though we use a CNN encoder for MAgNet[CNN], the performance seems to be better in most cases not only compared to FNO but also MPNN which is a graph-based architecture. This effectively shows that MAgNet can be used to turn existing CNN architectures into mesh-agnostic solvers. This is particularly interesting for meteorological applications where one needs to make predictions at the sub-grid level (at a specific coordinate) while only having access to measurements on a grid.

4.3 Ablation and Sensitivity studies

In this section, we study different architectural choices and the sensitivity of key parameters in MAgNet.

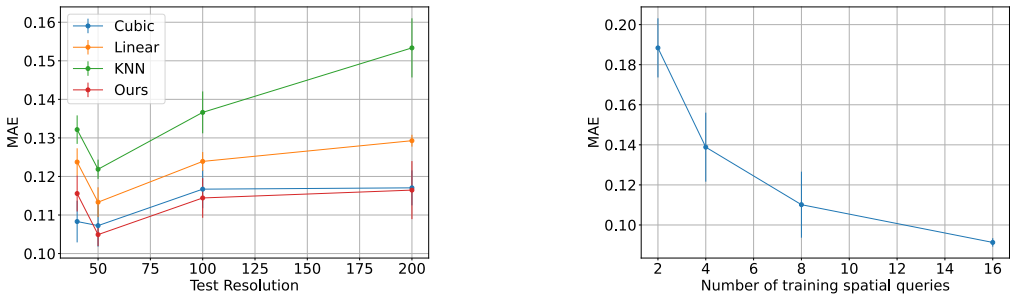
Basic Interpolators vs Learned Interpolators We investigate the contribution of the interpolation module to the general predictive performance of MAgNet. We compare the MAgNet[CNN] architecture against three ablated variants:

- **KNN**: We use K-Nearest-Neighbors interpolation (Qi et al., 2017) on the original frames directly to obtain the interpolated values at the spatial queries.
- **Linear**: We use Linear interpolation on the original frames directly to obtain the interpolated values at the spatial queries.
- **Cubic**: We use Cubic interpolation on the original frames directly to obtain the interpolated values at the spatial queries.

Everything else is kept the same. The evaluation is done on the **E1** dataset with regular meshes and a resolution of $n_x = 50$. Performance is tested on **E1** with regular meshes for test resolutions $n'_x \in \{40, 50, 100, 200\}$. Results are summarized in Figure 4a.

Effect of modeling interactions between the parent mesh and the spatial query In section 3.2, we presented the developed method which can be used to generate solutions at any spatial query through the "Encode-Interpolate-Forecast" framework. This means that we are free to choose any architecture for the three processes. In this section we investigate the choice of the "Forecast" architecture on the predictive performance as well as zero-shot super-resolution capabilities. We compare MAgNet[CNN] and a variant that uses LSTM with attention ((Hochreiter and Schmidhuber, 1997; Bahdanau et al., 2015) on the spatial queries only. Results are shown in Table 2. We see that leveraging the interaction between the coordinates from the spatial query and those in the parent mesh enables the model to give predictions consistent to different resolutions as opposed to generating the solutions at these queries with no interaction.

Impact of the number of point samples during training We study the impact of the number of spatial queries used during training (M). We train MAgNet[GNN] on the **E1** dataset with a resolution of $n_x = 50$ on a uniform grid and test on the same resolution. Our findings are summarized in Figure4b. Increasing the number of spatial queries increases the predictive performance as expected. Moreover, having many queries also decreases the variance of the results. When we have fewer points,



(a) Ablation study between learned interpolators (Ours) and existing interpolation schemes.

(b) Sensitivity study to assess the impact of the number of spatial queries seen during training.

Figure 4: In (a), we study the impact of having a learned interpolator as compared to basic ones. In (b), we assess the impact of the number of spatial queries during training. Error bars represent one standard deviation in both plots.

Table 2: Mean Absolute Error (MAE) reported for models trained on the **E1** dataset with a resolution of $n_x = 50$ on a uniform grid and tested on the same dataset for $n_x \in \{40, 50, 100, 200\}$. We evaluate the effect of the Forecast module on the zero-shot super-resolution capabilities. The model without interaction contains an LSTM with attention (Bahdanau et al., 2015; Hochreiter and Schmidhuber, 1997) for forecasting where spatial queries do not interact with the parent mesh. The model with interaction has a GNN that operates over the graph formed from the spatial-queries and the parent mesh (MAgNet[CNN]).

Model	$n_x = 40$	$n_x = 50$	$n_x = 100$	$n_x = 200$
Without Interaction	0.1650	0.0815	0.2810	0.4139
With Interaction	0.1079	0.1020	0.1142	0.1177

the random sampling can cause some of these points to be in regions that decrease the loss faster than other regions, hence, the model's performance becomes sensitive to randomization. However, this effect grows weaker as the number of queries increases since they would uniformly cover more regions in the mesh.

5 Limitations and Future Work

In this paper we introduced a novel framework that we call MAgNet for solving PDEs on any mesh, possibly irregular. We proposed two variants of the architecture which gave promising results on benchmark datasets. We were effectively able to beat graph-based and grid-based architectures even when using the CNN variant of the proposed framework, therefore suggesting a novel way of adapting existing CNN architectures to make predictions on any mesh. A key limitation of our work however, is the significance of the learned interpolator. Indeed, compared with a simple cubic interpolation, the approach introduced here doesn't seem to offer a significant advantage and we leave improvement regarding this point for future work. Another improvement could be seen in the forecasting module. For now, MAgNet forecasts using a first-order explicit time-stepping scheme that is known to suffer from instability problems in numerical PDE and ODE solvers. Learned solvers seem to somehow circumvent this limitation even when using large time steps (Sanchez-Gonzalez et al., 2020; Brandstetter et al., 2022; Stachenfeld et al., 2021). In a future work, we wish to explore other time-stepping schemes such as the 4th order Runge-Kutta method (Runge, 1895; Kutta, 1901) which is commonly used for solving PDEs.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning*

Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. (2018). Learning data driven discretizations for partial differential equations.

Berger, M. and Colella, P. (1989). Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84.

Berger, M. J. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512.

Brandstetter, J., Worrall, D., and Welling, M. (2022). Message passing neural pde solvers.

Brenowitz, N. D. and Bretherton, C. S. (2018). Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, 45(12):6289–6298.

Chen, Y., Liu, S., and Wang, X. (2020). Learning continuous image representation with local implicit image function.

Chen, Z. and Zhang, H. (2018). Learning implicit fields for generative shape modeling.

Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49(1):1–23.

de Bezenac, E., Pajot, A., and Gallinari, P. (2017). Deep learning for physical processes: Incorporating prior scientific knowledge.

Dissanayake, M. W. M. G. and Phan-Thien, N. (1994). Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201.

Evans, L. (2010). *Partial Differential Equations*. American Mathematical Society.

Falcon, W., Borovec, J., Wälchli, A., Eggert, N., Schock, J., Jordan, J., Skafte, N., Ir1dXD, Bereznyuk, V., Harris, E., Tullie Murrell, Yu, P., Præsius, S., Addair, T., Zhong, J., Lipin, D., Uchida, S., Shreyas Bapat, Schröter, H., Dayma, B., Karnachev, A., Akshay Kulkarni, Shunta Komatsu, Martin.B, Jean-Baptiste SCHIRATTI, Mary, H., Byrne, D., Cristobal Eyzaguirre, Cinjon, and Bakhtin, A. (2020). Pytorchlightning/pytorch-lightning: 0.7.6 release.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hrennikoff, A. (1941). Solution of problems of elasticity by the framework method. *Journal of Applied Mechanics*, 8(4):A169–A175.

Jiang, C. M., Esmaeilzadeh, S., Azizzadenesheli, K., Kashinath, K., Mustafa, M., Tchelepi, H. A., Marcus, P., Prabhat, and Anandkumar, A. (2020). *MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework*. IEEE Press.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2021). Neural operator: Learning maps between function spaces.

Kutta, W. (1901). Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeit. Math. Phys.*, 46:435–53.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9 5:987–1000.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020a). Neural operator: Graph kernel network for partial differential equations.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., and Anandkumar, A. (2020b). Multipole graph neural operator for parametric partial differential equations. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6755–6766. Curran Associates, Inc.

Li, Z.-Y., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2021). Fourier neural operator for parametric partial differential equations. *ArXiv*, abs/2010.08895.

Lim, B., Son, S., Kim, H., Nah, S., and Lee, K. M. (2017). Enhanced deep residual networks for single image super-resolution.

Novati, G., de Laroussilhe, H. L., and Koumoutsakos, P. (2021). Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1):87–96.

Park, J. and Choi, H. (2021). Toward neural-network-based large eddy simulation: application to turbulent channel flow. *Journal of Fluid Mechanics*, 914.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA.

Phillips, N. A. (1956). The general circulation of the atmosphere: A numerical experiment. *Quarterly Journal of the Royal Meteorological Society*, 82(352):123–164.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., and Courville, A. (2018). On the spectral bias of neural networks.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017a). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017b). Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations.

Rasp, S., Pritchard, M. S., and Gentine, P. (2018). Deep learning to represent sub-grid processes in climate models.

Richardson, L. F. and Lynch, P. (2007). *Weather Prediction by Numerical Process*. Cambridge University Press.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*.

Runge, C. (1895). Ueber die numerische auflsung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. (2020). Learning to simulate complex physics with graph networks.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control.

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., and Wetzstein, G. (2020). Implicit neural representations with periodic activation functions.

Smagorinsky, J. (1963). General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164.

Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A. (2021). Learned coarse models for efficient turbulence simulation.

Stone, J. M., Tomida, K., White, C. J., and Felker, K. G. (2020). The athena adaptive mesh refinement framework: Design and magnetohydrodynamic solvers. *The Astrophysical Journal Supplement Series*, 249(1):4.

Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains.

Wandel, N., Weinmann, M., and Klein, R. (2020). Learning incompressible fluid dynamics from scratch – towards fast, differentiable fluid models that generalize.

Wang, R., Kashinath, K., Mustafa, M., Albert, A., and Yu, R. (2020). Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. (2017). Visual interaction networks: Learning a physics simulator from video. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We are cleaning the github repository for public use. It will include the final code and link to our datasets.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Training and implementation details can be found in the Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We ran models for 5 random seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We are cleaning the github repository for public use. It will include the final code and link to our datasets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Visualizations

Figures 5 and 6 show the models' predictions on each of the test set resolutions $n'_x \in \{40, 50, 100, 200\}$ on the **E1** dataset. MAgNet[CNN] predictions visually match the ground-truth's while MPNN's prediction degrade as the predictions are advanced in time. The models shown here are the ones trained on uniform meshes.

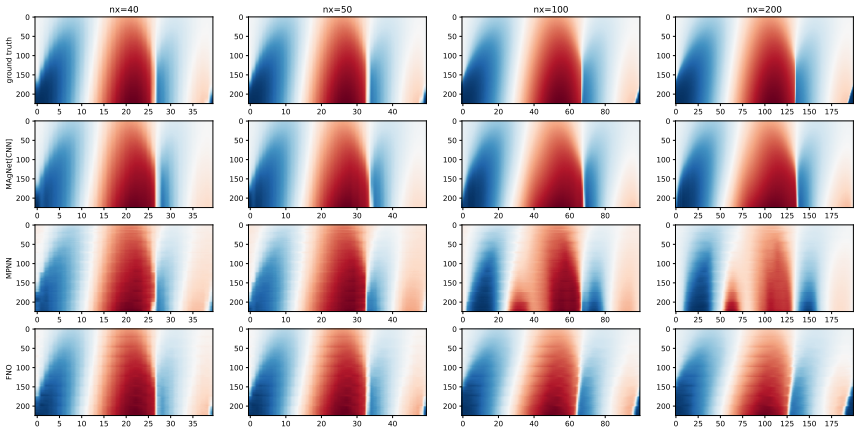


Figure 5: Vizualisation of the models' predictions on a simulation sample from the **E1** dataset. We present visualizations for each of the test resolutions $n'_x \in \{40, 50, 100, 200\}$. The temporal resolution is fixed at $n_t = 250$. The x axis represents space and y axis represents time. The arrow of time is from top to bottom.

A.2 Training details

We train all models for 250 epochs and early stopping with a patience of 40 epochs. All models are trained using Adam Optimizer (Kingma and Ba, 2014) and the StepLR learning scheduler (Paszke et al., 2019) which decays the learning rate by a factor k every N_{steps} epochs. All models were trained on 5 random seeds $\in \{5, 10, 21, 42, 2022\}$. We use Pytorch (Paszke et al., 2019) and Pytorch-Lightning (Falcon et al., 2020) for our implementations and we summarize all model's training hyper-parameters in Table 3.

A.3 Architectural details

A.3.1 MAgNet[CNN]

Encoder Architecture We adapt the original EDSR (Lim et al., 2017) architecture to work on 1D signals instead of 2D and use 4 residual blocks with a hidden dimension of 128.

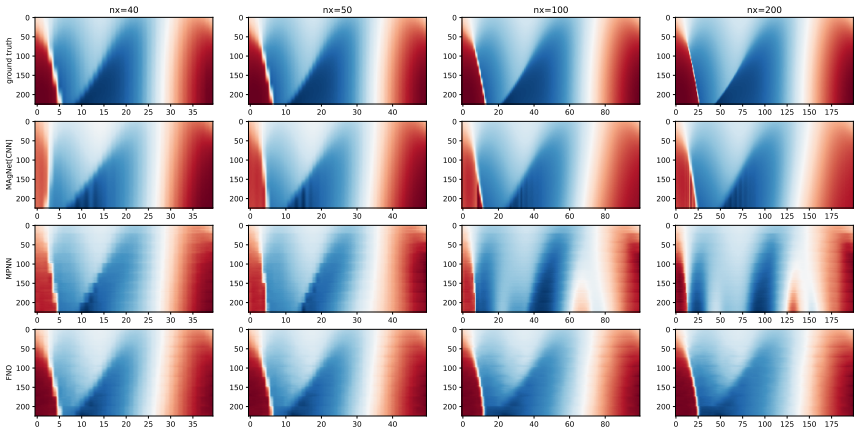


Figure 6: Vizualisation of the models' predictions on a simulation sample from the **E1** dataset. We present visualizations for each of the test resolutions $n'_x \in \{40, 50, 100, 200\}$. The temporal resolution is fixed at $n_t = 250$. The x axis represents space and y axis represents time. The arrow of time is from top to bottom.

Table 3: Training hyperparameters for FNO, MPNN, MAgNet[CNN] and MAgNet[GNN]

Parameters	**FNO**	**MPNN**	**MAgNet[CNN]**	**MAgNet[GNN]**
Learning Rate	0.001	0.001	0.001	0.001
Weight Decay	0	0	0	0
k	0.3	0.3	0.3	0.3
N_{steps}	50	50	40	50
GPU	RTX8000	RTX8000	RTX8000	RTX8000
Number of GPUs	1	1	2	2
Training duration (hours)	1	1	5	2.5

Interpolation Module We use a 4 layers MLP with a hidden size of 64 followed by Layernorm for g_θ. For d_θ, we use a 4 layer MLP with a hidden size of 64

Forecasting Module We use the same architecture as in Sanchez-Gonzalez et al. (2020). The encoder module uses a 4 layer MLP with a hidden size of 64 and the latent dimension to 32. We use 5 message-passing steps (with the same parameters for the MLP), the decoder also has a 4 layer MLP with hidden size of 64.

A.3.2 MAgNet[GNN]

Encoder Architecture We use the same architecture as in Sanchez-Gonzalez et al. (2020) but only keep the encoder and processor. We also use 5 message-passing steps for the processor and use 4 layer MLP with hidden size of 128 and a latent dimension of 128.

Interpolation Module We use a 4 layers MLP with a hidden size of 128 followed by Layernorm for g_θ. For d_θ, we use a 4 layer MLP with a hidden size of 128

Forecasting Module We use the same architecture as in Sanchez-Gonzalez et al. (2020). The encoder module uses a 4 layer MLP with a hidden size of 64 and the latent dimension set to 128. We use 5 message-passing steps for the processor (with the same parameters for the MLP), the decoder also has a 4 layer MLP with hidden size of 128.

A.3.3 MPNN Brandstetter et al. (2022)

We use the same hyperparameters as in Brandstetter et al. (2022).

A.3.4 FNO (Li et al., 2021)

We use 5 Fourier Layers and 12 modes in Fourier space. We use a hidden channel size of 256.