MULTI-SCALE MESSAGE PASSING NEURAL PDE SOLVERS

Léonard Equer
ETH Zürich

T. Konstantin Rusch
ETH Zürich and UC Berkeley
trusch@ethz.ch

Siddhartha Mishra
ETH Zürich

ABSTRACT

We propose a novel *multi-scale message passing neural network algorithm* for learning the solutions of time-dependent PDEs. Our algorithm possesses both temporal and spatial multi-scale resolution features by incorporating multi-scale sequence models and graph gating modules in the encoder and processor, respectively. Benchmark numerical experiments are presented to demonstrate that the proposed algorithm outperforms baselines, particularly on a PDE with a range of spatial and temporal scales.

1 INTRODUCTION

Time-dependent partial differential equations (PDEs) arise as mathematical models of many interesting phenomena in the sciences and engineering that involve the time-evolution of physical quantities of interest (Evans, 2010). Solving such PDEs entails computing the so-called *solution operator* that maps the initial conditions (and other inputs such as coefficients, sources etc) to the trajectories of the solution over time. *Classical* numerical methods which combine spatial discretizations such as finite differences, finite elements or spectral methods together with Runge-Kutta or multi-step temporal discretization schemes are widely used to simulate time-dependent PDEs (Quarteroni & Valli, 1994). However, these methods can be prohibitively expensive, particularly in several spatial dimensions and for long-time integration.

Recently, machine learning based algorithms are increasingly being used for the fast and accurate simulation of time-dependent PDEs. Examples include supervised learning algorithms (Zhu & Zabaras, 2018; Lye et al., 2020), physics informed neural networks (Raissi & Karniadakis, 2018; Raissi et al., 2019) and operator learning algorithms such as DeepONets (Lu et al., 2021) and Fourier Neural Operators (Li et al., 2021). However, each of these frameworks raises many unaddressed issues in terms of applicability, efficiency and generalization capacity.

In particular, existing frameworks such as FNOs or CNNs rely on input (and output) data on uniform Cartesian grids whereas in practice for engineering applications, data generated from simulations or observations is only available on unstructured grids. Given this discrepancy, learning frameworks that admit inputs and outputs on general and highly variable grids could be useful as PDE solvers. In this context, models based on graph neural networks are increasingly being considered as attractive frameworks for learning PDEs (Lino et al., 2021; Boussif et al., 2022; Battaglia et al., 2018) and references therein.

A very attractive framework in this regard was recently proposed in Brandstetter et al. (2022), where the authors suggested an autoregressive message passing procedure to learn solution operators of time-dependent PDEs. The approach utilized the well-known *encode-process-decode* paradigm (Battaglia et al., 2018) by embedding time-dependent inputs into a feature vector on a graph (related to the underlying possibly unstructured grid), processing this feature vector through message passing graph neural networks (GNNs) and mapping this latent representation into a time-update through a decoder. This approach was shown to be competitive vis a vis other models for some representative examples, particularly of PDEs where the solution only has a small range of scales in both space and time.

However in practice, solutions to a large class of time-dependent PDEs contain structures at a wide range of scales in both space and time. Such *multi-scale* PDEs (Kuehn, 2015) include the well-

known models of fluid dynamics, wave propagation and reaction-diffusion mechanisms. Our aim in this paper is to propose a machine learning framework that can accurately learn the solutions of such multiscale time-dependent PDEs. To this end, we base our algorithm on the graph learning based framework of Brandstetter et al. (2022) but endow it with *multi-scale* features. In particular, the encoder is supplemented with a very recent multi-scale sequence modeling algorithm called the *long expressive memory* (LEM) (Rusch et al., 2022). Similarly, instead of employing standard GNNs as processors, we modify them by a novel *gating* mechanism, analogous to the one suggested recently in Rusch et al. (2023). We demonstrate through numerical experiments that these multi-scale augmentations not only improve performance on standard single-scale benchmarks but also significantly outperform competing models on a multi-scale time-dependent PDE. These promising results pave the way for the design of a robust and accurate autoregressive message passing framework for learning time-dependent PDEs.

2 THE METHOD.

Setting. We consider the following abstract form of a time-dependent PDE,

$$\begin{aligned} \partial_t u &= \mathcal{N}_\eta(u), \quad \text{in } \Omega\times]0,T[\\ u &= u_0(x) \quad \text{in } \bar{\Omega}\times\{0\} \end{aligned} \tag{1}$$

Here, $\Omega \subset \mathbb{R}^d$ is a bounded open set and $T > 0$. The differential operator $\mathcal{N}_\eta : \mathcal{H} \mapsto \bar{\mathcal{H}}$, maps between two Hilbert spaces $\mathcal{H}, \bar{\mathcal{H}}$ and η models a coefficient, which for simplicity we assume to be finite-dimensional $\eta \in \mathbb{R}^{d_\eta}$. Finally the initial data is $u_0 \in \mathcal{H}$ and the PDE equation 1 is augmented with suitable boundary conditions. Our objective is to learn the solution operator

$$\mathcal{S}_t^\eta : \mathcal{H} \mapsto \mathcal{H}, \quad u(.,t) = \mathcal{S}_t^\eta u_0. \tag{2}$$

To this end, we will proceed by learning the *autoregressive* mapping

$$u(.,t+\Delta t) = \mathcal{A}_\eta^{\Delta t}(u(.,t)) \tag{3}$$

In other words, the above operator maps the solution $u(.,t)$ of equation 1 at current time t to the solution at a later time $t+\Delta t$. By iteratively applying $\mathcal{A}_\eta^{\Delta t}$, we can extend the solution over the entire time period. We drop the Δt-dependence below for notational convenience.

In order to learn this mapping, we define a grid $\{x_i \in \Omega \mid i \in \mathbb{N}, 1 \leq i \leq n_x\}$ such that we obtain a finite dimensional approximation of $u(.,t) \in \mathcal{H}$ as $\mathbf{u}(t) = [u(t,x_1),\ldots,u(t,x_{n_x})]^\top \in \mathbb{R}^{n_x}$. We can now approximate the infinite dimensional operator $\mathcal{A}_\eta$ by a finite dimensional operator $\mathcal{A}_{\eta,\theta}$ parameterized by $\theta \in \Theta \subset \mathbb{R}^D$ as

$$\mathbf{u}(t+\Delta t) = \mathcal{A}_{\eta,\theta}(\mathbf{u}(t)) \tag{4}$$

Next, we will describe the key ingredients of our multi-scale message-passing paradigm to learn the autoregressive map 4.

The message passing framework of Brandstetter et al. (2022). We start by a brief description of the message passing framework of Brandstetter et al. (2022), which we will augment with multi-scale features later.

As mentioned before, this framework follows a *encode-process-decode* paradigm. The input to the **encoder** is the vector of K-lagged [1] solutions $\mathbf{u}^{k-K:k}$, containing the (recent) history of the solution trajectory. This vector, at each grid point, is then *embedded* into a high-dimensional feature vector $\mathbf{X}_i^0$ at each node i of a *Graph* $\mathcal{G}$, which in turn, is defined in terms of the underlying grid points forming nodes $\{i\}_i$ and sets of nearest neighbors $\{\mathcal{N}(i)\}_i$ constituting edges (see **SM** A.2 for details of this computational graph). The encoder mapping of Brandstetter et al. (2022) is a shallow neural network. Next, the feature vector $\mathbf{X}_i^0$ is augmented with relative positions $x_i - x_j$, the equation parameters η as well as the solution differences $\mathbf{u}_i^{k-K:k} - \mathbf{u}_j^{k-K:k}$ and is processed through a multi-hidden layer *message passing neural network* (MPNN) (Gilmer et al., 2017), with the relative positions, parameters and solution finite differences being fed as inputs to each hidden layer. The output of the last hidden layer $\{\mathbf{X}_i^L\}_i$ is then transformed into the update $\mathbf{u}^{k:k+K}$ that

[1] A K-lagged solution $\mathbf{u}^{i-K:i}$ is defined as the following set of vectors $\{\mathbf{u}(t_{i-K}), \mathbf{u}(t_{i-K+1}), ..., \mathbf{u}(t_{i-1})\}$

provides future trajectories of the solution. The **decoder** is a one-dimensional convolutional neural network. Summarizing the message passing framework of Brandstetter et al. (2022) yields the following mapping,

$$\mathbf{u}^{k:k+K} = \mathcal{A}_{\eta,\theta}(\mathbf{u}^{k-K:k}, \mathcal{G}) \tag{5}$$

for updating the solution trajectories of the time-dependent PDE equation 1.

Resolving multiple time scales with long-expressive memory (LEM). Long expressive memory (LEM) was proposed recently in Rusch et al. (2022) as a sequence model that can i) learn long-term dependencies in sequential data as it solves the exploding and vanishing gradient problem and ii) it can efficiently process multiple scales in the data. It is this latter feature that we seek to exploit in our context. LEM is based on the structure preserving implicit-explicit discretization of an ODE system such that the recurrent update rule becomes

$$\begin{aligned}
\mathbf{\Delta t}_n &= \Delta t \hat{\sigma}\left(\mathbf{W}_1 \mathbf{y}_{n-1} + \mathbf{V}_1 \mathbf{u}_n + \mathbf{b}_1\right) \\
\overline{\mathbf{\Delta t}}_n &= \Delta t \hat{\sigma}\left(\mathbf{W}_2 \mathbf{y}_{n-1} + \mathbf{V}_2 \mathbf{u}_n + \mathbf{b}_2\right) \\
\mathbf{z}_n &= \left(1 - \mathbf{\Delta t}_n\right) \odot \mathbf{z}_{n-1} + \mathbf{\Delta t}_n \odot \sigma\left(\mathbf{W}_z \mathbf{y}_{n-1} + \mathbf{V}_z \mathbf{u}_n + \mathbf{b}_z\right) \\
\mathbf{y}_n &= \left(1 - \overline{\mathbf{\Delta t}}_n\right) \odot \mathbf{y}_{n-1} + \overline{\mathbf{\Delta t}}_n \odot \sigma\left(\mathbf{W}_y \mathbf{z}_n + \mathbf{V}_y \mathbf{u}_n + \mathbf{b}_y\right)
\end{aligned} \tag{6}$$

where $\hat{\sigma}$ is a sigmoid activation function, $\sigma(u) = \tanh(u)$, $\mathbf{W}_{1,2,y,z}$ and $\mathbf{V}_{1,2,y,z}$ are weight matrices and $\mathbf{b}_{1,2,y,z}$ bias vectors. The discretized system evolves the *hidden states* by the update formula $\mathbf{z_n}, \mathbf{y_n} = \mathbf{f}(\mathbf{z_{n-1}}, \mathbf{y_{n-1}}, \mathbf{u_n}, \Delta t)$. We will apply LEM in the encoder step of our proposed architecture to embed our input vector into an expressive high dimensional feature vector.

Resolving multiple spatial scales with Graph Gating. Following the recent paper Rusch et al. (2023), we can endow a GNN with the explicit ability to resolve multiple spatial scales by adding a *gating* mechanism. To this end, let $\mathbf{X} \in \mathbb{R}^{n_x \times N_{\text{hid}}}$ be a feature matrix, $\mathcal{G}$ the graph representation and $\mathbf{F}_{\hat{\theta}}$, $\mathbf{F}_\theta$ two MPNN with different weights. We can now represent the MPNN updates as

$$\mathbf{X}^n = \left(\mathbf{1} - \hat{\sigma}\left(\mathbf{F}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})\right)\right) \odot \mathbf{X}^{n-1} + \hat{\sigma}\left(\mathbf{F}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})\right) \odot \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})) \tag{7}$$

Note that in this case we create the two graph neural networks with the same architecture, $\mathbf{F}_{\hat{\theta}}(\mathbf{X}, \mathcal{G})$ learns the scales in the data to act as gating switches when its output are normalized between 0 and 1 while $\mathbf{F}_\theta(\mathbf{X}, \mathcal{G})$ updates the feature vector according to the message passing rule.

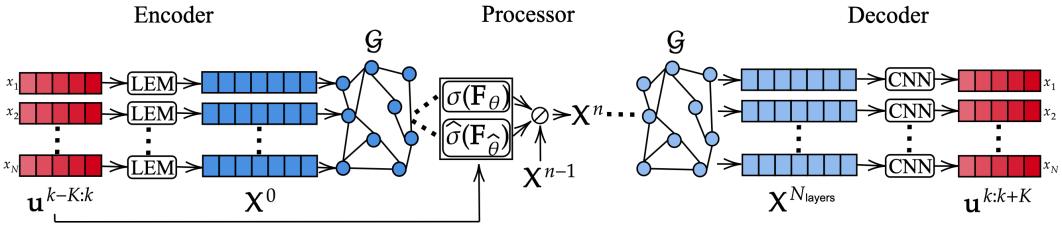

Figure 1: Multi-scale message passing architecture, the encoder applies a LEM on each node input to generate the node embeddings, the processor then performs N_{layers} of message passing with gating ($\oslash(a, b, c) = (1 - b) \odot a + b \odot c$). The node features are then passed through a 1D convolutional layer to project back to the input dimension.

The multi-scale message passing algorithm. We combine the above three ingredients to form the multi-scale message passing neural PDE solver, which is summarized in Figure 1. We make the following changes to the message passing architecture of Brandstetter et al. (2022): i) in the encoder step, an additional LEM layer is introduced that processes the input vector $\mathbf{u}^{k-K:k}$, at each node, with a LEM recurrent neural network that processes this input to resolve multiple time scales and ii) in the processor step, we augment the standard message passing neural network with a gating neural network as in equation 7 to endow the architecture to resolve multiple spatial scales (see **SM** A.2).

3 RESULTS

We test the proposed multi-scale message passing procedure on 3 benchmark experiments below. As a baseline, we will use the message passing algorithm of Brandstetter et al. (2022) that we abbreviate as **MP-PDE**. Our multi-scale variant, which uses LEM as a component of the encoder and a gating GNN is abbreviated as **MSMP-PDE**. As additional baselines, we ablate different parts of the proposed algorithm namely, remove the LEM component to obtain **Gated**, remove the gating to obtain **LEM**, replace LEM with LSTM in **LEM** and in **MSMP-PDE** to obtain **LSTM** and **LSTMGated**, respectively.

Table 1: Test L^2 (mean $\pm$ std) relative errors for competing models in all the experiments. The two-best performing models are highlighted in bold.

	E1	**E2**	**MS-wave**
MP-PDE	$0.472\% \pm 0.037\%$	$7.309\% \pm 0.484\%$	$20.39\% \pm 1.64\%$
LSTM	$0.587\% \pm 0.137\%$	$6.788\% \pm 0.176\%$	$15.84\% \pm 1.27\%$
LEM	$0.455\% \pm 0.026\%$	$6.866\% \pm 0.235\%$	$17.08\% \pm 1.45\%$
Gated	$0.364\% \pm 0.051\%$	$6.469\% \pm 0.259\%$	$\mathbf{11.9\% \pm 1.22\%}$
LSTMGated	$\mathbf{0.326\% \pm 0.026\%}$	$\mathbf{6.207\% \pm 0.345\%}$	$12.49\% \pm 2.79\%$
MSMP-PDE	$\mathbf{0.323\% \pm 0.034\%}$	$\mathbf{6.302\% \pm 0.373\%}$	$\mathbf{10.36\% \pm 0.99\%}$

For both the first and second numerical experiments, we will consider Burgers' equation (see **SM** A.1.1) on the computational domain $[0, L]$. In the first experiment, that we abbreviate as **E1**, we follow Brandstetter et al. (2022) to consider the inviscid Burgers' equation with the sinusoidal initial data given in **SM** equation 9. The training (and test) data are generated for different initial conditions with a finite volume scheme and we train all the models on 2048 samples and test them on 128 samples. The test errors are presented in Table 1. We observe from this table that all the models yield very low relative errors. This is not unexpected as the solution in this case (see **SM** Figure 3 for an illustration) has rather simple dynamic evolution. Nevertheless, **MSMP-PDE** still outperforms the baseline **MP-PDE** to some extent. Ablating different components shows that the gating mechanism is of greater importance in this case than resolving temporal multi-scale behavior.

In the second experiment, labelled as **E2**, we again follow Brandstetter et al. (2022) to consider the forced viscous version of Burgers' equation **SM** equation 8 with the same sinusoidal initial conditions as in **E1**. In this case, training and test samples are generated by varying both the initial conditions as well as the time dependent source terms, leading to significantly more complicated dynamic behavior as compared to **E1** (see **SM** Figure 4 for an illustration). Consequently, the test errors for each of the models, reported in Table 1 are much higher. Again, **MSMP-PDE** outperforms the baseline **MP-PDE** significantly, with the test error being reduced by almost 14%. Ablating models shows that adding both gating and temporal multi-scale resolution helps reduce the error, with gating playing a bigger role.

For the final experiment that we label as **MS-wave**, we consider a one-dimensional wave equation, written as a linear hyperbolic system (**SM** equation 10), that was proposed in Hiltebrand & Mishra (2017) to test numerical methods for multi-scale problems. The initial conditions, given in the form of sinusoidal data (**SM** equation 9) evolve in the form of waves that propagate at different speeds. The training (and test) samples are generated from different initial conditions as well as wave speeds (see details in **SM** A.1.2). The test errors with the competing models is reported in Table 1 and we observe from this table that the errors are quite high for this problem given the fact that the models have to learn dynamics at different temporal and spatial scales. For instance, the baseline **MP-PDE** algorithm of Brandstetter et al. (2022) yields errors greater than 20%. On the other hand, adding both temporal and spatial multi-scale resolution features in our proposed **MSMP-PDE** model leads to large (factor of 2) decrease in this error. Ablating components shows that both gating and a LEM encoder seem necessary to obtain the best results.

Thus, with these numerical experiments, we have demonstrated that adding temporal and spatial multiscale resolution capabilities to a message passing neural PDE solver significantly enhances its abilities in approximating PDEs, particularly those with multiple scales, accurately. This study paves the way for the further development of operator learning models that can deal with multi-scale data on both structured and unstructured grids.

REFERENCES

P.W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, and R. Faulkner et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Oussama Boussif, Yoshua Bengio, Loubna Benabbou, and Dan Assouline. MAgnet: Mesh agnostic neural PDE solver. In *Advances in Neural Information Processing Systems*, 2022.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

A. Hiltebrand and S. Mishra. Efficient computation of all speed flows using an entropy stable shock-capturing space-time discontinuous galerkin method. In *Partial Differential Equations, Mathematical Physics and Stochastic analysis.* EMS Congress Reports, 2017.

Christian Kuehn. *Multiple time scale dynamics*, volume 191. Springer, 2015.

Randall J. LeVeque. *Numerical methods for conservation laws (2. ed.).* Lectures in mathematics. Birkhäuser, 1992.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

M. Lino, C. Cantwell, A. Bharath, and S. Fotiadis. Simulating continuum mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2106.04900*, 2021.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Kjetil O Lye, Siddhartha Mishra, and Deep Ray. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics*, pp. 109339, 2020.

A. Quarteroni and A. Valli. *Numerical approximation of Partial differential equations*, volume 23. Springer, 1994.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

T Konstantin Rusch, Siddhartha Mishra, N Benjamin Erichson, and Michael W Mahoney. Long expressive memory for sequence modeling. In *International Conference on Learning Representations*, 2022.

T. Konstantin Rusch, Benjamin P. Chamberlain, Michael W. Mahoney, Michael M. Bronstein, and Siddhartha Mishra. Gradient gating for deep multi-rate learning on graphs. In *International Conference on Learning Representations*, 2023.

Kimberly Stachenfeld, Drummond B. Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation. In *International Conference on Learning Representations*, 2022.

Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 336:415–447, 2018.

Supplementary Material for:
Multi-scale message passing neural PDE solvers.

A.1 EXPERIMENTS

A.1.1 BURGER'S EQUATION (**E1/E2**)

As a first benchmark, we consider Burger's equation with variable viscosity and forcing, i.e.,

$$\left[\partial_t u + \partial_x \left(u^2 - \beta \partial_x u\right)\right](t,x) = \alpha f(t,x) \tag{8}$$

$$u(t,0) = u(t,L), \quad u(0,x) = f(0,x), \quad f(t,x) = \sum_{j=1}^{J} A_j \sin\left(\omega_j t + 2\pi \ell_j x/L + \phi_j\right), \tag{9}$$

and perform two experiments:

- **E1** Inviscid Burger's Equation ($\alpha = \beta = 0$, i.e., no forcing and no diffusion).
- **E2** Burger's Equation with variable viscosity and forcing ($\beta \in [0, 0.2]$ and $\alpha = 1$).

We generate the corresponding training, validation and test sets based on the numerical solver from Brandstetter et al. (2022). Thereby, a numerical ground truth is generated using a WENO5 scheme for the convection term and a fourth order finite difference for the diffusion term. The space discretization is then integrated with an explicit Runge-Kutta solver (RK4) with adaptive timestepping. The numerical ground truth is generated with $t \in [0, 4]$ and $\Omega = [0, 16]$ on a $(n_t, n_x) = (250, 200)$-grid. We further compute the training data by down-sampling with a 1D convolution operator to obtain a $(n_t, n_x) = (250, 100)$-grid.

Initial conditions and forcing term are defined as f in equation 9 with $A_j \sim U([-1/2, 1/2]) \quad \phi_j \sim U([0, 2\pi]), \quad \omega_j \sim U([-0.4, 0.4]) \quad$ and $\quad l_j \in \{1, 2, 3\}$[2].

A.1.2 LINEAR ADVECTION SYSTEM (**MS-WAVE**)

In this experiment, we focus on the following two-speed advection problem in order to test the ability of the models to effectively learn multi-scale properties,

$$\partial_t \mathbf{u} + \mathbf{A}\partial_x \mathbf{u} = 0, \quad \mathbf{u}(t,x) = \begin{pmatrix} u^{(1)}(t,x) \\ u^{(2)}(t,x) \end{pmatrix}, \quad \mathbf{u}(t,0) = \mathbf{u}(t,L). \tag{10}$$

Since equation 10 is a linear hyperbolic system, an analytic solution can be efficiently computed to generate training instances. To this end, in this experiment we consider,

$$\mathbf{A} = \begin{pmatrix} a+b & b-a \\ b-a & a+b \end{pmatrix}, \quad a, b \in \mathbb{R}. \tag{11}$$

Let $\mathbf{A} = \mathbf{R}\mathbf{\Lambda}\mathbf{R}^{-1}$ be its eigendecomposition with,

$$\mathbf{\Lambda} = \begin{pmatrix} 2a & 0 \\ 0 & 2b \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \text{and} \quad \mathbf{R}^{-1} = \begin{pmatrix} -1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}. \tag{12}$$

Since $\mathbf{\Lambda}$ is diagonal, equation 10 can be written as a system of two uncoupled advection equations,

$$\partial_t \mathbf{w} + \mathbf{\Lambda}\partial_x \mathbf{w} = 0, \quad \text{with} \quad \mathbf{w} = \mathbf{R}^{-1}\mathbf{u}. \tag{13}$$

Given an initial condition $\mathbf{u}_0(x)$, we can define the initial condition in the eigenbasis,

$$\mathbf{w}_0(x) = \mathbf{R}^{-1}\mathbf{u}_0(x) = \begin{pmatrix} w_0^{(1)}(x) \\ w_0^{(2)}(x) \end{pmatrix}. \tag{14}$$

[2] $U([a, b])$ denotes the uniform distribution with support $[a, b]$.

We can then use the solution of scalar conservation laws by the method of characteristics (LeVeque, 1992) to get the solution of the system,

$$\mathbf{u}(t,x) = \mathbf{R}\mathbf{w}(t,x) = \mathbf{R}\begin{pmatrix} w_0^{(1)}(x-2at) \\ w_0^{(2)}(x-2bt) \end{pmatrix}. \tag{15}$$

We generate a training, validation and test set using the aforementioned method. The numerical ground truth is generated with $\Omega = [0, 16]$ and $t \in [0, 4]$ on a $(n_t, n_x) = (250, 200)$-grid and the training data is computed by downsampling to a $(n_t, n_x) = (250, 100)$-grid. Initial conditions $u_0^{(1)}$ and $u_0^{(2)}$ are defined as f in equation 9 with $A_j \sim U([-1/2, 1/2])$ $\phi_j \sim U([0, 2\pi])$ and $l_j \in \{1, 2, 3\}$. The equation parameters are sampled as $a \sim U([0.1, 1])$ and $b \sim U([1, 10])$ such that two very different scales can be represented.

A.2 Network Architecture

In this section, we briefly describe the model implementation.

Encoder. In the scalar case, the input to the encoder is given by the $(K + \dim(\eta) + 2)$-dimensional vector $[\mathbf{u}_i^{k-K:k}, x_i, t_k, \eta]$. We distinguish between two different types of encoders: (i) a simple two-layer feedforward neural network (with swish activation function) for **MP-PDE**, (ii) a one-layer LEM (or LSTM) for all other methods considered here, where the 3-dimensional sequence $\{[u_i^{k-K+l}, x_i, \eta]^\top\}_{l=0}^{K-1}$ of length K gets recurrently processed and only the final hidden state of the RNN gets further propagated through a 2-layer feedforward neural network (with swish activation) before it gets passed to the message-passing processor.

Processor. The processor relies on the node feature vectors $\{\mathbf{X}_i^n\}_i = \mathbf{X}^n$ and graph $\mathcal{G}$ to perform the following message passing operations:

$$\begin{aligned} \text{edge } j \to i \text{ message:} \quad & \mathbf{m}_{ij}^n = \phi_\theta\left(\left[\mathbf{X}_i^{n-1}, \mathbf{X}_j^{n-1}, \mathbf{u}_i^{k-K:k} - \mathbf{u}_j^{k-K:k}, x_i - x_j, \eta\right]\right), \\ \text{node } i \text{ update:} \quad & \mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})_i = \psi_\theta\left(\left[\mathbf{X}_i^{n-1}, \textstyle\sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}^n, \eta\right]\right), \end{aligned} \tag{16}$$

where ϕ and ψ are 2-layer feedforward neural networks with Swish activation function. For the non-gated models, the node vectors are simply updated as $\mathbf{X}^n = \mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})$. In contrast to that, for our proposed gated GNN model we need to construct two MPNNs, i.e., $\mathbf{F}_\theta$ and $\mathbf{F}_{\hat{\theta}}$ each defined as in equation 16. We then use the propagation rule in equation 7 to update the node feature vectors.

Decoder. The decoder is based on a 2-layer convolutional neural network (CNN) to project the node feature outputs of the final processor layer back to the input-output dimension of the underlying PDE. To this end, we follow Stachenfeld et al. (2022); Brandstetter et al. (2022) and output the difference of the current to the new timestep solution $d_i = \left(d_i^1, d_i^2, \ldots, d_i^K\right)$, i.e., the final node-wise update of our network is,

$$u_i^{k+\ell} = u_i^k + (t_{k+\ell} - t_k)\, d_i^\ell, \quad 1 \le \ell \le K. \tag{17}$$

We summarize the architectures used for 1D scalar PDEs (experiments **E1/E2**) in table 2. In practice we fix $N_{\text{hid}} = 128$ with 6 processor hidden layers (N_{layers}).

In order to accommodate 1D systems of N equations (as in experiment **MS-wave** where $N = 2$) we add a linear layer at the beginning of the decoder that maps the hidden dimension to a vector of size $N \times K$ such that we can recover the N K-lagged output vectors.

A.3 Training and Testing Details

The training and inference of the models is done on an NVIDIA GeForce RTX 2080 Ti, where the training takes between 24h and 65h depending on the model complexity.

Table 2: Architecture summary of the tested models.

Model	Encoder	Processor	Decoder
MP-PDE	Linear-Swish-Linear-Swish	6 × MPNN	Conv1D-Swish-Conv1D
LSTM	LSTM-Linear-Swish-Linear-Swish	6 × MPNN	Conv1D-Swish-Conv1D
LEM	LEM-Linear-Swish-Linear-Swish	6 × MPNN	Conv1D-Swish-Conv1D
Gated	Linear-Swish-Linear-Swish	6 × Gated MPNN	Conv1D-Swish-Conv1D
LSTMGated	LSTM-Linear-Swish-Linear-Swish	6 × Gated MPNN	Conv1D-Swish-Conv1D
MSMP-PDE	LEM-Linear-Swish-Linear-Swish	6 × Gated MPNN	Conv1D-Swish-Conv1D

Table 3: Number of model trainable parameters per experiment.

Model	E1	E2	MS-wave
MP-PDE	634'745	636'409	693'738
LSTM	715'769	717'817	772'842
LEM	715'257	717'305	772'330
Gated	1'249'145	1'252'345	1'330'410
LSTMGated	1'330'169	1'333'753	1'409'514
MSMP-PDE	1'329'657	1'333'241	1'409'002

A.3.1 Dataset and Errors

In all experiments the domain is given by $\Omega = [0, 16]$ and $t \in [0, 4]$ with $(n_t, n_x) = (250, 100)$ and $K = 25$.

For experiments **E1/E2** we create a training, validation and test set of sizes 2048, 128 and 128 respectively (1024,128,128 for **MS-wave**). We perform a 5-fold cross validation to get estimates of the standard deviation associated to changing the dataset and resampling the initial weights. The models are trained using the method described in section A.3.2, we use the validation dataset to perform early stopping and report the errors obtained on the test set.

Let u^i and u^i_θ be the ground truth solution and the network prediction for the test sample i, we report in table 1 the relative error,

$$\text{RE} = \frac{\frac{1}{N_{\text{sample}}} \sum_{i=1}^{N_{\text{sample}}} \|u^i_\theta - u^i\|_{L^2(\Omega\times]0,T])}}{\frac{1}{N_{\text{sample}}} \sum_{i=1}^{N_{\text{sample}}} \|u^i\|_{L^2(\Omega\times]0,T])}}, \tag{18}$$

which in practice is computed by unrolling the full trajectory with the network of interest and by computing the norms in discretized form.

A.3.2 Autoregressive Training Details

We follow the training procedure of Brandstetter et al. (2022), i.e., the networks are trained for 20 epochs (with early stopping), we use a batch size of 16 and a learning rate of $1\text{e}{-4}$, which is reduced by a factor of 0.4 for every 5 epochs. Moreover, we train the models using the AdamW optimizer with a root mean squared loss. We define the underlying computational graph using a radius around the node of interest such that 3 nearest neighbors on each side are connected to the node.

The training procedure is illustrated in Figure 2. We start by grouping each of the temporal sequences of the training set to a set of K-lagged solutions. We further randomly choose one of the K-lagged solutions as well as the number of unrolling steps to be performed (in this paper we use a maximum unrolling depth of 2). After unrolling, the loss is computed between the model output and the ground truth solution, where the errors are then backpropagated only through the last model call in order to mitigate the distribution shift problem.

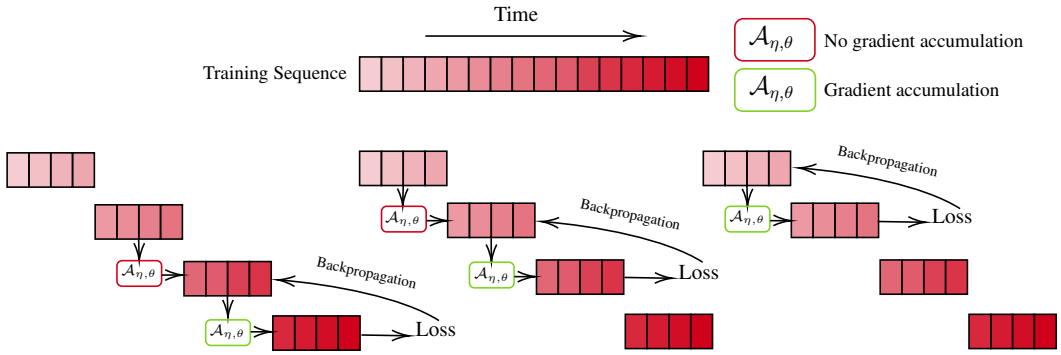

Figure 2: Example of three autoregressive training input-output instances with K-lagged solutions, truncated backpropagation and a maximum unrolling depth of 2.

A.4 PLOTS

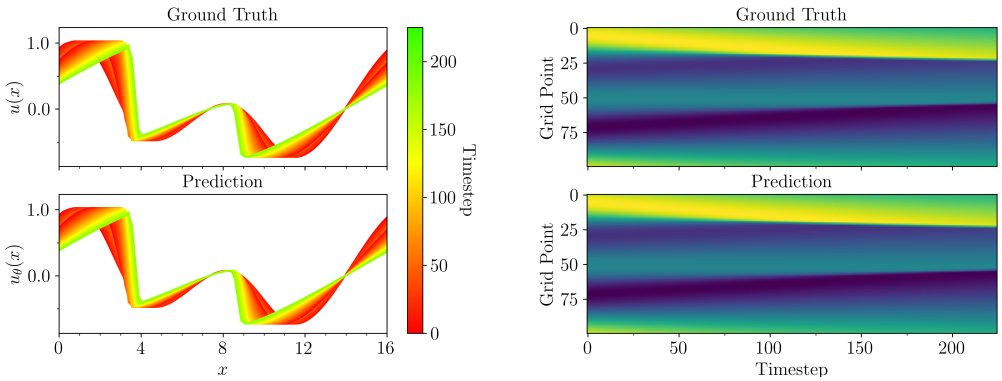

Figure 3: Experiment **E1**, MSMP-PDE prediction

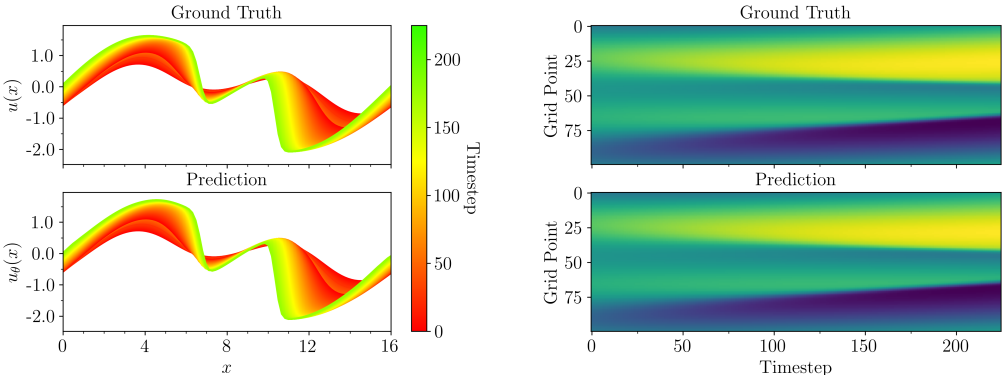

Figure 4: Experiment **E2**, MSMP-PDE prediction

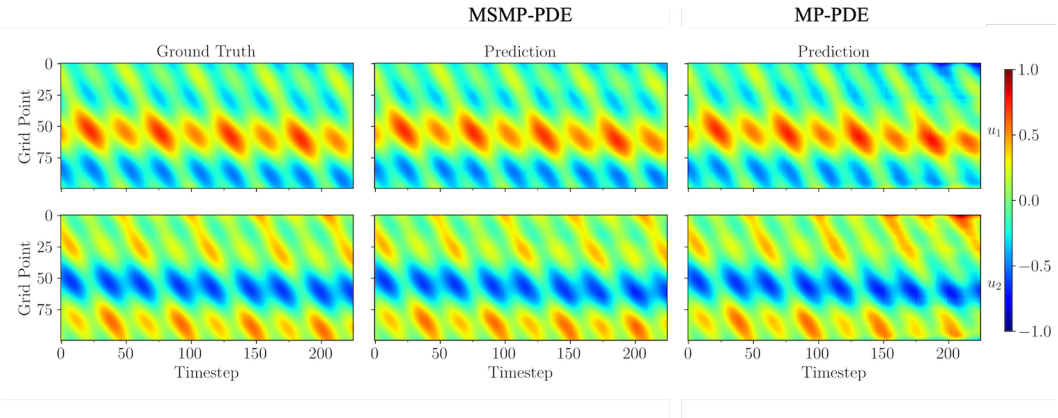

Figure 5: Experiment **MS-wave** with equation parameters $a = 0.21$ and $b = 9.42$. MSMP-PDE shows a relative L^2 error of 9.6% while MP-PDE shows a relative L^2 error of 26.7%.

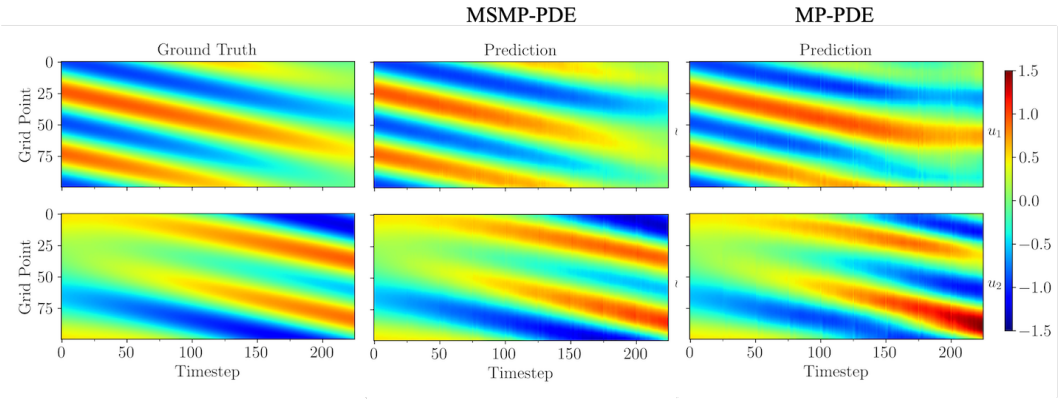

Figure 6: Experiment **MS-wave** with equation parameters $a = 0.87$ and $b = 1.31$. MSMP-PDE shows a relative L^2 error of 14.05% while MP-PDE shows a relative L^2 error of 44.8%.