

---

# Neuro-Spectral Architectures for Causal Physics-Informed Networks – Supplementary Material –

---

Anonymous Author(s)

Affiliation

Address

email

## 1 A Causality of NeuSA

2 NeuSA is a “causal” architecture in the sense that it produces solutions which are *evolutionary*  
 3 by nature, implying uniqueness and continuous dependence on initial conditions. It is generally  
 4 impossible to prove the convergence of a numerical method to the solution of a general PDE without  
 5 imposing strong restrictions upon the spectrum of initial conditions and the function  $\mathbf{F}$ . We may  
 6 nevertheless prove that the solutions generated by our method have the properties associated with  
 7 solutions to evolution problems. For the finite-dimensional system in eq. (4), these properties  
 8 are encapsulated in the properties of the associated *flow operator*  $\Phi : [0, T] \times \mathbb{C}^{c_1 \times \dots \times c_d \times n} \rightarrow$   
 9  $\mathbb{C}^{c_1 \times \dots \times c_d \times n}$ , defined as:

$$\Phi^t \hat{\mathbf{u}}_0 = \hat{\mathbf{u}}_0 + \int_0^t \hat{\mathbf{F}}(\hat{\mathbf{u}}(\tau)) d\tau. \quad (14)$$

10 These operators have the following semigroup properties:

- 11 1. There exists an identity element:  $\Phi^0 \hat{\mathbf{u}}_0 = \hat{\mathbf{u}}_0$ .
- 12 2. The flow is an additive group action:  $\Phi^{t_2} \Phi^{t_1} \hat{\mathbf{u}}_0 = \Phi^{t_2+t_1} \hat{\mathbf{u}}_0$ .

13 Most importantly, these two properties translate immediately to the causal properties that we aim  
 14 to prove: Property 1 implies that initial conditions are enforced, while Property 2 is equivalent to  
 15 unicity [7]. Theorem 1 then follows as a consequence of the fact that NeuSA solutions define a flow.

16 Let  $S_{\mathbf{b}} \subset L^2(\Omega)$  be the finite-dimensional subspace spanned by our basis elements  $\mathbf{b}$ . Consider  
 17 now the decomposition operator  $P : S_{\mathbf{b}} \rightarrow \mathbb{C}^{c_1 \times \dots \times c_d \times n}$ , an isomorphism mapping each element  
 18 of  $S_{\mathbf{b}}$  into the tensor of coefficients of its expansion over the elements of  $\mathbf{b}$ . We also define the  
 19 reconstruction operator  $P^\dagger : \mathbb{C}^{c_1 \times \dots \times c_d \times n} \rightarrow S_{\mathbf{b}}$  mapping the coefficient tensor onto the respective  
 20 linear combination of the vectors  $\mathbf{b}_k$ . Inference for NeuSA consists of the following steps:

- 21 1. obtaining expansion coefficients  $\hat{\mathbf{u}}_0 = P\mathbf{u}_0$  of the initial data in the basis  $\mathbf{b}$  (possibly,  
 22 projecting  $\mathbf{u}_0$  onto  $S_{\mathbf{b}}$ );
- 23 2. acting on  $\mathbf{u}_0$  by the flow of the vector field  $\mathbf{F}_\theta$  using NODE;
- 24 3. reconstructing the continuous solution via  $\mathbf{u}(t) = P^\dagger \hat{\mathbf{u}}(t)$ .

25 These steps can be summarized as

$$\mathbf{u}_\theta(t) = P^\dagger \Phi_\theta^t P \mathbf{u}_0, \quad \text{where} \quad \Phi_\theta^t \hat{\mathbf{u}}_0 = \hat{\mathbf{u}}_0 + \int_0^t \hat{\mathbf{F}}_\theta(\hat{\mathbf{u}}(\tau)) d\tau. \quad (15)$$

26 Now we can formulate and prove the following

**Theorem 1.** For band-limited initial conditions  $\mathbf{u}_0 \in S_{\mathbf{b}}$  and globally-Lipschitz neural vector fields  $\hat{\mathbf{F}}_{\theta}$ , the orbits created by NeuSA obey initial conditions and are unique:

1. fulfillment of initial conditions:  $\mathbf{u}_{\theta}(0, \mathbf{x}) = \mathbf{u}(0, \mathbf{x})$ ;
2. uniqueness:  $\mathbf{u}_{\theta}^1(0, \cdot) \neq \mathbf{u}_{\theta}^2(0, \cdot) \implies \mathbf{u}_{\theta}^1(t, \cdot) \neq \mathbf{u}_{\theta}^2(t, \cdot) \quad \forall t \in [0, T]$ .

*Proof.* For band-limited functions  $\mathbf{u} \in S_{\mathbf{b}}$ , the decomposition  $P$  and reconstruction  $P^{\dagger}$  are bijections, and  $P^{\dagger}$  is the inverse of  $P$ :

$$P^{\dagger}P\mathbf{u} = \mathbf{u}. \quad (16)$$

This fact immediately follows from uniqueness of expansion coefficients of any function  $\mathbf{u} \in S_{\mathbf{b}}$  for the given basis  $\mathbf{b}$ . Likewise, for a globally Lipschitz neural network  $\hat{\mathbf{F}}_{\theta}$ , there exists a flow  $\Phi_{\theta}$  associated with the NODE  $d\hat{\mathbf{u}}/dt = \hat{\mathbf{F}}_{\theta}(\hat{\mathbf{u}})$  (i.e., the flow determined by the vector field  $\hat{\mathbf{F}}_{\theta}(\hat{\mathbf{u}})$ ). Moreover, its orbits are unique. This follows from the classical existence and uniqueness of solutions for ordinary differential equations in Banach spaces [6, 7].

Property 1 then follows from the uniqueness of the spectral decomposition combined with the flow properties of  $\Phi_{\theta}$ :

$$\mathbf{u}_{\theta}(0) = P^{\dagger}\Phi_{\theta}^0P\mathbf{u}_0 = P^{\dagger}P\mathbf{u}_0 = \mathbf{u}_0. \quad (17)$$

Likewise, to prove property 2 we use the uniqueness of the spectral decomposition as well as the fact that NeuSA encodes the flow  $\Phi_{\theta}$  (under which the orbits do not intersect), represented as follows:

$$P^{\dagger}\Phi_{\theta}^{t_2}P\mathbf{u}_{\theta}(t_1) = P^{\dagger}\Phi_{\theta}^{t_2}PP^{\dagger}\Phi_{\theta}^{t_1}P\mathbf{u}_0 = P^{\dagger}\Phi_{\theta}^{t_2+t_1}P\mathbf{u}_0 = \mathbf{u}_{\theta}(t_2 + t_1). \quad (18)$$

□

As mentioned above, in practice decomposition over the spectral basis  $\mathbf{b}$  is preceded by the projection of the initial data onto  $S_{\mathbf{b}}$ . The basis  $\mathbf{b}$  can be always chosen to be large enough to approximate any smooth function with the required accuracy. Likewise, it is generally reasonable to assume that common (finite-sized) feedforward networks are globally Lipschitz-continuous, since they are essentially compositions of Lipschitz maps and Lipschitz nonlinear activations.

## B Implementation details

In this section, we provide further details concerning our code implementation. The construction of the Fourier basis together with the extraction of its coefficients is explained in Subsection B.1. Lastly, Subsection B.2 discusses the numerical scheme adopted to enhance dimensionwise layers for multiple space dimensions, as they enable efficient modeling of complex interactions across spatial dimensions while avoiding the prohibitive costs of fully dense layers.

### B.1 Spectral decomposition

To operate in the spectral domain, we represent the target functions using a spectral decomposition. The following subsections describe how the Fourier basis is constructed and how the corresponding coefficients are extracted.

#### B.1.1 Constructing the basis

We assume for this section that the domain  $\Omega$  may be split into a direct product (e.g., into a direct product of 1D intervals), leading to the following product representation for the basis:

$$\mathbf{b}_k(\mathbf{x}) = \prod_{i=1}^d \mathbf{b}_{k_i}(\mathbf{x}_i). \quad (19)$$

For example, the standard Fourier basis functions may be expressed as the products of the following form (up to a normalization constant):

$$\mathbf{b}_k(\mathbf{x}) \propto \prod_{i=1}^d \exp(i\omega_{k_i}\mathbf{x}_i), \quad (20)$$

or, likewise, in terms of the respective sine/cosine functions, depending on the boundary conditions that have to be imposed.

In addition to enabling our initialization procedure, initializing  $\mathbf{b}$  as the Fourier basis tends to result in non-sparse representations for nonlinear and/or non translation-invariant dynamics<sup>1</sup> [5]. While in the context of numerical solution of PDEs this is often undesirable, in the context of neural networks this will become an advantage, as denser connections between coefficients should lead to (positive) overparametrization [4].

Nevertheless, the Fourier basis has limitations. Foremost among them is its strictly non-local nature, which often results in difficulties when reproducing highly localized PDE solutions. For this reason, we make the basis  $\mathbf{b}$  learnable.

In practice, we implement learnable bases by applying learnable orthogonal maps  $O_\theta$  to the Fourier basis elements and the Fourier coefficients associated with the initial conditions  $\mathbf{u}_0$ . These maps are initialized near the identity and remain orthogonal throughout training by means of a specialized parametrization [1], which is available in most modern deep-learning frameworks. The spectral projection and reconstruction operators (see Appendix A) can then be represented as:

$$P\mathbf{u} = O_\theta P_{\text{Fourier}} \mathbf{u}, \quad P^\dagger \hat{\mathbf{u}} = P_{\text{Fourier}}^\dagger O_\theta^T \hat{\mathbf{u}}, \quad (21)$$

where  $P_{\text{Fourier}}$  and  $P_{\text{Fourier}}^\dagger$  denote the decomposition and reconstruction operators for the Fourier basis, respectively. We observed that this addition moderately increases performance, at very little additional computational cost.

### B.1.2 Extracting the coefficients $\hat{\mathbf{u}}$ and evaluating derivatives

In order to perform the spectral decomposition  $\mathbf{u} \mapsto \hat{\mathbf{u}}$ , we sample the function over  $N$  collocation points  $\{\mathbf{x}^i\}_{i=0}^{N-1}$  and represent it in terms of  $N$  basis terms. This may be expressed as:

$$\mathbf{u}(t, \mathbf{x}^i) = \sum_k \hat{\mathbf{u}}_k(t) \mathbf{b}_k(\mathbf{x}^i), \quad i \in 1, \dots, N, \quad (22)$$

where  $k$  is a  $d$ -dimensional multi-index with  $N$  elements. This is a linear system that can be solved in a straightforward manner; for example, given a grid structure, the Discrete Fourier Transform and its variations may be used. As it is usually done in the literature on spectral methods, we can simplify the basis representation by reshaping the domain via a transformation  $\chi$ , leading to:

$$\mathbf{u}(t, \mathbf{x}^i) = \sum_k \hat{\mathbf{u}}_k(t) \mathbf{b}_k(\chi(\mathbf{x}^i)). \quad (23)$$

Differentiation then takes place as described in Section 2:

$$\frac{d}{d\mathbf{x}_j} \mathbf{u}(t, \mathbf{x}^i) = \sum_k \hat{\mathbf{u}}_k(t) \frac{d}{d\mathbf{x}_j} \mathbf{b}_k(\chi(\mathbf{x}^i)), \quad (24)$$

where the chain rule is used to calculate the right-most term. For a concrete example, a 2D problem expressed in the Fourier basis over a square domain  $\Omega = [-L, L] \times [-L, L]$  may be mapped into the canonical domain  $[-\pi, \pi] \times [-\pi, \pi]$  with the transformation  $\chi(\mathbf{x}) = \pi\mathbf{x}/L$ , leading to the following form for the derivatives:

$$\frac{d}{d\mathbf{x}_j} \mathbf{u}(t, \mathbf{x}^i) = \sum_k \frac{i\pi\omega_{k_j}}{L} \hat{\mathbf{u}}_k(t) \prod_{i=1}^d \exp(i\omega_{k_i} \mathbf{x}_i). \quad (25)$$

$$\frac{d^2}{d\mathbf{x}_j^2} \mathbf{u}(t, \mathbf{x}^i) = - \sum_k \left( \frac{\pi\omega_{k_j}}{L} \right)^2 \hat{\mathbf{u}}_k(t) \prod_{i=1}^d \exp(i\omega_{k_i} \mathbf{x}_i). \quad (26)$$

<sup>1</sup>This can be easily seen from the Fourier convolution theorem as it appears when expressing nonlinear or non-translation-invariant equations in the Fourier basis: pointwise products become convolutions involving coefficients that are possibly far apart, resulting in dense connectivity matrices.

In geeneral, linear translation-invariant differential operators may be obtained as a polynomial on the frequencies  $\omega$ , leading to the following representation for the Fourier multiplier  $M$  used as part of the initialization:

$$\mathbf{F}_{\text{linear}}(\mathbf{u}, \nabla \mathbf{u}, \nabla^2 \mathbf{u}, \dots) := a_0 \mathbf{u} + \sum_i a_{1i} \frac{d}{d\mathbf{x}_i} \mathbf{u} + \sum_{i,j} a_{2ij} \frac{d^2}{d\mathbf{x}_i d\mathbf{x}_j} \mathbf{u} + \dots \quad (27)$$

96

$$\hat{\mathbf{F}}_{\text{linear}}(\hat{\mathbf{u}}) = M \odot \hat{\mathbf{u}}, \text{ with } M_k = a_0 + \sum_i a_{1i} \left( \frac{i\pi\omega_{k_i}}{L} \right) - \sum_{i,j} a_{2ij} \left( \frac{\pi^2\omega_{k_i}\omega_{k_j}}{L^2} \right) + \dots, \quad (28)$$

i.e. the  $k$ -indexed element of the multiplier  $M$  is given as a polynomial over the frequencies  $\omega$ . This multiplier may then be used for the initialization procedure described previously. For example, the Laplacian operator in 2 space dimensions may be represented as:

$$M_k = - \left( \frac{\pi^2\omega_{k_1}^2}{L^2} + \frac{\pi^2\omega_{k_2}^2}{L^2} \right). \quad (29)$$

Analogous schemes are also used for the sine and Fourier basis.

## B.2 Dimensionwise layers for multiple space dimensions

Spectral methods inherently induce dense interactions between coefficients, even across distant modes. However, in multi-dimensional settings, employing fully dense layers becomes computationally prohibitive. In particular, we can discuss the case of the 2D wave propagation problem presented in the experiments section: note that discretizing each dimension with just 100 spectral coefficients yields a total of  $10^4$  basis elements. A single dense linear layer operating on this space would then require  $O(10^8)$  parameters. As a result, when such layers are applied repeatedly (as would be done by a Neural ODE using it for a vector field), the computational cost becomes excessive.

Similar challenges in computer vision have led to the development of convolutional neural networks (CNNs), which employ finite-support kernels to create localized linear connections among neighboring nodes. However, this approach is ill-suited to our setting. First, CNNs are built to embed the translation invariance inherent to image recognition tasks, which is a property that does not generally apply to our problems. Moreover, their local receptive fields restrict long-range interactions, which are essential in spectral representations.

Instead, it is necessary to construct a layer that performs the dense and global-reaching connections associated with spectral methods while remaining parameter-light. We have opted for low-rank, *dimensionwise linear layers*, as shown in Fig 7.

These networks consist essentially of a Hadamard (element-wise) product followed by alternating dense linear transformations applied row-wise and column-wise. This structure can be efficiently implemented by combining tensor transpositions with batched matrix multiplication operations available in PyTorch and similar deep-learning frameworks (see Algorithm 1). For instance, given 2D coefficient matrices  $\hat{\mathbf{u}}(t) \in \mathbb{R}^{m \times n}$  and layer parameters  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times n}$ , and  $C \in \mathbb{R}^{m \times m}$ , this yields:

### 1. Element-wise scaling (Hadamard product):

$$\hat{\mathbf{u}} \mapsto \hat{\mathbf{u}} \odot A$$

where each element  $\hat{u}_{ij}$  is multiplied by the corresponding  $A_{ij}$ .

126

### 2. Row-wise linear transformation:

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} \mapsto \begin{bmatrix} r_1 B \\ r_2 B \\ \vdots \\ r_m B \end{bmatrix}$$

where each row vector  $r_i$  is multiplied by the matrix  $B$ .

129



130 **3. Column-wise linear transformation:**

$$[c_1 \ c_2 \ \cdots \ c_n] \mapsto [Cc_1 \ Cc_2 \ \cdots \ Cc_n]$$

131 where each column vector  $c_j$  is multiplied by the matrix  $C$ .

132

133 The matrix  $\hat{\mathbf{u}}$  can be visualized as

$$\hat{\mathbf{u}} = \begin{bmatrix} \hat{\mathbf{u}}_{1,1} & \hat{\mathbf{u}}_{1,2} & \cdots & \hat{\mathbf{u}}_{1,n} \\ \hat{\mathbf{u}}_{2,1} & \hat{\mathbf{u}}_{2,2} & \cdots & \hat{\mathbf{u}}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\mathbf{u}}_{m,1} & \hat{\mathbf{u}}_{m,2} & \cdots & \hat{\mathbf{u}}_{m,n} \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} = [c_1 \ c_2 \ \cdots \ c_n].$$

134 Each of these layers applied to a 2D spatial domain of dimensions  $m, n$  requires  $O(mn)$  parameters  
 135 and operations, in contrast to the  $O(m^2n^2)$  complexity of a naive fully dense layer. This approach  
 136 allows us to leverage the “densification” inherent to spectral methods, enhancing deep learning  
 137 performance, while reducing computational and memory demands. In practice, we have found  
 138 these layers to be effective drop-in replacements for dense linear layers when handling large, multi-  
 139 dimensional inputs.

---

**Algorithm 1** Implementation of dimensionwise layers for a two dimensional input

---

$\hat{\mathbf{u}} \leftarrow \text{input}()$	$\triangleright \hat{\mathbf{u}} \in \mathbb{R}^{m \times n}$
$\hat{\mathbf{u}} \leftarrow \hat{\mathbf{u}} \odot A$	$\triangleright$ Hadamard (pointwise) product, $A \in \mathbb{R}^{m \times n}$
$\hat{\mathbf{u}} \leftarrow \text{linear}(\hat{\mathbf{u}}, B)$	$\triangleright$ Batched matrix multiplication, $B \in \mathbb{R}^{n \times n}$
$\hat{\mathbf{u}} \leftarrow \hat{\mathbf{u}}^T$	$\triangleright$ Transpose
$\hat{\mathbf{u}} \leftarrow \text{linear}(\hat{\mathbf{u}}, C)$	$\triangleright$ Batched matrix multiplication, $C \in \mathbb{R}^{m \times m}$
$\hat{\mathbf{u}} \leftarrow \hat{\mathbf{u}}^T$	$\triangleright$ Transpose
$\hat{\mathbf{u}} \leftarrow \text{act}(\hat{\mathbf{u}})$	$\triangleright$ Nonlinear Activation

---

140 As illustrated by comparing Figs. 6 and 7, these layers stand in contrast to convolutional layers,  
 141 yet serve a complementary purpose. While convolutional layers impose sparsity through local  
 142 connectivity and translation invariance, dimensionwise linear layers aim to preserve sparsity while  
 143 retaining long-ranging connections. They can also be interpreted as low-rank tensor applications or  
 144 Kronecker products.

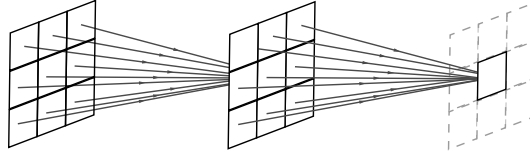


Figure 6: Schematic for convolutional linear layers.

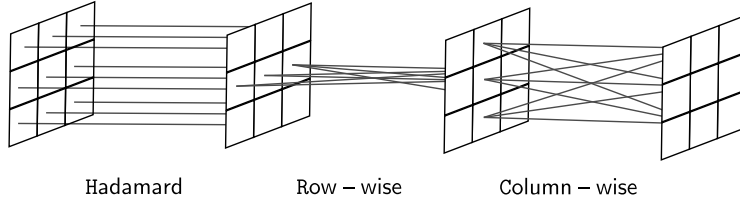


Figure 7: Schematic for dimensionwise linear transformations.

145 **C Experimental details**

146 In this section, we provide further information concerning hardwares and licensing. We also discuss  
 147 additional details on the numerical experiments presented in Section 3 of the paper.

148 **Hardware and resources.** All experiments were executed on identical machines, containing an  
 149 Nvidia RTX 4090 GPU with 24GB VRAM, an Intel i9-13900K processor, and 128GB RAM.

150 **Licenses.** The code is implemented in Python using the libraries PyTorch (version 2.1.0) and  
 151 torchdyn (version 1.0.6), distributed under the BSD 3-Clause and Apache licenses, respectively.

152 **General setup for numerical experiments.** All experiments are evaluated against a ground-truth  
 153 solution obtained using a state-of-the-art numerical method. For each model and experiment, we  
 154 report the error metrics rMAE (relative  $L_1$  error) and rMSE (relative  $L_2$  error) between the predicted  
 155 solution  $u_{\text{pred}}$  and the ground-truth solution  $u_{\text{GT}}$ , computed over  $N$  evaluation pairs  $(t_i, \mathbf{x}_i)$ , as  
 156 follows:

$$\text{rMAE} = \frac{\sum_{i=1}^N |u_{\text{pred}}(t_i, \mathbf{x}_i) - u_{\text{GT}}(t_i, \mathbf{x}_i)|}{\sum_{i=1}^N |u_{\text{GT}}(t_i, \mathbf{x}_i)|}, \quad (30)$$

$$\text{rMSE} = \sqrt{\frac{\sum_{i=1}^N (u_{\text{pred}}(t_i, \mathbf{x}_i) - u_{\text{GT}}(t_i, \mathbf{x}_i))^2}{\sum_{i=1}^N (u_{\text{GT}}(t_i, \mathbf{x}_i))^2}}. \quad (31)$$

157 In the particular case of the Burgers' equation, where the output is a 2D vector  $(u, v)$ , the rMAE and  
 158 rMSE are computed separately for each component, and their average is reported as the final error  
 159 metric.

160 Due to memory constraints, it is not feasible to feed the entire spatial grid to the MLP-based  
 161 architectures for 2D equations. Instead, we adopted random uniform sampling at every step. For  
 162 PINN, QRes, and FLS models, we sample 10,000 points for the PDE residual, 1000 points for the  
 163 initial condition and 500 points for the boundary condition. In contrast, PINNsFormer generates a  
 164 temporal sequence of five collocation points for each sample, which are then processed through its  
 165 encoder-decoder architecture. To ensure comparable memory usage and training time with the other  
 166 baseline methods, we train PINNsFormer on the Burgers' and wave equations using 2,000 points for  
 167 the PDE loss, 200 for the initial condition, and 100 for the boundary condition. This corresponds to  
 168 one-fifth of the amount sampled for other baseline methods.

### 169 C.1 2D wave equation

170 For the wave equation, by introducing  $v = \frac{\partial u}{\partial t}$ , we can rewrite the problem as a system of first-order  
 171 equations (with respect to time) given by

$$\begin{cases} \frac{\partial}{\partial t} u = v, \\ \frac{\partial}{\partial t} v = c^2(\mathbf{x}) \Delta u. \end{cases} \quad (32)$$

172 The vector field to be learned by NeuSA is defined as:

$$\hat{\mathbf{F}}_{\theta}(\hat{\mathbf{u}}) = \left( M \odot \hat{u} + \epsilon \mathcal{F}_{\theta}(\hat{u}) \right), \quad (33)$$

173 where  $\hat{\mathbf{u}} = (\hat{u}, \hat{v})$ , the weight  $\epsilon = 1$  and the entries of the matrix  $M$  as defined in (29). Using a  
 174 simplified version of  $\mathcal{F}_{\theta}$  that takes only  $\hat{u}$  as input accelerates learning, as  $\hat{v}$  does not influence the  
 175 equation, and also enables a more compact model. The heterogeneous three-layer medium, depicted  
 176 in Fig. 8(a), has its velocity field defined as

$$c(x, y) = 1.0 + 0.25(\text{sigmoid}(1000(y - 0.5)) + \text{sigmoid}(1000(y - 1))). \quad (34)$$

177 To train the NeuSA model, we simulate an infinite domain by extending the original spatial region  
 178 from  $[-2, 2] \times [-2, 2]$  to  $[-4, 4] \times [-4, 4]$ . The extensions of the original domain are designed to  
 179 ensure that, within the simulation's temporal window, any reflected waves do not re-enter the original  
 180 region of interest. This effectively prevents boundary artifacts from interfering the solution. We use  
 181  $201 \times 201$  basis elements and integrate over 201 time steps, creating a grid with a spatial step of

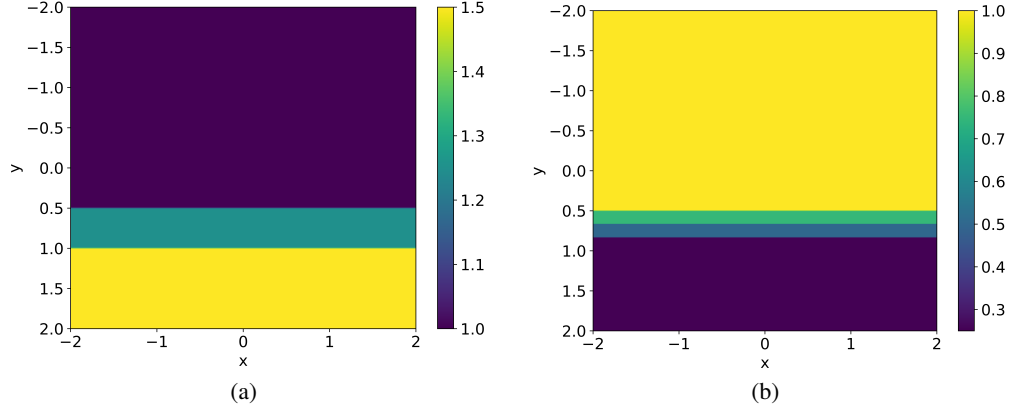


Figure 8: The heterogeneous media with three and four layers for the wave propagation experiments.

182  $\Delta x = \Delta y = 0.04$  and a temporal step of  $\Delta t = 0.01$ . For the spectral decomposition, we adopt a  
 183 cosine basis.

184 For the baseline models, we set the initial condition weight to  $10^3$  (on both Dirichlet boundary  
 185 condition and first-order initial condition. See Eq. (11) in the main text). This weighting led to  
 186 consistently improved accuracy across experiments.

187 The ground-truth solution is calculated using a standard finite central difference scheme for the  
 188 second derivatives with order 8 in the space and order 2 in time [3], with a spatial step of 0.01 and a  
 189 time step of 0.001. The domain was also extended to simulate an infinite domain.

190 We conduct a second experiment on the wave equation, considering a four-layer heterogeneous  
 191 medium, as shown in Fig. 8(b). These layers are chosen to create a more interesting and complex  
 192 wave-field pattern. Qualitative results shown in Fig. 9 indicate that our architecture faces significant  
 193 challenges in accurately simulating wave propagation in more complex media, particularly where  
 194 thin overlapping layers are present. Nevertheless, our method still achieves the lowest rMSE among  
 195 all models, highlighting its potential.

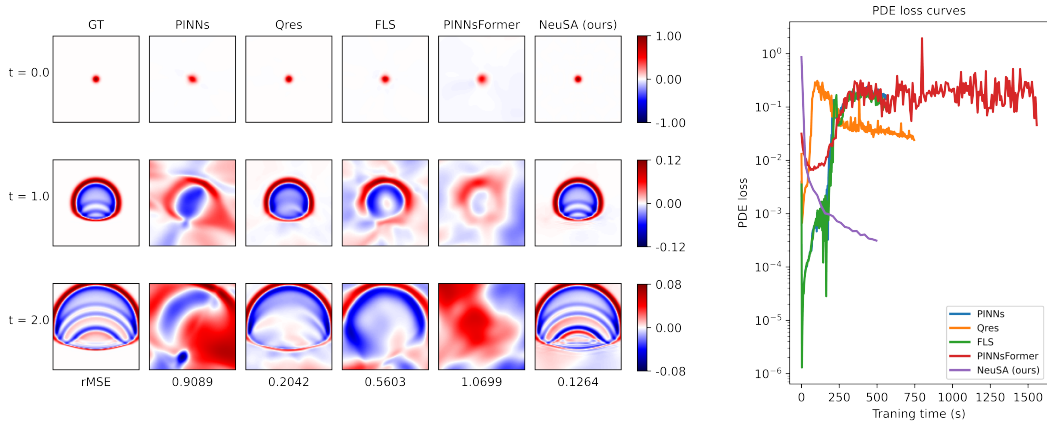


Figure 9: Time snapshots of the solution of the wave equation for a four-layer medium for each model (left), and the evolution of the PDE loss over training time for each model (right). As NeuSA does not need to learn the initial condition, the PDE loss curves on the beginning of the training show that, while NeuSA is learning how to satisfy the PDE loss, the other models are still trying to learn the initial condition. Therefore, during the early stages of training, the PDE loss in the baseline models is not directly connected to the model’s convergence, making the training slower.

## 196 C.2 1D sine-Gordon equation

197 Let us define again  $v = \frac{\partial u}{\partial t}$ , which allows us to rewrite the problem as a system of first-order  
 198 equations (with respect to the time derivative) given by

$$\begin{cases} \frac{\partial}{\partial t} u = v, \\ \frac{\partial}{\partial t} v = \Delta u - 10 \sin(u). \end{cases} \quad (35)$$

199 From this and Eq. (12), the vector field to be learned from NeuSA can be defined as:

$$\hat{\mathbf{F}}_{\theta}(\hat{\mathbf{u}}) = \begin{pmatrix} \hat{v} \\ M \odot \hat{u} + \epsilon \mathcal{F}_{\theta}(\hat{u}) \end{pmatrix} \quad (36)$$

200 where the weight  $\epsilon$  is set to 0.1, and the entries of the matrix  $M$  are taken as defined in (29).

201 Owing to its architectural design, NeuSA automatically satisfies the initial condition, while the  
 202 use of a sine basis ensures compliance with the boundary condition. For the baseline models, we  
 203 adopted a weight of  $10^3$  to the initial condition, as this choice yielded the most accurate results in our  
 204 preliminary experiments.

205 In contrast to the experiments involving 2D equations, training for the sine-Gordon equation can be  
 206 performed using the full spatial grid at each step. Since PINNsFormer produces 5 training points per  
 207 grid location, it is trained on a coarser regular grid of size  $101 \times 101$ , whereas all other models are  
 208 trained on a finer  $201 \times 201$  grid. This choice of grid size presented a reasonable balance of training  
 209 time, accuracy, and memory usage.

210 The ground-truth solution was computed through a pseudo-spectral method combined with a 4th-order  
 211 Runge-Kutta integrator. We used a time step of  $\Delta t = 0.001$  and a spatial resolution of  $\Delta x = 0.04$ .

## 212 C.3 2D Burgers' equation

213 Since the Burgers' equation is a first-order partial differential equation in time, its conversion into a  
 214 system of nonlinear parabolic equations results in

$$\begin{cases} \frac{\partial}{\partial t} u = \nu \Delta u - u \frac{\partial u}{\partial x} - v \frac{\partial u}{\partial y}, \\ \frac{\partial}{\partial t} v = \nu \Delta v - u \frac{\partial v}{\partial x} - v \frac{\partial v}{\partial y}. \end{cases} \quad (37)$$

215 From this and (13), the vector field for NeuSA can be directly defined by

$$\hat{\mathbf{F}}_{\theta}(\hat{\mathbf{u}}) = \begin{pmatrix} \nu M \odot \hat{u} + \epsilon \mathcal{F}_{\theta}^u(\hat{u}, \hat{v}) \\ \nu M \odot \hat{v} + \epsilon \mathcal{F}_{\theta}^v(\hat{u}, \hat{v}) \end{pmatrix}, \quad (38)$$

216 where  $M$  is given in (29), and the weight  $\epsilon$  is set to 0.1. Also,  $\mathcal{F}_{\theta}^u(\hat{u}, \hat{v})$  and  $\mathcal{F}_{\theta}^v(\hat{u}, \hat{v})$  are the neural  
 217 networks whose parameters we optimize during training, so the loss function is composed by the  
 218 residues of each of them. The equal subscript  $\theta$  on both is a slight abuse of the notation, since the  
 219 parameters of these networks are not the same.

220 NeuSA inherently satisfies the initial conditions by construction (Theorem 1), and the use of a Fourier  
 221 basis for the Burgers' equation ensures that periodic boundary conditions are also met automatically.  
 222 For the baseline models, we set the initial and boundary conditions weights to  $\lambda_{IC} = 10^2$  and  
 223  $\lambda_{BC} = 1$ , as these values produce the best performance in preliminary experiments.

224 The results presented in Table 1 for the Burgers' equation correspond to model training and evaluation  
 225 over the time interval  $[0, 1]$ . We compare them and the extrapolation experiment with a ground-truth  
 226 obtained through a pseudo-spectral method integrated with a 4th-order Runge-Kutta for the time  
 227 interval  $[0, 2]$ . The temporal and spatial discretizations are set to  $\Delta t = 0.001$  and  $\Delta x = \Delta y = 0.02$ ,  
 228 respectively. Figure 10 provides a visual comparison of the predicted solutions and their corresponding  
 229  $L_2$  errors. Notably, NeuSA, FLS, and QRes closely replicate the qualitative behavior of the ground-  
 230 truth solution, while vanilla PINNs only approximate the solution, and PINNsFormer fails to converge.

Table 2: 2D Wave: mean and standard deviation of rMAE, rMSE and runtime (TT).

Model	rMAE		rMSE		TT	
	mean	std	mean	std	mean	std
PINN	$7.66 \times 10^{-1}$	$1.27 \times 10^{-1}$	$5.45 \times 10^{-1}$	$8.31 \times 10^{-2}$	566	7.2
QRes	$1.54 \times 10^{-1}$	$3.45 \times 10^{-2}$	$1.15 \times 10^{-1}$	$3.10 \times 10^{-2}$	750	2.4
FLS	$8.87 \times 10^{-1}$	$1.63 \times 10^{-1}$	$5.90 \times 10^{-1}$	$9.96 \times 10^{-2}$	577	5.5
PINNsFormer	1.56	$2.78 \times 10^{-1}$	1.07	$1.41 \times 10^{-1}$	1,484	3.4
NeuSA (ours)	$1.02 \times 10^{-1}$	$3.42 \times 10^{-2}$	$7.49 \times 10^{-2}$	$2.24 \times 10^{-2}$	530	1.4

Table 3: 1D Sine–Gordon: mean and standard deviation of rMAE, rMSE and runtime (TT).

Model	rMAE		rMSE		TT	
	mean	std	mean	std	mean	std
PINN	$1.70 \times 10^{-1}$	$3.31 \times 10^{-2}$	$1.39 \times 10^{-1}$	$4.70 \times 10^{-3}$	976	41.4
QRes	$2.58 \times 10^{-2}$	$4.50 \times 10^{-3}$	$1.99 \times 10^{-2}$	$2.80 \times 10^{-3}$	1,315	49.0
FLS	$1.43 \times 10^{-1}$	$7.20 \times 10^{-3}$	$1.35 \times 10^{-1}$	$2.01 \times 10^{-2}$	1,015	68.3
PINNsFormer	$7.85 \times 10^{-1}$	$4.95 \times 10^{-1}$	$6.81 \times 10^{-1}$	$3.31 \times 10^{-1}$	3,333	186.6
NeuSA (ours)	$1.23 \times 10^{-3}$	$1.49 \times 10^{-5}$	$9.16 \times 10^{-4}$	$5.98 \times 10^{-6}$	215	1.3

Table 4: 2D Burgers: mean and standard deviation of rMAE, rMSE and runtime (TT).

Model	rMAE		rMSE		TT	
	mean	std	mean	std	mean	std
PINN	$1.65 \times 10^{-1}$	$8.91 \times 10^{-2}$	$2.21 \times 10^{-1}$	$9.30 \times 10^{-2}$	871	2.9
QRes	$2.32 \times 10^{-2}$	$1.90 \times 10^{-3}$	$7.27 \times 10^{-2}$	$1.60 \times 10^{-3}$	1,135	2.3
FLS	$1.47 \times 10^{-1}$	$7.68 \times 10^{-2}$	$2.02 \times 10^{-1}$	$8.08 \times 10^{-2}$	885	0.3
PINNsFormer	1.06	$1.91 \times 10^{-1}$	1.05	$1.71 \times 10^{-1}$	2,294	108.1
NeuSA (ours)	$7.66 \times 10^{-2}$	$1.96 \times 10^{-2}$	$1.15 \times 10^{-1}$	$1.71 \times 10^{-2}$	62	0.2

## C.4 Quantitative results

The 2D experiments (Burgers’ and wave equations) were run with 7 different seeds each, while the 1D sine-Gordon experiment was run with 3 different seeds. The relative  $L_1$  and  $L_2$  metrics and training times (in seconds) presented in the paper are an average over all random seeds. In Tables 2, 3, and 4, we list the mean and standard deviation of these values for the three main experiments reported in the paper, namely, that of 2D wave equation in Section 3.1, 1D sine-Gordon equation in Section 3.2, and 2D Burgers’ equation in Section 3.3, respectively.

## D Limitations

As mentioned in Section 2.1, NeuSA requires the initialization of the linear model, which may be not obvious. For Burgers’ equation, for example, we have found the best results initializing NeuSA near solution to the associated heat equation. As is well known, however, some linear PDEs may turn out to be stiff when expressed in the spectral domain.

This stiffness may introduce instability into the integration of the Neural ODE. When we apply spectral decompositions to the Korteweg–de Vries (KdV) equation [2], for example, the resulting systems of ordinary differential equations becomes essentially stiff, especially when a wide range of frequencies is considered. This will lead to excessively high values of the eigenvalues of the matrix used to implement the associated solver, and may resolve with a numerically unstable solution.

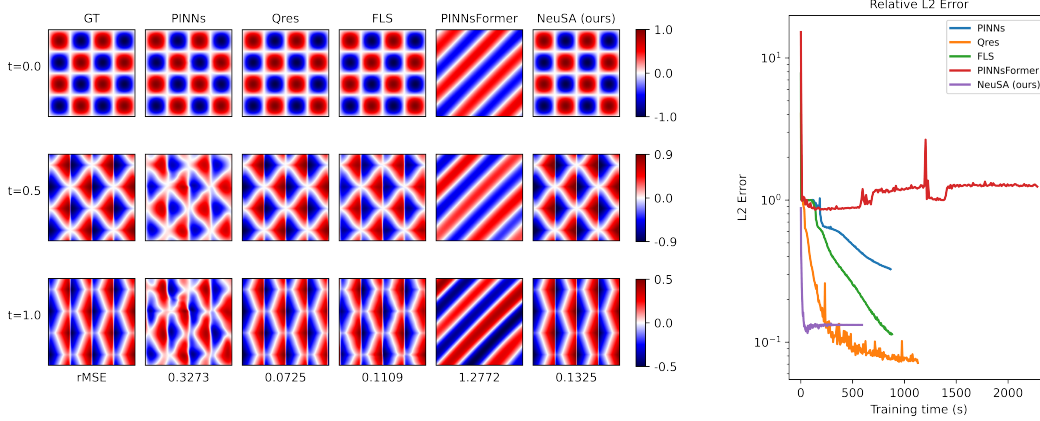


Figure 10: Time snapshots of the  $u$ -solution of the Burgers' equation for each model (left), and the evolution of the  $L_2$  error over training time for each model (right).

248 This numerical issue does not arise in a wide range of equations of interest. Consider, for instance,  
 249 the canonical example of the wave equation:

$$u_{tt} - \Delta u = 0. \quad (39)$$

250 After Fourier Transform

$$\frac{d}{dt} \begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} = A \begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix}, \quad \text{where } A = \begin{bmatrix} 0 & I \\ D & 0 \end{bmatrix} \quad \text{and} \quad D = -\text{diag}(1, \dots, N^2). \quad (40)$$

251 The block matrix  $A$  has eigenvalues  $\lambda = \pm ik$ , for  $k = 1, \dots, N$ . According to the stability region  
 252 of the Runge-Kutta method, a time step of  $h = \mathcal{O}(\frac{1}{N})$  is required for the wave equation, which is  
 253 significantly larger than the time step required for the KdV equation.

254 This indicates that the Runge-Kutta method allows for a larger time step when applied to the wave  
 255 equation compared to the KdV equation, thereby improving the practicality and efficiency of our  
 256 method for a broad class of problems, including those involving wave phenomena.

## 257 References

- 258 [1] Mario Lezcano Casado. Trivializations for gradient-based optimization on manifolds. *Advances*  
 259 *in Neural Information Processing Systems*, 32, 2019.
- 260 [2] Felipe Linares and Gustavo Ponce. *Introduction to nonlinear dispersive equations*. Springer New  
 261 York, 2015. doi: 978-1-4939-2181-2.
- 262 [3] Fabio Luporini, Mathias Louboutin, Michael Lange, Navjot Kukreja, Philipp Witte, Jan Hück-  
 263 elheim, Charles Yount, Paul H. J. Kelly, Felix J. Herrmann, and Gerard J. Gorman. Ar-  
 264 chitecture and performance of devito, a system for automated stencil computation. *ACM*  
 265 *Trans. Math. Softw.*, 46(1), apr 2020. ISSN 0098-3500. doi: 10.1145/3374916. URL  
 266 <https://doi.org/10.1145/3374916>.
- 267 [4] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards  
 268 understanding the role of over-parametrization in generalization of neural networks. *arXiv*  
 269 *preprint arXiv:1805.12076*, 2018.
- 270 [5] Nischay Rai and Sabyasachi Mondal. Spectral methods to solve nonlinear problems: A review.  
 271 *Partial Differential Equations in Applied Mathematics*, 4:100043, 2021.
- 272 [6] David E. Stewart. *Numerical Analysis: A Graduate Course*. CMS/CAIMS Books in Mathematics,  
 273 4. Springer, 2022.
- 274 [7] Marcelo Viana and José M Espinar. *Differential equations: a dynamical systems approach to*  
 275 *theory and practice*, volume 212. American Mathematical Society, 2021.