DOI: 10.61782/fa.2023.0894

forum acusticum 2023

PHYSICAL MODELLING OF STIFF MEMBRANE VIBRATION USING NEURAL NETWORKS WITH SPECTRAL CONVOLUTION LAYERS

Carlos De La Vega Martin*
Queen Mary University of London
Centre for Digital Music

Mark Brian Sandler
Queen Mary University of London
Centre for Digital Music

ABSTRACT

Physical modelling synthesis is currently limited in its applications due to the high computational cost of some of the algorithms. Typically, these models are obtained by discretizing a mathematical model described by ordinary or partial differential equations, using well-established methods like finite differences or modal decomposition. Recent advances in machine-learning have sought to model these systems of differential equations using specialised architectures such as the Fourier Neural Operator (FNO), enabling extremely fast inference times and resolution-independent computational cost. Building on recent work extending the FNO approach for its application to acoustics problems, we examine the performance and robustness of these methods in the case of a stiff and lossy membrane. We show that the FNO approach is only able to accurately model the behaviour of the membrane within the range of timesteps used for training, becoming unstable or decaying rapidly beyond that.

Keywords: *fourier neural operator, physical modelling, stiff membrane, neural networks, spectral convolution*

1. INTRODUCTION

In the context of sound synthesis *physical modelling* is the use of physically motivated mathematical models to generate sound. The main advantage of this approach is that it allows to generate sounds that while physically plausible are not necessarily physically realizable. This is particularly useful in the context of musical instruments, where the goal is to generate sounds that are musically interesting. Many physical models are based on the solution of *partial differential equations* (PDEs) that describe the physical system of interest. The solution of the PDEs can be obtained using frequency-domain methods, such as the *functional transformation method* (FTM) [1], or time-domain numerical methods, such as *finite difference time domain* (FDTD) methods [2]. The main advantage of frequency-domain methods is that they are computationally efficient, but they are limited to linear systems. Time-domain methods are more computationally expensive, but they can be used to model nonlinear systems.

Recent years have seen an large amount of research in deep learning and neural networks [3] applied to many different fields, including audio [4]. Regarding scientific computing, recent advances in the field of *operator learning* [5–7] have shown that neural networks can be used to approximate the solution of PDEs. In particular recent work has proposed the use of recurrent neural networks (RNNs) with *spectral convolution layers* [7] to model acoustic systems. In this work we will focus on the evaluation of these recently proposed architectures to model the vibration of stiff membranes. The code used to generate the results presented in this paper is available at `https://github.com/cdelavegamartin/fa2023`

In Section 2 we will introduce the concept of operator learning, and review the network architectures proposed in [7]. In Section 3 we will introduce the mathematical model of stiff membrane vibration. In Section 4 we will explain the experimental setup and present the results, and in Section 5 we will draw our conclusions.

*__Corresponding author__: c.delavegamartin@qmul.ac.uk.

2. OPERATOR LEARNING FOR PDES

In the context of this work, we will define an *operator*, $\mathcal{G}$, as the mapping between two (possibly infinite-dimensional) function spaces, which are usually taken to be Banach spaces, or suitable subsets of them [6, 8]. In a more formal way,

$$\mathcal{G} : \mathcal{A}\left(D; \mathbb{R}^{d_a}\right) \to \mathcal{B}\left(D; \mathbb{R}^{d_b}\right), \quad a \mapsto b := \mathcal{G}(b), \quad (1)$$

where $\mathcal{A}$ and $\mathcal{B}$ are the input and output function spaces, and $D \subset \mathbb{R}^d$ is a domain in $\mathbb{R}^d$, with $d \in \mathbb{N}$ denoting its dimensionality. Therefore the operator $\mathcal{G}$ maps a function $a : D \to \mathbb{R}^{d_a}$ with $d_a \in \mathbb{N}$ components to a function $b : D \to \mathbb{R}^{d_b}$ with $d_b \in \mathbb{N}$ components.

Modeling time-dependent PDEs can be formulated as learning the operator $\mathcal{G}_{t_1 \mapsto t_2}$ that describes mapping between an N-dimensional state, $\mathbf{u}(\mathbf{x}, t) = (u_i)_{i=0}^N$ at time t_1 to the state at time t_2, where $\mathbf{x} \in \mathbb{R}^d$ is the spatial coordinate, and $t \in \mathbb{R}$ is the time coordinate. The operator $\mathcal{G}_{t_1 \mapsto t_2}$ can be then defined as,

$$\mathbf{u}(\mathbf{x}, t_2) = \mathcal{G}_{t_1 \mapsto t_2}(\mathbf{u}(\mathbf{x}, t_1)) \quad (2)$$

We note that the input and output function spaces are then assumed to be the identical $\mathcal{A} = \mathcal{B} = \mathcal{U}$, and the spatial domain is $D = \mathbb{R}^d$. To make this formulation compatible with a machine learning approach, we need to discretize the state $\mathbf{u}(\mathbf{x}, t)$ in time and space. Our choice for this discretization is a regular grid of discrete sampled points $\mathbf{x_n}$ and timesteps $t = kT$, where k is a temporal sampling index and T the sampling period. The regular sampling in time allows us to define a single discretized operator that can theoretically be used for extrapolation to any time t. The choice of a regular grid in space is not essential, but it will be explained later (See Sec. 2.1) why this is the natural choice given the architectures chosen to approximate the operator $\mathcal{G}_{k \mapsto k+1}$ in this work. This yields the tensor $\mathbf{U}\left[\mathbf{x_n}, k\right]$, and we can define the operator $\hat{\mathcal{G}} := \mathcal{G}_{k \mapsto k+1}$ as,

$$\mathbf{U}\left[\mathbf{x_n}, k+1\right] = \hat{\mathcal{G}}\left(\mathbf{U}\left[\mathbf{x_n}, k\right]\right) \quad (3)$$

Following the notation in [7], we will denote the state tensor at time k as $\mathbf{U}_k$ and $u^k_{ij\ldots}$ as its elements, where i is the index of the the state (physical) variable, and $j\ldots$ are the n spatial indices.

The approximation of operators such as $\hat{\mathcal{G}}$ using neural networks has been termed as the *neural operator* approach [5]. Many different approaches have been proposed for this task [5, 9, 10], but in this work we will focus on the *Fourier Neural Operator* (FNO) approach [5] and in particular the recurrent architectures derived from it and originally proposed in [7].

2.1 Spectral Convolution layer

The *spectral or fast convolution layer* [7] or *Fourier layer* [5] is the fundamental building block of the FNO approach. It is composed of a *Fourier transform* followed by a *affine transform*, and an *inverse Fourier transform*.

In mathematical terms the operation of the layer on the internal state tensor $\boldsymbol{H}$ with elements $h_{\nu j\ldots}$ can be written as follows,

$$\mathcal{S}\left(h_{\nu j\ldots}\right) = \tilde{\mathcal{F}}^{-1}\left[\sum_{\nu} A_{\nu\kappa j\ldots} \tilde{\mathcal{F}}\left(h_{\nu j\ldots}\right) + b_{\kappa j\ldots}\right] \quad (4)$$

where $\tilde{\mathcal{F}}$ denotes an operator encompassing padding, applying the FFT and truncating the spectrum to remove negative frequency components. Conversely $\mathcal{F}^{-1}$ encompasses reconstructing the negative frequency components, applying the inverse FFT and removing the padding. The indices ν and κ correspond the channels (internal state variables) of the input and output tensors. The coefficients of $A_{\nu\kappa j\ldots}$ and $b_{\nu j\ldots}$ are the trainable parameters of the layer and parametrized as complex numbers. This formulation has some changes respect to the original formulation in [5] and used in our reference model from [11]. First, in the original proposal only a fixed number of FFT bins, starting in the lower frequencies, were used, effectively zeroing the high frequency components of the spectrum. This can be seen as a kind of lowpass filtering, adn it was argued that it didn't significantly degrade performance while reducing the number of trainable parameters [5]. Second, the spatial dimensions of the input and output tensors are padded to ensure that the convolution is non-cyclic [7], which was not the case in the original proposal [5]. In the original implementation of the FNO [5], this was argued to be unnecesary since a trainable weighted skip connection in parallel to the spectral convolution layer was able to compensate in the case of a non-periodic input.

2.2 FNO-derived architectures

In this work we evaluate the performance of three different architectures for the approximation of operator $\hat{\mathcal{G}}$ in Eq. 3. The reference model, *markov neural operator* (MNO) [11], uses a modified version of the FNO architecture proposed in [5]. The main aspects are the adition of

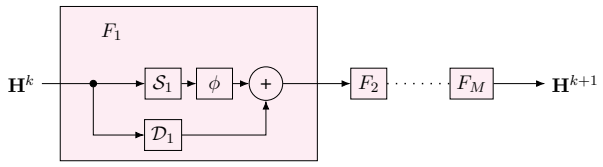

Figure 1. Block diagram showing the recurrent cell of the FRNN structure (dependencies of $\mathbf{H}$ on discrete time index k are denoted in the superscript for brevity). Taken from [7] and reproduced with permission of the authors.

a fixed padding to the input, and the conditioning with the coordinates of the spatial grid, concatenated to the input tensor. The mapping from the internal state tensor to the output tensor is done using a two linear layers with a nonlinear activation inbetween, whereas the input mapping is a single linear layer. The MNO uses the GeLU activation function [12]. The FNO architecture can be thought of as a *recurrent neural network* (RNN) with either a multiple [5], or single input time step [11, 13], depending on the implementation.

2.2.1 FRNN

The *Fourier recurrent neural network* (FRNN) [7] is a recurrent architecture based on the *spectral convolution layer* described in section 2.1. It maps, through the composition of a variable number of blocks, subsequent timesteps of the internal state of the network,

$$\mathbf{H}^{k+1} = F_M \circ F_{M-1} \circ \dots F_1 \left(\mathbf{H}^k\right), \tag{5}$$

where $\mathbf{H}$ is a tensor containing the internal states of the system at all spatial-sampling points, and the operation $F_m(\mathbf{H})$ is given by

$$F_m(\mathbf{H}) = \phi\left(\mathcal{S}_m(\mathbf{H})\right) + \mathcal{D}_m(\mathbf{H}) \tag{6}$$

where $\mathcal{S}$ is the described spectral convolution layer 4, ϕ is an elementwise activation function such as tanh or ReLU and $\mathcal{D}$ represents a weighted skip connection. The FRNN architecture is illustrated in figure 1. Note that the skip connection happens after the activation function, which is different from the MNO architecture, where the skip connection is before the activation function.

2.2.2 FGRU

The *Fourier gated recurrent unit* (FGRU) [7] is a recurrent architecture derived from the *gated recurrent unit*

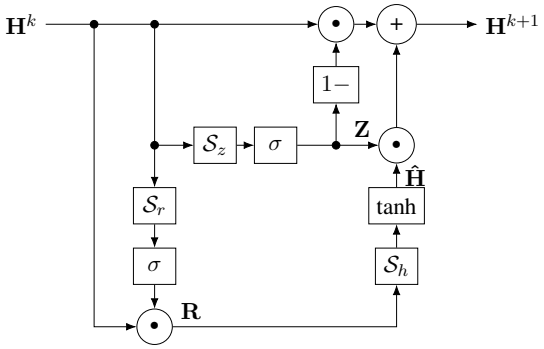

Figure 2. Block diagram showing the recurrent cell of the FGRU structure (dependencies of $\mathbf{H}$ on discrete time index k are denoted in the superscript for brevity). Taken from [7] and reproduced with permission of the authors.

(GRU) [14] incorporating the *spectral convolution layer* described in section 2.1. The operation of the FGRU can be written as follows,

$$\mathbf{Z} = \sigma\left(\mathcal{S}_z\left(\mathbf{H}^k\right)\right), \tag{7}$$

$$\mathbf{R} = \sigma\left(\mathcal{S}_r\left(\mathbf{H}^k\right)\right), \tag{8}$$

$$\hat{\mathbf{H}} = \tanh\left(\mathcal{S}_h\left(\mathbf{R} \odot \mathbf{H}^k\right)\right), \tag{9}$$

$$\mathbf{H}^{k+1} = (1 - \mathbf{Z}) \odot \mathbf{H}^k + \mathbf{Z} \odot \hat{\mathbf{H}}, \tag{10}$$

where $\mathcal{H}$ is again the internal state tensor, $\mathcal{S}_z$, $\mathcal{S}_r$ and $\mathcal{S}_h$ are spectral convolution layers and σ is the element-wise sigmoid activation function. The FGRU architecture is illustrated in figure 2.

2.2.3 Input and output mapping

The number of internal states for these recurrent architectures can be freely specified, and therefore we require an input $\mathcal{M}_{\text{in}}$ and output $\mathcal{M}_{\text{out}}$ mapping between the physical state tensor $\mathbf{U}$ and the internal state tensor $\mathbf{H}$. These are parametrized as linear maps, and can be written as

$$h^0_{\nu j\dots} = \mathcal{M}_{\text{in}}\left(u^0_{ij\dots}\right) = \sum_\kappa A_{\nu\kappa} u^0_{\kappa j\dots} + b_\nu, \tag{11}$$

$$u^k_{ij\dots} = \mathcal{M}_{\text{out}}\left(h^k_{\kappa j\dots}\right) = \sum_\kappa \tilde{A}_{i\kappa} h^k_{\kappa j\dots} + \tilde{b}_i, \tag{12}$$

Using these linear embeddings, is possible to train the network using *backpropagation through time* (BPTT) [15], as seen in Figure 3.

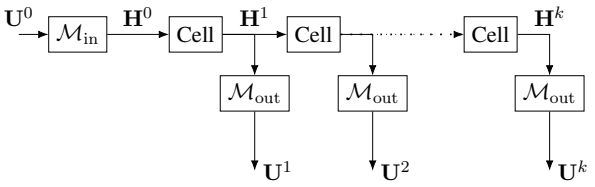

Figure 3. Block diagram showing the repeated application of a GRU or RNN cell along with the input and output mapping layers. $\mathbf{H}^k$ denotes the internal state of the NN, and $\mathbf{U}^k$ denotes the physical state of the system respectively, at step k. Taken from [7] and reproduced with permission of the authors.

3. THE LINEAR STIFF MEMBRANE

The linear stiff membrane is a 2D system that can be used to model a drum head or a flexible thin plate [2, 16, 17]. The system is described by the following PDE,

$$\rho H \ddot{u}(\mathbf{x},t) = T\nabla^2 u(\mathbf{x},t) - \frac{EH^3}{12(1-\nu^2)}\Delta^2 u(\mathbf{x},t) - 2d_0\dot{u}(\mathbf{x},t) + 2d_1\nabla^2\dot{u}(\mathbf{x},t) \quad (13)$$

where $u(\mathbf{x},t)$ is the displacement of the membrane at position $\mathbf{x}$ and time t, ρ is the mass density, H is the thickness of the membrane, T is the tension, E is the Young's modulus, ν is the Poisson ratio, d_0 is a frequency-independent damping coefficient and d_1 is a frequency-dependent coefficient. Note that *partial* time derivatives are denoted by a dot, ∇^2 is the Laplace operator and $\Delta^2 = \nabla^4$ is the bi-laplacian. We can combine the physical parameters and scale the spatial coordinates x, y by the length of the plate along the x-axis, L_x,

$$\ddot{u}(\mathbf{x},t) = \gamma^2\nabla^2 u(\mathbf{x},t) - \kappa^2\Delta^2 u(\mathbf{x},t) - 2\sigma_0\dot{u}(\mathbf{x},t) + 2\sigma_1\nabla^2\dot{u}(\mathbf{x},t) \quad (14)$$

where γ, κ, σ_0 and σ_1 have units of Hz, and $u(\mathbf{x},t)$ is in m. We consider the PDE to be defined over a rectangular domain $\Omega \subset \mathbb{R}^{[0,1]\times[0,ratio]}$ where $ratio = L_y/L_x$ and subject to clamped boundary conditions,

$$u(\mathbf{x},t) = 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega, \quad (15)$$

$$u'(\mathbf{x},t) = 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega. \quad (16)$$

where $\partial\Omega$ is the boundary of the domain Ω, and u' denotes the space derivative in the direction normal to the boundary of the domain.

3.1 Damping coefficients

In order to obtain realistic values for the damping coefficients σ_0 and σ_1, we use the following approximate formulae [2], which are valid for realistic values of the PDE parameters,

$$\xi(\omega) = \frac{-\gamma^2 + \sqrt{\gamma^4 + 4\kappa^2\omega^2}}{2\kappa^2} \quad (17)$$

$$\sigma_0 = \frac{6\ln(10)}{\xi(\omega_2) - \xi(\omega_1)}\left(\frac{\xi(\omega_2)}{T_{60}(\omega_1)} - \frac{\xi(\omega_1)}{T_{60}(\omega_2)}\right) \quad (18)$$

$$\sigma_1 = \frac{6\ln(10)}{\xi(\omega_2) - \xi(\omega_1)}\left(-\frac{1}{T_{60}(\omega_1)} + \frac{1}{T_{60}(\omega_2)}\right) \quad (19)$$

where ω is the angular frequency, ω_1 and ω_2 are the lower and upper angular frequencies at which the 60dB decay time, T_{60} is specified.

4. EXPERIMENTS

4.1 Data generation

To generate the datasets for training and evaluation, a numerical method was used to solve the PDE. The domain was discretized using a uniform grid with $N_x \times N_y$ grid-points, and the spatial partial derivatives were discretized using a centered finite difference scheme (FDM) with 2nd order accuracy. The resulting system of ODE was then integrated numerically using the SciPy python library [18], with a sampling frequency of 48 kHz (a timestep of 20.833 µs). Four combinations of the PDE parameters γ and κ were used to generate the datasets, (1) $\gamma = 1.0$, $\kappa = 0.1$, (2) $\gamma = 1.0$, $\kappa = 1.0$, (3) $\gamma = 100.0$, $\kappa = 0.1$, (4) $\gamma = 100.0$, $\kappa = 1.0$. The PDE parameters were chosen to cover a range of numerical behaviour, with only (3) corresponding to roughly a Mylar drumhead [16]. The damping coefficients were set to realistic values, corresponding to a 60dB decay time of 5 s at 100 Hz and 3 s at 2 kHz. The dimensions ratio L_y/L_x was chosen as 0.95. For each of the four combinations of PDE parameters, 1024 pairs of 1 input state and 39 output steps were generated, of which a 10% (103) were used for evaluation and the remaining 90% (921) were used for training. This corresponds to approximately 0.8 ms.

4.2 Training details

The input to the model is a tensor of size $B \times N_x \times N_y \times 2$, where B is the batch size, N_x and N_y the number of grid-

points in the x and y directions and we have 2 state variables, the initial displacement $u_0 = u(t = 0, x, y)$, and velocity $v_0 = \dot{u}(t, x, y)|_{t=0}$. The output is a tensor of size $B \times N_T \times N_x \times N_y \times 2$, where N_T is the number of output steps the first channel contains the displacement and the second channel contains the velocity. We employed the MSE between the target and predicted output sequences as the loss function, using BPTT to back-propagate the gradients through time. To train the models, we used the same hyperparameters as in [7]. We used the AdamW optimizer [19] with default parameters, and the 1-cycle learning-rate scheduling scheme [20], modulating from a learning rate of 10^{-4} to 10^{-3}. Training was conducted for 5000 epochs with batch size set to 400, as that is the maximum that would fit in the GPU memory. The training was done on a single GPU (Nvidia A100).

4.3 Model validation

Three (3) different random seeds were used both when instantiating and training the models, and when generating the datasets. This resulted in 9 possible combinations of trained models and evaluation datasets, for each of the 4 different PDE parameter combinations. In 1 we can see the aggregated mean and standard deviation for each combination of PDE paramenters and NN architecture, FGRU, FRNN and the reference MNO architecture, REF. Note that in the case of the REF architecture with $\gamma = 1.0$ and $\kappa = 1.0$, one initial condition was removed due to a diverging result to obtain the statistics. This was run 85 of the validation data generated with seed=1, and evaluated on the model initialised with seed=2. Preliminary investigations didn't reveal any obvious reason for this behaviour, although it is likely to be related to the instability showed by the model when evaluating the extrapolation performance. A a more in-depth study of this anomaly is left for future work.

As we can see in Table 1 the performance of the three architectures is comparable, with the FGRU performing better for $\gamma = 1.0$, $\kappa = 0.1$ and FRNN and REF models performing better for $\gamma = 100.0$, $\kappa = 0.1$ and $\gamma = 100.0$, $\kappa = 1.0$. In the case of $\gamma = 1.0$, $\kappa = 1.0$ the models perform simlarly, within 1 standard deviation of each other.

4.3.1 Extrapolation

A dataset of 30 initial conditions and 2000 timesteps (41.7 ms) were generated for each combination of PDE parameters, and used to evaluate the extrapolation performance of the models trained. The initial conditions are generated using the same method as for the validation dataset, but with a different random seed. The displacement MSE for the three models trained on three of the PDE parameter combinations can be seen in Figures 4 and 5 and 6. The performance of the REF model for $\gamma = 100.0$, $\kappa = 1.0$ is very similar to the $\gamma = 100.0$, $\kappa = 0.1$ case, and the velocity MSE behaves qualitatively in the same manner as the displacement in all cases, with the model outputs always diverging in the cases the displacement diverges.

Table 1. Results MSE *mean (std)*. * Excluding a single diverging evaluation.

			Displacement	Velocity
γ	κ	Model		
1.0	0.1	FGRU	0.0007 (0.0001)	0.0046 (0.0011)
		FRNN	0.0028 (0.0024)	0.0064 (0.0039)
		REF	0.0019 (0.0009)	0.0083 (0.0020)
	1.0	FGRU	0.0039 (0.0007)	0.0092 (0.0015)
		FRNN	0.0020 (0.0014)	0.0062 (0.0047)
		REF*	0.0023 (0.0005)	0.0074 (0.0007)
100.0	0.1	FGRU	0.0050 (0.0002)	0.0081 (0.0008)
		FRNN	0.0011 (0.0007)	0.0023 (0.0015)
		REF	0.0017 (0.0002)	0.0028 (0.0003)
	1.0	FGRU	0.0072 (0.0002)	0.0131 (0.0005)
		FRNN	0.0012 (0.0003)	0.0032 (0.0010)
		REF	0.0016 (0.0001)	0.0037 (0.0004)

The extrapolation performance is extremely poor for all models, with the accuracy falling off rapidly beyond the timesteps covered by the training data in all cases. The degradation in performance happens very soon after the end of the time span covered by the training data, as it can be seen in Figure 7, and this is the case for all combinations of PDE parameters. The FGRU output decays rapidly to zero in all cases, while the FRNN and REF output diverges for many initial conditions, two examples of which can be seen in Figures 8 and 9. We can see that the REF model diverges for almost all initial conditions when $\gamma = 100.0$, and for over two thirds of the runs for $\gamma = 1.0$, while the FGRU model doesn't diverge in any case, as expected from the GRU architecture. The FRNN model divergence behaviour is the most unpredictable one, with the divergence rate varying from 0% to 100% depending on the PDE parameters and initial conditions. There is also no correlation between the values of γ and κ and the divergence rate, as the FRNN model diverges for both low

Table 2. Rate of diverging runs after 2000 timesteps.

			Rate of diverging runs		
			mean	min	max
γ	κ	Model			
1.0	0.1	FGRU	0.0000	0.0000	0.0000
		FRNN	0.8333	0.5000	1.0000
		REF	0.7556	0.6000	0.9333
	1.0	FGRU	0.0000	0.0000	0.0000
		FRNN	0.3333	0.0000	1.0000
		REF	0.6778	0.0333	1.0000
100.0	0.1	FGRU	0.0000	0.0000	0.0000
		FRNN	0.3222	0.0000	0.6333
		REF	0.9889	0.9667	1.0000
	1.0	FGRU	0.0000	0.0000	0.0000
		FRNN	0.8333	0.6000	1.0000
		REF	1.0000	1.0000	1.0000

and high values of γ and κ. A summary of the extrapolation performance regarding instability for all models and PDE parameter combinations can be seen in Table 2.

These stands in contrast with the previous results for the 2D wave equation presented in [7], where the REF model was able to extrapolate up to twice the training length. Although the system at hand is slightly more complicated, it is still linear, with the frequencies and wave speeds being at most 1% (case $\gamma = 100.0$, $\kappa = 0.1$) or 10% (cases $\gamma = 100.0$, $\kappa = 1.0$ and $\gamma = 1.0$, $\kappa = 0.1$) higher than the ones in the 2D wave equation [21]. An initial guess for the significant difference in performance could be that the previous results [7] were obtained training the model using data generated with frequency domain methods, specifically the *functional transformation method* [1] and the *Fourier-Sine transform* [22]. Due to the bandlimiting imposed by selecting a finite number of modes, the training data might have been more suitable for representation with architectures using *spectral convolution layers*. The exploration of this hypothesis is left for future work.

5. CONCLUSION

In this work evaluated the performance of some recently proposed neural network architectures [7] for solving the stiff membrane PDE. While the validation loss is comparable to the previously published results for other acoustic systems [7], none of of the architectures are able to extrapolate in our case, and in many instances they present unstable behaviour. More investigation is needed to un-

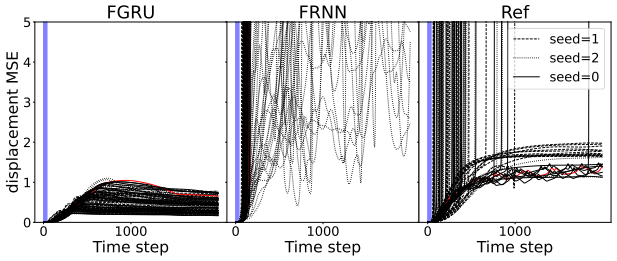

Figure 4. Extrapolation results for $\gamma = 1.0$ and $\kappa = 0.1$. Shaded in blue is the timestep span covered by the training data.

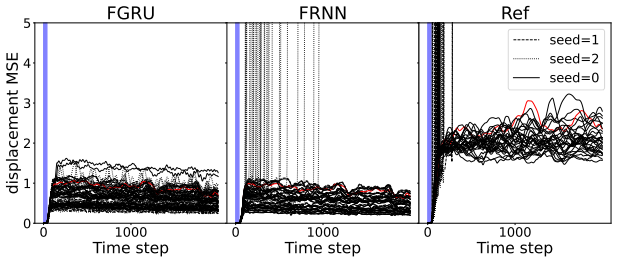

Figure 5. Extrapolation results for $\gamma = 1.0$ and $\kappa = 1.0$. Shaded in blue is the timestep span covered by the training data.

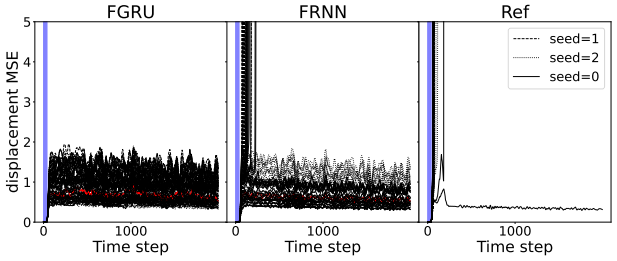

Figure 6. Extrapolation results for $\gamma = 100.0$ and $\kappa = 0.1$. Shaded in blue is the timestep span covered by the training data.

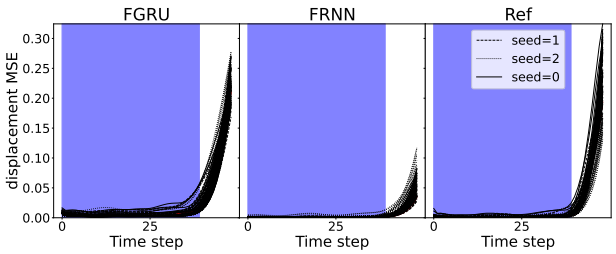

Figure 7. Close up of the extrapolation results for $\gamma = 100.0$ and $\kappa = 0.1$. Shaded in blue is the timestep span covered by the training data.

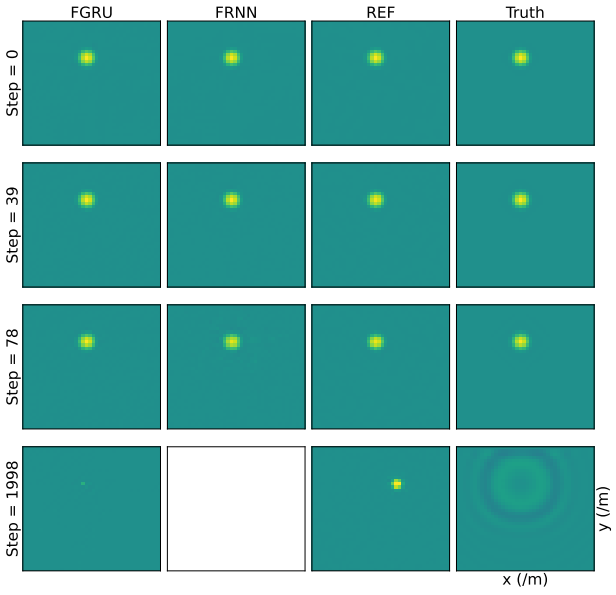

Figure 8. Example output for $\gamma = 1.0$ and $\kappa = 0.1$. The initial condition is highlighted in red in Fig. 4. Blank space indicates that the model diverged.

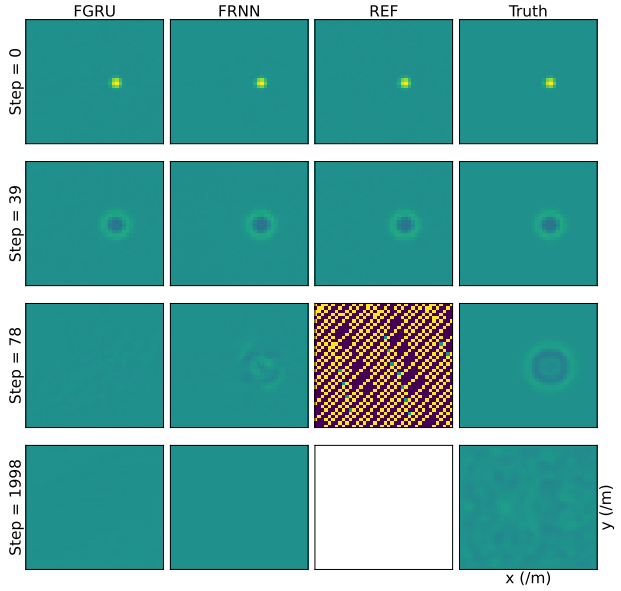

Figure 9. Example output for $\gamma = 100.0$ and $\kappa = 0.1$. The initial condition is highlighted in red in Fig. 6. Blank space indicates that the model diverged.

derstand the difference in performance.

6. ACKNOWLEDGMENTS

This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. `http://doi.org/10.5281/zenodo.438045`

7. REFERENCES

[1] L. Trautmann and R. Rabenstein, *Digital Sound Synthesis by Physical Modeling Using the Functional Transformation Method*. Kluwer Academic/Plenum Publishers, 2003.

[2] S. Bilbao, *Numerical Sound Synthesis*. Chichester, UK: John Wiley & Sons, Ltd, Oct. 2009.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[4] H. Purwins, B. Li, T. Virtanen, J. Schluter, S.-Y. Chang, and T. Sainath, "Deep learning for audio signal processing," *IEEE J. Sel. Top. Signal Process.*, vol. 13, pp. 206–219, May 2019.

[5] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier Neural Operator for Parametric Partial Differential Equations," in *International Conference on Learning Representations*, May 2021.

[6] S. Lanthaler, S. Mishra, and G. Karniadakis, "Error estimates for DeepOnets: A deep learning framework in infinite dimensions," *Transactions of Mathematics and Its Applications*, 2022.

[7] J. D. Parker, S. J. Schlecht, R. Rabenstein, and M. Schäfer, "Physical Modeling using Recurrent Neural Networks with Fast Convolutional Layers," in *Proceedings of the 25th International Conference on Digital Audio Effects (DAFx20in22)*, (Vienna, Austria), June 2022.

[8] N. Kovachki, S. Lanthaler, and S. Mishra, "On Universal Approximation and Error Bounds for Fourier Neural Operators," *Journal of Machine Learning Research*, vol. 22, no. 290, pp. 1–76, 2021.

[9] Z.-Y. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Graph kernel network for partial differential equations," *ICLR 2020*, 2020.

[10] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nat Mach Intell*, vol. 3, pp. 218–229, Mar. 2021.

[11] Z. Li, M. Liu-Schiaffini, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Learning Chaotic Dynamics in Dissipative Systems," in *Advances in Neural Information Processing Systems*, Oct. 2022.

[12] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)." `http://arxiv.org/abs/1606.08415`, June 2016.

[13] A. Tran, A. Mathews, L. Xie, and C. S. Ong, "Factorized Fourier Neural Operators," in *International Conference on Learning Representations*, Feb. 2023.

[14] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, (Doha, Qatar), pp. 103–111, Association for Computational Linguistics, Oct. 2014.

[15] P. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Oct. 1990.

[16] N. H. Fletcher and T. D. Rossing, *The Physics of Musical Instruments*. New York, NY: Springer New York, 1991.

[17] A. Torin, *Percussion Instrument Modelling in 3D: Sound Synthesis through Time Domain Numerical Simulation*. PhD thesis, University of Edinburgh, 2016.

[18] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nat Methods*, vol. 17, pp. 261–272, Mar. 2020.

[19] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization." `http://arxiv.org/abs/1711.05101`, Jan. 2019.

[20] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006, pp. 369–386, SPIE, May 2019.

[21] P. M. Morse and K. U. Ingard, *Theoretical Acoustics*. International Series In Pure And Applied Physics, McGraw-Hill, 1st edition ed., 1968.

[22] S. Bilbao, "Modal-Type Synthesis Techniques for Nonlinear Strings with an Energy Conservation Property," in *Proceedings of the 7th International Conference on Digital Audio Effects (DAFx04)*, 2004.