Fast Acoustic Wave Simulation with Neural Operators

Ngoc Vo
Stanford University
hnngocvo@stanford.edu

Fangjun Zhou
Stanford University
fzhou48@stanford.edu

Kevin Liu
Stanford University
liuxk@stanford.edu

Abstract

Simulating realistic acoustic wave propagation is essential for physically accurate audio rendering in computer graphics. Traditional finite-difference time-domain (FDTD) solvers provide accurate results but suffer from high computational costs due to stability constraints on time and space resolution. In this work, we investigate the use of neural operator architectures for accelerating 2D acoustic wave simulations. We implement and evaluate several neural operator models, including the Fourier Neural Operator (FNO), its low-rank tensorized variant (TFNO), the U-Net-augmented U-FNO, and the bandlimited Convolutional Neural Operator (CNO). Our experiments are conducted on a novel dataset of animated acoustic scenes generated using the WaveBlender FDTD solver, capturing time-varying boundary conditions and complex geometries. Furtheremore, we demonstrate an iterative inference strategy to extend neural operators for long-horizon audio generation. Quantitative results show that the CNO achieves the best accuracy while being over 150 times faster than the FDTD baseline. Our findings suggest that neural operators offer a promising direction for real-time, high-fidelity acoustic simulation in graphics applications.

1. Introduction

The propagation of sound waves through an acoustic medium is governed by partial differential equations (PDEs), most commonly the linear wave equation. Simulating such wave dynamics is essential in domains like virtual acoustics and computer graphics, where realistic sound rendering requires accurate modeling of reflection, diffraction, and interference patterns. Traditionally, finite-difference time-domain (FDTD) solvers have been the method of choice for these simulations. While FDTD methods are conceptually straightforward and parallelizable, they suffer from a significant limitation: to maintain numerical stability, the Courant–Friedrichs–Lewy (CFL) condition requires small time steps as spatial resolution increases. This leads to substantial computational costs, making high-resolution or real-time simulations infeasible in many settings.

Recent advances in scientific machine learning offer a compelling alternative. Neural operators, especially the Fourier Neural Operator (FNO), provide a framework for learning mappings between function spaces, enabling fast, resolution-invariant approximations to PDE solutions. Once trained, these models produce predictions in a single forward pass, bypassing iterative updates and allowing for larger time steps without sacrificing accuracy. This study explores the viability of applying neural operators to accelerate acoustic wave simulations in computer graphics applications, focusing on scenarios with reflections, boundary interactions, and animated sound sources. The objective is to assess whether neural operators can retain physical fidelity while reducing computational burden, thereby facilitating scalable acoustic simulation for graphics applications.

2. Related Works

2.1. Neural Operators and Operator Learning

Neural operators have emerged as a transformative approach for simulating complex physical systems. Unlike traditional neural networks, which learn point-wise mappings between fixed-resolution inputs and outputs, neural operators approximate mappings between entire function spaces. This capability allows them to generalize across discretizations and perform inference at unseen resolutions. As emphasized by Azizzadenesheli et al. [1], neural operators overcome key limitations of conventional numerical solvers, including their dependence on fine grids, lack of differentiability, and inability to exploit observational data. By learning efficient representations and operating in the frequency domain, models like the FNO achieve orders-of-magnitude speedups across a wide range of applications—from weather forecasting to inverse design—while remaining physically consistent and scalable.

2.2. Neural Operators in Acoustic Simulation

In the context of acoustics, Middleton et al. [9] demonstrated that FNOs can effectively simulate 2D linear acoustic wave propagation, including complex interactions such as superposition, reflections, and diffraction. Compared to FDTD baselines, FNOs were up to 25 times faster, while maintaining high fidelity in both time and frequency domains. Moreover, the network compressed 24.4 GB of FDTD training data into just 15.5 MB of model weights, showcasing its efficiency as both a predictor and a data compressor.

Li et al. [6] extended these ideas to seismic modeling, showing that FNOs can handle variable velocity fields and complex geometries by learning the frequency-domain solution to the Helmholtz equation. Their parallelized FNO framework (PFNO) enabled scalable learning across multiple source positions and frequencies, retaining high accuracy even under distributional shifts and noisy training labels. These results indicate that operator-learning frameworks can effectively decouple resolution from runtime, offering substantial improvements over finite-difference and finite-element methods.

2.3. Acoustic Simulation in Computer Animation

In previous studies about using neural operators in acoustic simulation, both [9] and [6] simulate their dataset in static environments. Specifically, Middleton et al. [9] generated 4 types of static environment with different boundary conditions. Li et al. [6] use OpenFWI dataset which only contains static environment as well. In computer animation, it's often the case that the sound sources in the scene are animated. Therefore, the datasets in previous work will not generalize in the computer animation applications.

As pointed out by Wang et al. [12] and Xue et al. [14], simulating animated sound source directly with traditional FDTD solver will cause artifacts in the results. Specifically, when the surface of the sound source moves from one cell to another, the discrete cell jump can result in undesired popping sound in the audio. [12] solved this problem by introducing a layer of ghost cells around the boundaries and [14] solved the problem by using a blending scheme to avoid discrete cell movement. In this study, we adopted the WaveBlender algorithm from [14] to generate our training and testing dataset.

3. Problem Statement

3.1. Governing Equation

We consider the 2D acoustic wave equation given by:

$$\frac{\partial^2 p(x,y,t)}{\partial t^2} = c^2 \nabla^2 p(x,y,t),$$

where $p(x,y,t)$ denotes the acoustic pressure field parameterized by spatial coordinate (x,y) and time t, c is the wave speed, and ∇^2 is the Laplacian. A Neumann boundary condition is imposed:

$$\frac{\partial p}{\partial n} = g(t),$$

where $g(t)$ is a time-dependent function defining the pressure gradient at the boundaries. In practice, this is related to the normal velocity of the vibrating sound source.

We aim to learn a discrete-time operator $\mathcal{G}$ that maps the initial state and boundary condition to the pressure field at a future time T:

$$p_{1\cdots T} = \mathcal{G}(p_0, v_{0\cdots T-1}),$$

where $p_{1\ldots T}$ are the time sequenced pressure field from time step 1 to T and $v_{0\ldots T-1}$ are the boundary conditions (surface normal velocity) from time step 0 to $T-1$.

3.2. Baseline Method

Fourier Neural Operator (FNO)

As previous studies used FNO to learn acoustic wave function, we choose it as our baseline model.

The FNO approximates a global integral operator by parameterizing it in the frequency domain. At each layer, it updates the state via:

$$u_{k+1}(x) = \sigma(\mathbf{W}u_k(x) + \mathcal{F}^{-1}(\mathbf{R}\cdot\mathcal{F}(u_k))) \qquad (1)$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier and inverse Fourier transforms, $\mathbf{R}$ is a learned complex-valued kernel in Fourier space, and $\mathbf{W}$ is a learned point-wise linear transformation. This architecture enables efficient modeling of non-local spatial interactions and supports inputs of arbitrary resolution.

FDTD Numerical Solver

One of the main advantages of neural operators over traditional numerical methods is its high computational efficiency. To evaluate this, we also compare our model with the WaveBlender FDTD solver as a benchmark component for real-world application. However, as we are using WaveBlender to generate ground-truth training and testing data, we did not use it as a baseline for accuracy.

4. Dataset

To generate training and testing data for our models, we simulate acoustic wave propagation using our implementation of the WaveBlender solver in Taichi. This solver numerically integrates the 2D acoustic wave equation in a discrete grid. The input of the solver is a function of boundary conditions. The solver then solves the pressure field and velocity field over time (Figure 1). Unlike the FDTD solver implemented in [9], our solver does not enforce periodic or perfect reflecting domain boundary conditions. Instead, we used the perfectly matched layer (PML) [8] to simulate the absorbing domain boundary.

We generated 9 different environments with random audio source and obstacles moving in the scene. We also generated the audio played by the sound source randomly in different frequencies varying from 20Hz to 20kHz. The simulation domain size of all the scenes are $1m \times 1m$.

We've used 8 out of 9 scenes for training and validation and test all models on the 9th scene. We also split 20% of the training dataset for validation and save the best model on validation set to avoid overfitting. As each scene are randomly generated, the testing dataset is completely unseen to the models.

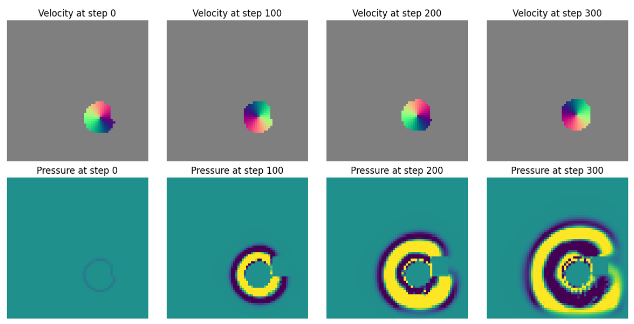

Figure 1. WaveBlender Solver

5. Methods

To approximate the solution operator of the 2D acoustic wave equation, we implement and train three types of neural operator architectures: U-FNO [5], CNO [5], and TFNO. Each model is trained to map from the initial acoustic state and time-dependent boundary condition to the pressure field at a future time, effectively learning the evolution of the wave equation.

U-FNO

U-FNO (U-Net Fourier Neural Operator) is a hybrid architecture that enhances the expressive power of Fourier Neural Operators by appending a U-Net encoder-decoder pathway within each Fourier layer. This design is especially effective for capturing both low-frequency global patterns and high-frequency local features such as sharp wavefronts or discontinuities. We adopt the U-FNO formulation introduced by Wen et al. [13], originally designed for modeling multiphase CO_2 flow in porous media.

The U-FNO architecture (see Figure 2) consists of an initial lifting layer P that projects input features to a high-dimensional latent space, followed by a series of Fourier and U-Fourier layers. Each U-Fourier layer augments the standard Fourier layer with an additional U-Net block, enhancing its ability to capture multi-scale information. The layer update is given by:

$$v_{k+1}(x) = \sigma\left(\mathcal{F}^{-1}\left(R \cdot \mathcal{F}(v_k)\right)(x) + \mathcal{U}(v_k)(x) + W(v_k(x))\right),$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and its inverse, R is a learnable spectral filter, $\mathcal{U}$ is a U-Net applied in the spatial domain, W is a pointwise linear operator, and σ denotes a nonlinear activation function such as ReLU.

In our implementation, we define a 2D U-FNO model using PyTorch by adapting the original architecture to operate on purely spatial inputs. The model begins with a lifting linear layer that projects input fields into a higher-dimensional latent space. This is followed by a configurable sequence of standard Fourier layers and U-Fourier layers, where the number of each (denoted L and M, respectively) is treated as a tunable hyperparameter. Each Fourier or U-Fourier layer performs spectral convolution in the frequency domain using a truncated set of Fourier modes, followed by pointwise bias and nonlinear activation. The U-Fourier layers additionally incorporate a 2D U-Net encoder-decoder path in the spatial domain, enabling localized refinement and better representation of high-frequency features. Finally, a projection linear layer maps the output back to the original target space.

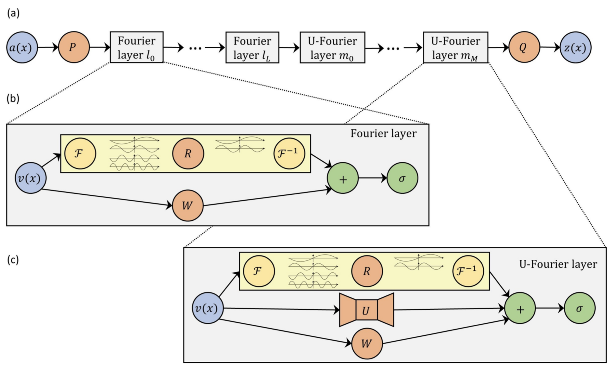

Figure 2. U-FNO architecture adapted from [13]. The U-Fourier layer (c) integrates Fourier spectral convolution with a U-Net block to improve resolution of localized features.

CNO

The CNO replaces global spectral convolution with learned local convolutions, modeling the operator through spatial integral kernels:

$$u_{k+1}(x) = \int_{\Omega} \kappa(x-y)u_k(y)\,dy + b(x),$$

where κ is a learnable kernel and $b(x)$ is a bias term. This formulation is implemented using convolutional layers, offering computational simplicity and improved local feature extraction. Although CNOs may not capture long-range dependencies as effectively as FNOs, they are often easier to

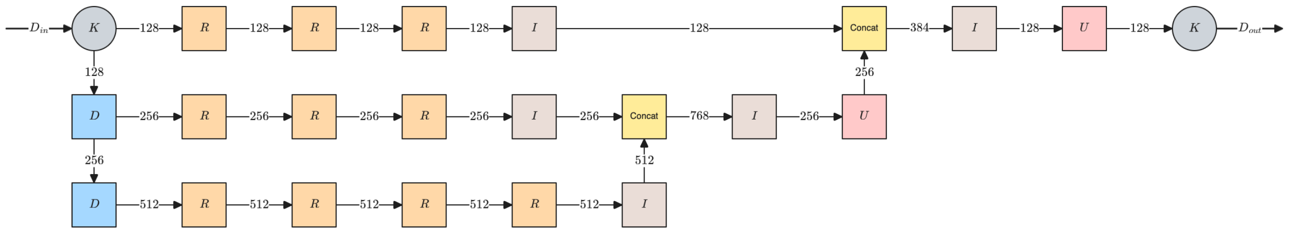

Figure 3. Our Acoustic CNO Architecture

train and interpret. These properties made us believe that CNO fits our application well as the acoustic wave problem is local and does not depend on distance input.

As pointed out in the original FNO paper [7], the use of convolution kernels makes the model no longer scale invariant. In other words, the kernel trained in one resolution cannot be used for inference in another resolution. Raonic et al.[10] proposed a solution to this problem by introducing the band limit to the input function. Specifically, CNO assumes that the highest frequency component ω in the input function does not exceed a certain threshold. This allows the model to train on the discrete data set that satisfy this condition and inference on any input with sample rate higher than 2ω.

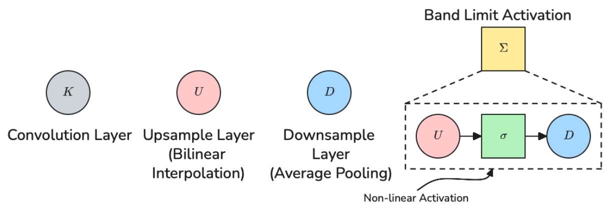

Figure 4. Basic Layers in CNO

With this band limit constraint, CNO differs from traditional CNN in a few ways. First, it's important to avoid aliasing when applying upsample and downsample layers. In the original CNO paper, the authors apply a Sinc low pass filter before downsample and after upsample layers [10]. In our implementation, we use average pooling layers for downsample layers and bilinear interpolation for upsample layers to achieve similar result. Another difference is non-linear layers cannot be applied directly as it'll introduce aliasing as well. To mitigate this issue, all the non-linear layers are applied in higher frequency and wrapped with a upsample layer and a downsample layer. These layers are shown in Figure 4.

Using these band limit layers, we built four types of block for CNO following the implementation in [10] (Figure 5). The invariant block is the band limit equivalent of the convolution layer with activation in regular CNN. Upsample and downsample blocks appends an upsample and

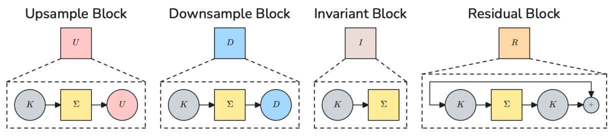

Figure 5. Building Blocks in CNO

downsample layer after an invariant block. Residual block is the band limit equivalent of the residual block in ResNet [3].

We use the aforementioned band limit blocks to build our CNO model. Following [10], we designed our model as a U-Net like fully convolutional neural network. The detail architecture is shown in Figure 3.

TFNO

The tensorized Fourier neural operator (TFNO) is a variant of the FNO that is designed to be more lightweight and efficient. Specifically, consider the Fourier convolution kernel

$$\mathbf{R} \in \mathbb{C}^{\overbrace{\alpha \times \cdots \times \alpha}^{d \text{ times}} \times I \times O \times 2^{d-1} L}$$

from (1), where d is the spatial dimension (in this case, $d = 2$), α is the number of modes per dimension, I is the number of input channels, O is the number of output channels, and L is the number of layers.

We can obtain a low-rank approximation of $\mathbf{R}$ via the Tucker decomposition [11]:

$$\mathbf{R} \approx \mathbf{G} \times_1 \mathbf{U^{(1)}} \cdots \times_d \mathbf{U^{(d)}} \times_{d+1} \mathbf{U^{(I)}} \times_{d+2} \mathbf{U^{(O)}} \times_{d+3} \mathbf{U^{(L)}}$$

where

$$\begin{aligned} &\mathbf{G} \in \mathbb{C}^{R_L \times R_I \times R_O \times R_1 \times \cdots \times R_d}, \\ &\mathbf{U^{(i)}} \in \mathbb{C}^{R_i \times \alpha}, \qquad (i = 1, \ldots, d) \\ &\mathbf{U^{(I)}} \in \mathbb{C}^{R_I \times I}, \quad \mathbf{U^{(O)}} \in \mathbb{C}^{R_O \times O}, \quad \mathbf{U^{(L)}} \in \mathbb{C}^{R_L \times L} \end{aligned}$$

are learned low-rank factor matrices to be learned. The multilinear ranks $(R_1, \ldots, R_d, R_I, R_O, R_L)$ are hyperparameters that control the approximation accuracy and model capacity [4].

This factorization significantly reduces the number of learnable parameters, leading to lower memory usage and faster training (compared to the standard FNO architecture), without compromising expressiveness. In some cases, TFNO may also improve generalization by implicitly regularizing the convolution weights [4].

Here, we implement a TFNO model for our acoustic wave simulation task, using the Python `neuraloperators` library.

6. Experiment and Results

Implementation and Training Details

As aforementioned, we model the acoustic propagation problem by predicting T pressure field steps from the initial pressure field and T boundary conditions. Using a small T means the model needs to learn the mapping over a smaller time step. This is often not a hard problem as the pressure field change is also small over this time step. Using a small T also means we need to call the kernel more frequently to get the audio clip in the same length.

By contrast, using a large T would increase the model size significantly and increase the training time. Additionally, projecting larger input into the same latent space makes it harder for the model to encode all the motion information necessary for prediction. In our study, we choose $T = 63$ for all the models considering the balance between training speed and model accuracy.

For the FNO and TFNO models, using the Neural Operator library, we trained each model using the Adam optimizer with a learning rate of 0.008. The TFNO model was configured with a decomposition rank of 16 in both spatial dimensions, enabling a lower-memory approximation of the spectral weights.

For the CNO model, we used the Adam optimizer with a learning rate of 0.001 and trained for 32 epochs. The CNO architecture consists of 15 residual blocks, each built from 3×3 convolutions with ReLU activations and skip connections to ensure stable training and preserve local structure. We use average pooling and bilinear interpolation within intermediate layers for bandlimiting, and a final linear projection layer maps the latent features back to the output space. ReLU nonlinearities are applied throughout the network.

The U-FNO architecture included 3 Fourier layers followed by 3 U-Fourier layers, each using 24 retained Fourier modes per spatial dimension. We used a 2-layer U-Net block in each U-Fourier layer, along with ReLU activations throughout. We used the Adam optimizer with learning rate 0.001, and trained for 50 epochs.

Model selection was also based on validation loss to ensure generalization.

Evaluation

To test the accuracy of all the models, we evaluate them with MSE, L2, and H1 loss. In the case of acoustic simulation, MSE measures the average per-pixel error which reflects the overall fidelity of the simulation. L2 loss measures the sum of pressure field different over the entire domain, which reflects the model's ability to preserve total energy. H1 loss takes the gradient of the predicted pressure field into account which reflects the stability of the model.

Based on Table 1, the CNO model outperforms all other neural operator variants in the acoustic simulation task. This is theoretically justified by CNO's use of localized convolutional kernels and bandlimited filtering, which are well-suited for modeling wave propagation where high-frequency details and local interactions dominate. In contrast, the U-FNO, while less accurate than CNO, still significantly outperforms the FNO baseline by combining global Fourier modeling with U-Net-based spatial refinement, allowing it to better resolve multi-scale features. Meanwhile, the TFNO shows only a modest improvement over FNO in accuracy, which aligns with its design: TFNO leverages tensor decomposition to compress the spectral weights, improving computational efficiency but at the cost of reduced expressiveness.

While all neural operator models outperform the FNO baseline in accuracy (Table 1), their runtime efficiency varies. Interestingly, the CNO achieves the fastest inference speed, surpassing even the simpler FNO baseline, likely due to its use of local convolutional operations. In contrast, the U-FNO, despite its improved accuracy, incurs the highest computational cost, which can be attributed to its more complex architecture that combines Fourier spectral convolutions with U-Net encoder-decoder blocks. These results suggest a trade-off between model complexity and inference efficiency, with CNO offering the best performance in terms of accuracy and speed.

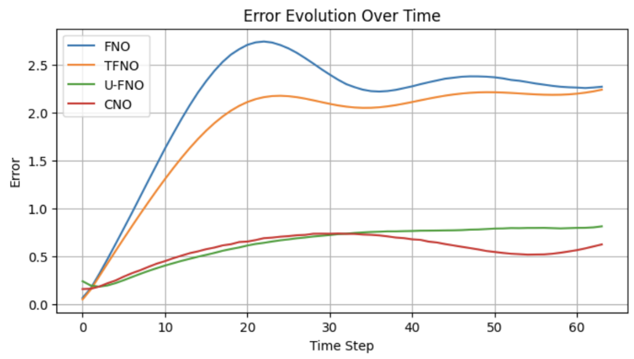

Figure 6. Error Evolutions of Neural-Operator Variants

Another important evaluation in our study is prediction error evolution over time steps. As our models predict 63 steps at once, the accuracy may be different at different

Model	L^2	H^1	MSE	Speed (steps/second)	Number of Parameters
FNO (baseline)	49.23	51.07	13.93	5349	19,272,640
U-FNO	24.77	25.14	2.98	1900	16,631,040
CNO	**20.45**	**21.90**	**1.98**	**5359**	8,885,440
TFNO	45.08	47.06	12.21	3092	**958,828**

Table 1. Comparison between Neural-Operator Variants

steps. We plot a mean absolute error curve to evaluate this property. Figure 6 shows this plot for each of the four neural operator models we tested.

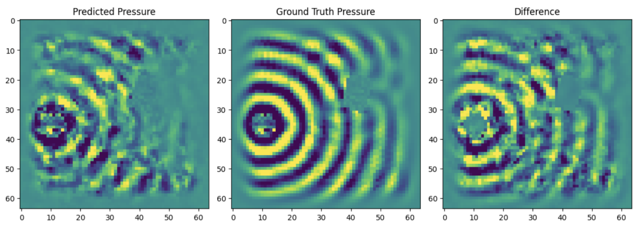

Figure 7. FNO Inference Result (final time step)

We observe that the FNO model performs better at the beginning. The error steadily increases before dipping in the middle and plateauing. The TFNO model produces a similar error evolution but slightly better, especially at the middle time steps.

Plotting the inference result at the last time step (Figure 7), we see that the FNO model captures part of the acoustic wave but generally yields a poor recovery. The TFNO inference result (Figure A.1) is similar.

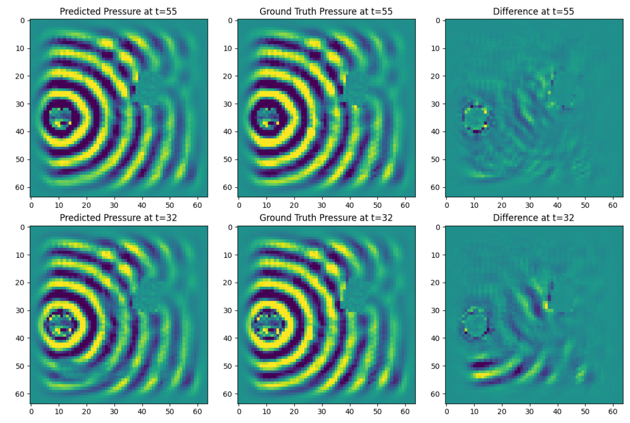

Figure 8. CNO Inference Result

On the other hand, Figure 6 suggests that the CNO model tends to perform better at the beginning and near the end of the sequence while producing worse results in the middle. We also show this difference quantitatively in Figure 8. As demonstrated in the plot, the simulation degrades at step 32 and recovers at step 55.

Meanwhile, the U-FNO model exhibits gradual error accumulation, with performance degradation most pronounced in the later steps. This trend reflects the increasing difficulty of maintaining physical consistency in long-range temporal extrapolation.

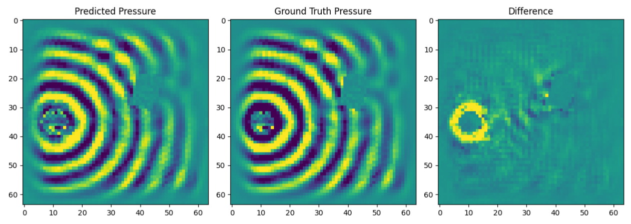

Figure 9. U-FNO inference result at the final time step. Left: predicted pressure; center: ground truth; right: absolute error.

At the same time, Figure 9 shows that U-FNO is able to reproduce the overall structure of the wavefronts, including major reflections and interference patterns. However, differences remain, especially near high-frequency edges, where artifacts and attenuation can be observed.

As a result, our observation of the error evolution and inference plots coheres with our theoretical intuition that CNO is particularly effective (exceeding that of our FNO variants) at capturing localized and high-frequency features at the start and end of a wave sequence, but may struggle with long-range dependencies due to its lack of explicit temporal modeling.

Finally, we also consider the memory usage of each model (i.e., number of parameters) in Table 1. The FNO baseline contains the most parameters (over 19 million), followed by the U-FNO and then the CNO, which is about half the size as the baseline. The smallest model is the TFNO (less than 1 million), which is about 20 times smaller than the FNO baseline. These parameters align with the intuition discussed in the Methods section.

Therefore, we are able to find that CNO balances memory, accuracy, and speed, allowing for efficient and reliable acoustic wave simulation.

Iterative Sampling

The goal of this study is to identify a neural operator model that can generate audio samples more efficiently than the FDTD solver without sacrificing too much accuracy. While we have tested a few models above, these were evaluated only on audio samples of limited size (depth of 64 time steps). We now describe an *iterative sampling* method to generate the full audio sample given the velocity and alpha fields but *only* the initial pressure field.

To do this, we choose one of the models trained above and first generate the first 64 time steps of the sample normally, i.e., using the initial pressure field. We then use the last predicted pressure field as the initial condition for generating the next 63 time steps, and so on, repeating until we reach the end of the full audio sample. Concatenating the predicted pressure fields for each iteration gives us an estimate of the full audio sample.

To test the efficacy of this method, we use our best model (i.e., CNO) to iteratively generate an estimate of our evaluation sample ($n = 8757$ time steps). We achieve a MSE of 10.40, outperforming the MSEs of the FNO baseline and TFNO models (Table 1).

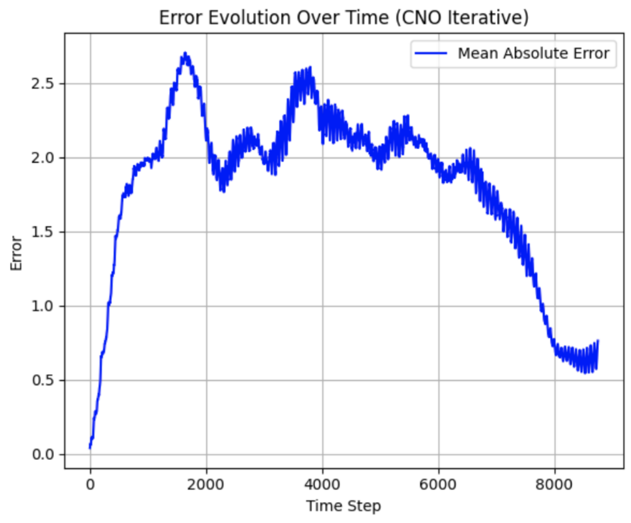

Figure 10. Evolution of the mean absolute error of the iterative CNO estimate over the full evaluation sample ($n = 8757$ time steps).

Figure 10 shows the evolution of the resulting mean absolute error over the full evaluation sample. Unlike the error evolutions in Figure 6, we see an oscillatory pattern here, likely because the iterative CNO estimate should be most accurate for the pressures used as initial conditions, with error going back up as it predicts the next 63 time steps.

Figure 11 shows an example of the CNO iterative estimate compared to the ground truth for a fixed time step. While the estimate sufficiently captures much of the ground truth, it performs especially poorly in a region at the bottom (marked as A in the prediction and B in the error). This artifact could be especially prominent in the middle time steps, which could explain the higher errors from Figure 10.

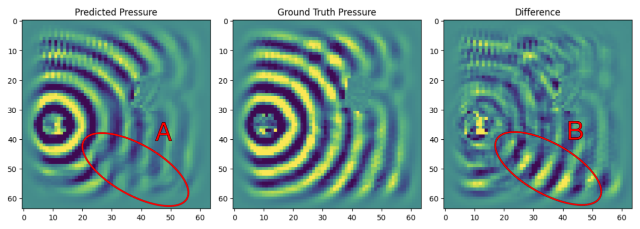

Figure 11. CNO iterative estimate at the middle time step (4441 out of 8757). The areas marked in red correspond to regions in the predicted pressure field (A) and the absolute error (B) where the estimate performed especially poorly.

As such, while we have shown that applying the iterative sampling scheme to the CNO model can result in adequately accurate estimates (albeit with some unwanted artifacts), it is also important to evaluate the increase in efficiency compared to the FDTD solver. On our NVidia Tesla T4 GPU, the FDTD solver generates about 12 steps per second, while we observed rates of 1,800 to 2,400 steps per second for the CNO iterative method. Thus, this method speeds up the audio sampling process by a factor of about 150 to 200, a significant practical advantage compared to the FDTD solver. However, we note that our implementation of the FDTD solver is not fully optimized, which may contribute to the observed performance gap. Future investigation should focus on more optimized solvers and consistent benchmarking to more precisely quantify the efficiency gains of neural operator methods. Nonetheless, neural operators, particularly CNO, demonstrate significant potential for efficient audio wave simulation over traditional numerical solvers.

7. Conclusion and Future Work

This study demonstrates that CNOs offer superior accuracy over other neural operator architectures in simulating acoustic wave propagation, while all tested neural operators provide orders-of-magnitude speedup compared to traditional FDTD solvers. These findings highlight the promise of neural operator models for real-time, physics-based sound synthesis in computer graphics applications.

Our current work is limited by the scope of the dataset, which includes approximately 64,000 samples from only eight scenes with simple geometric configurations of simple and sparsely positioned obstacles. Future work can expand the training dataset to include a wider range of environments, particularly those with multiple, dynamically moving obstacles, more diverse object shapes and dynamic boundary conditions, to improve generalization and robustness. Additionally, as noted in our experiments, CNO exhibits noticeable degradation in the middle of long-range predictions. This issue could be addressed by incorporating time-sequenced models such as RNNs or Transformers, which are better suited for long-horizon temporal modeling and enable autoregressive inference. Furthermore, these include exploring recent advances like GNOT [2], which demonstrate the potential of Transformer-based operator networks in capturing complex spatio-temporal dynamics, making them a promising direction for improving accuracy and stability over longer time horizons.

Thus, while our study establishes the feasibility of neural operator-based methods for fast and accurate simulation, further investigation is required to assess their performance on more complex, real-world audio scenarios. In particular, evaluating time and memory efficiency over longer horizons and larger spatial domains will be important for scaling these models in practical graphics and audio pipelines.

8. Contributions & Acknowledgments

Fangjun Zhou generated the dataset with WaveBlender and trained the FNO baseline. He also designed, implemented, and trained the CNO architecture.

Ngoc Vo designed, implemented, and trained the U-FNO architecture.

Kevin Liu trained the TFNO model and performed the iterative sampling analysis.

References

[1] K. Azizzadenesheli, N. Kovachki, Z. Li, M. Liu-Schiaffini, J. Kossaifi, and A. Anandkumar. Neural operators for accelerating scientific simulations and design. *arXiv preprint arXiv:2309.15325*, 2024.

[2] Z. Hao, Z. Wang, and H. Su. Gnot: A general neural operator transformer for operator learning. *arXiv preprint arXiv:2302.14376*, 2023.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition.

[4] J. Kossaifi, N. Kovachki, K. Azizzadenesheli, and A. Anandkumar. Multi-grid tensorized fourier neural operator for high-resolution pdes, 2023.

[5] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 23(241):1–97, 2022.

[6] B. Li, H. Wang, S. Feng, X. Yang, and Y. Lin. Solving seismic wave equations on variable velocity models with fourier neural operator. *arXiv preprint arXiv:2209.12340*, 2023.

[7] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[8] Q.-H. Liu and J. Tao. The perfectly matched layer for acoustic waves in absorptive media. 102(4):2072–2082.

[9] M. Middleton, D. T. Murphy, and L. Savioja. Modelling of superposition in 2d linear acoustic wave problems using fourier neural operator networks. *Acta Acustica*, 9:20, 2025.

[10] B. Raonić, R. Molinaro, T. De Ryck, T. Rohner, F. Bartolucci, R. Alaifari, S. Mishra, and E. de Bézenac. Convolutional neural operators for robust and accurate learning of pdes. *arXiv preprint arXiv:2302.01178*, 2023.

[11] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[12] J.-H. Wang, A. Qu, T. R. Langlois, and D. L. James. Toward wave-based sound synthesis for computer animation. 37(4):1–16.

[13] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Journal of Computational Physics*, 469:111543, 2022.

[14] K. Xue, J.-H. Wang, T. Langlois, and D. James. WaveBlender: Practical sound-source animation in blended domains. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–10. ACM.

A. Appendix

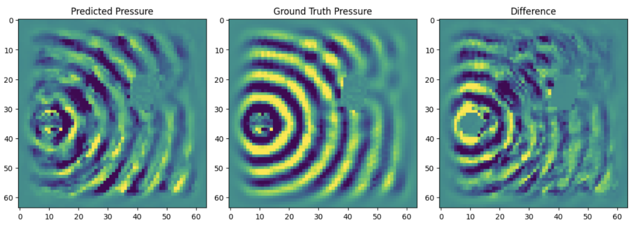

Figure A.1. TFNO Inference Result (final time step)

GitHub repository: https://github.com/fangjunzhou/acoustic-no

Library	Version
`numpy`	1.26.4
`torch`	2.6.0+cu126
`neuralop`	1.0.2
`matplotlib`	3.10.1
`tqdm`	4.67.1
`seaborn`	0.13.2
`argparse`	1.4.0
`multiprocess`	0.70.18

Table 2. Table of Python Libraries Used