A Deep Learning Framework for Musical Acoustics Simulations

AIMC 2024 (09/09 - 11/09)

A Deep Learning Framework for Musical Acoustics Simulations

Jiafeng Chen[1] Kıvanç Tatar[2] Victor Zappi[1]

[1]Northeastern University, USA, [2]Chalmers University of Technology, Sweden

Published on: Aug 29, 2024

URL: https://aimc2024.pubpub.org/pub/5cl1cvmy

ABSTRACT

The acoustic modeling of musical instruments is a heavy computational process, often bound to the solution of complex systems of partial differential equations (PDEs). Numerical models can achieve a high level of accuracy, but they may take up to several hours to complete a full simulation, especially in the case of intricate musical mechanisms. The application of deep learning, and in particular of *neural operators* that learn mappings between function spaces, has the potential to revolutionize how acoustics PDEs are solved and noticeably speed up musical simulations. However, extensive research is necessary to understand the applicability of such operators in musical acoustics; this requires large datasets, capable of exemplifying the relationship between input parameters (excitation) and output solutions (acoustic wave propagation) per each target musical instrument/configuration. With this work, we present an open-access, open-source framework designed for the generation of numerical musical acoustics datasets and for the training/benchmarking of acoustics neural operators. We first describe the overall structure of the framework and the proposed data generation workflow. Then, we detail the first numerical models that were ported to the framework. This work is a first step towards the gathering of a research community that focuses on deep learning applied to musical acoustics, and shares workflows and benchmarking tools.

Author Keywords

Musical Acoustics Simulations, Deep Learning, Numerical Modeling, Acoustics Benchmarking, Datasets

Introduction

The study of the acoustics of musical instruments is a challenging topic. Physics phenomena underlying music making are quite various and include excitation, resonant behavior, as well as the coupling and the dynamic modification of the involved mechanical parts. These make musical instruments remarkable examples of engineering, but also acoustic systems difficult to model. The most accurate simulations that exist today leverage the numerical solution of partial differential equations (PDEs), that are in turn designed to model the specific acoustic behavior of the targeted instruments [1].

Recent advancements in deep learning have shown how neural networks may be used to enhance and even replace traditional PDE solvers [2], with the aim to improve performance. In particular, the use of *neural operators* has yielded promising results in fluids dynamics [3][4], suggesting that their application may be successfully extended to revolutionize the simulation of the acoustics and the aeroacoustics of musical instruments. Being completely data-driven, neural operators could be trained to solve acoustics PDEs with synthetic datasets, generated via the large array of traditional numerical implementations that are available in the literature.

Although exciting, the application of neural operators and other novel architectures to musical instrument simulation still requires extensive investigation to assess its advantages and limitations [5]. Unfortunately, this scenario is hindered by a lack of common practices that are needed to bridge the domains of musical acoustics and deep learning. These include shared datasets, benchmarks, as well as general tools to help researchers categorize, manage and employ acoustics data for training and inference.

The aim of our research is to foster the rapid growth of an active community where these common practices could be discussed and formalized, along with the overall emerging field of deep learning-based musical acoustics. In line with this mission, in this work we present the *Neuralacoustics* framework, a collection of open-access/open-source scripts and tools designed to address the aforementioned needs. In particular, we provide an in-depth description of the dataset generation workflow proposed as part of the framework, and we introduce the first numerical models available in it.

Background

Musical Acoustic Simulations

In the musical domain, the practice of designing mathematical models of instruments is often referred to with the term *physical modeling synthesis*. Common techniques include modal synthesis [6] and digital waveguides [7]. Yet, the most precise techniques rely on numerical analysis [8] (e.g., finite elements, finite differences). Numerical models implement solvers of PDE systems; they can finely simulate fundamental aspects of musical acoustics, like wave propagation and aeroacoustics, as well as physical phenomena beyond instruments and music [9][10]. The downside of numerical approaches lies in the computational load of the resulting models, as well as in the amount of parameters they have to comprise to properly simulate the instrument’s behavior.

Of particular interest to our work is the case of time-domain simulations of musical instruments [1]. In this context, the PDEs solved by the models describe the relationship between previous and next states of the instruments, organized over discrete time steps. Other than taking into account time-varying acoustic excitation of the instruments, this approach potentially enables the design of interactive models.

Despite the high computational requirements of numerical analysis, real-time interactive models of musical instruments have been designed in recent years [11] [12][13]. Unfortunately, this approach relies on expensive dedicated hardware (GPUs) and implementations are characterized by noticeable technical constraints, that limit access to models' parameters and interaction [14]. Aside for a few notable exceptions (e.g., [15]), as a result numerical analysis is employed for the greater part to model simple musical systems [1] [16], or for batch (i.e., non-real-time) simulations [17][10] that may require run-times of several hours.

Deep Learning and PDE Solvers

Recently, deep learning has been successfully explored for the generation of PDE solvers describing time-dependent problems [18] [3]. These neural solvers may reduce the overall computational requirements of traditional ones, while approximating their output with a remarkable degree of precision. One of the simplest examples of neural solvers consists of deep convolutional neural networks parametrizing the operator that maps inputs and outputs (i.e., solutions) of the PDEs [2][19]. The limitation to this approach lies in its dependence on the chosen mesh, meaning that it is not possible to compute solutions outside the discretization grid used for training. Physics informed neural networks solve this issue, as they are mesh-independent and designed to work alongside classical schemes (e.g., Runge-Kutta) [20]. Although capable of addressing problems in the small data setting and with high dimensionality [18], they are often employed to solve time-dependent PDEs that share many similarities with the ones modeling musical acoustics---e.g., Navier-Stokes equations [21][22][23]. Being only partially data-driven, this approach requires to tailor the network to a specific instance of the PDEs and to repeat training at any given new input.

Most of the individual advantages of the approaches introduced so far are collated in neural operators [3]. Neural operators are mesh-free operators that require no prior knowledge of the underlying PDEs. They learn mappings between infinite-dimensional spaces of functions relying only on a finite collection of observations; and they can be used without retraining to solve PDEs with different discretizations. Although recent, they showed promising results not only in fluid dynamics [3], but also in the solution of wave equations [24] \citep{guan2021fourier}.

Deep Learning and Musical Acoustics

The application of deep learning to musical acoustics simulations is less straightforward than what it may seem. In the most general sense, the problem can be framed as mapping the state of a numerical model across the last T_{in} time steps to its state across T_{out} future time steps. Both convolutional neural networks [25][26] [27] and neural operators can be used to approximate this mapping with high degrees of confidence [3], in most cases treating the input tensors like a time series of images/video frames. To this end, such networks have been trained by using numerical datasets that exemplify how the target acoustics model evolves over time given a set of initial conditions; and during inference, they can be used auto-regressively to generate a continuous output. These are promising results and stem from a working scenario that appears to align well with the general domain of acoustic problems.

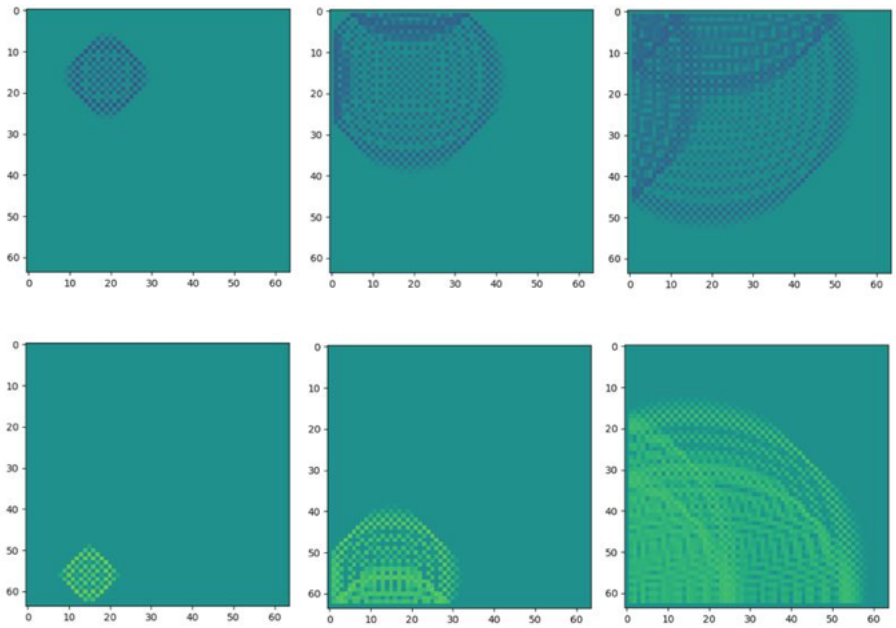

Figure 1
Visualization of a 2-dimensional membrane model. Under different excitations, the top and the bottom row each shows three snapshots of the evolution of the membrane's displacements over time.

However, these examples of PDE neural solvers do not take into account two important aspects that are specific to musical acoustic simulations. The first one pertains to the excitation of musical models as exemplified in Figure 1. Rather than simply simulating the behavior of an instrument set into motion by an initial condition, acoustics numerical models can account for the effects of *continuous excitation* functions, that may drive the instrument throughout the full duration of the simulation. Examples of continuous excitations include basic sinusoidal waves, as well as gaussian pulses used to simulate mallet strikes on membranes and plates [28], and glottal pulse trains that resonate in singing vocal tracts [29][30]. In more advanced simulations, continuous excitation is not pre-computed; it is outputted by a self-oscillating system coupled with the main acoustics model, a common example being a reed coupled with the bore of a woodwind [31][12] . An excitation can start at any given time step of the simulation and the effects of consecutive/overlapping functions may be quite difficult to predict, especially in non-linear models. The training strategies explored so far in deep learning to predict the solution of time-dependent PDEs are not designed to capture this aspect of musical interaction.

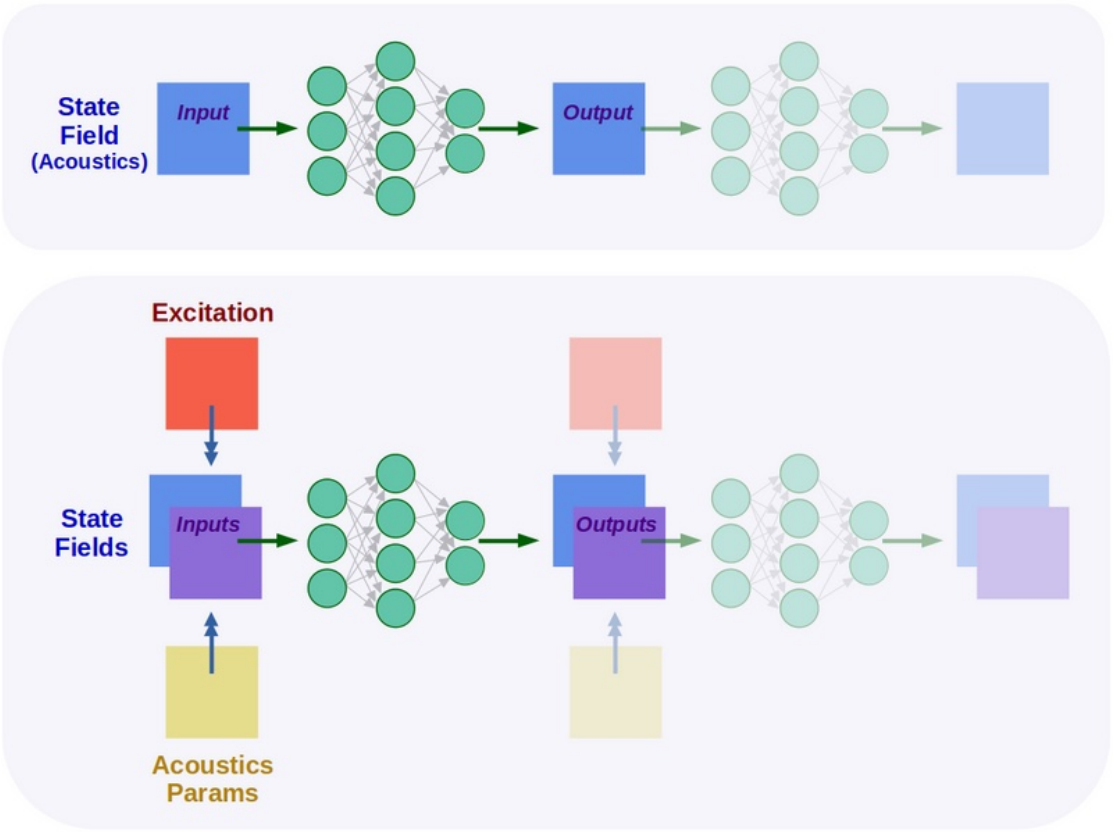

Figure 2
Neural network approximating the mapping between input/output states representing the same physical quantity (top); neural network approximating the mapping between heterogeneous input/output states, driven by continuous excitation and dynamic acoustics parameters (bottom). Only the first working scenario has been explored in the literature.

A second aspect that is missing from the current state-of-art time-dependent PDE neural solvers is input/output heterogeneity in Figure 2. To allow for the synthesis of the instruments' sound, a numerical model has to output a field representing the physical quantity where the acoustic wave is propagating. Figure 1 shows a practical example of the model of a membrane that outputs how its displacement changes over time, with respect to its equilibrium position. In the simplest case, the output acoustic field represents the next state of the system. This means that, at any given time t, the latest output of the PDE becomes the input for the computation of the solution at time $t + 1$. This simple mapping can be approximated by networks characterized by feature and prediction tensors that represent the same single (acoustic) quantity, and that consequently allow for a straightforward auto-regression mechanism during inference. To our knowledge, this is the only working scenario explored in the literature on the application of deep learning to time-dependent PDEs (e.g., input/output flow velocity [27], input/output vorticity [3], input/output displacement [5]). Yet, in more complex numerical models the state of the system may be composed of more than one field, each representing a different physical quantity. This is required when the model implements an implicit solver, meaning that the

next acoustic output is calculated via the solution of a coupled system of equations—hence the necessity for an extra state quantity. Other examples include models where the boundary conditions and/or the acoustic properties of the simulated materials vary over time, due to musical interaction [16][13]. We can refer to them as *acoustics parameters*.

Moreover, these two aspects of musical instruments' numerical modeling are often intertwined. While in some cases the excitation is directly applied to the field where wave propagation is simulated (and sampled for audio output), in many other models it is the additional state field that gets altered by the excitation function, only implicitly affecting the simulation output. This is the case for the simulation of woodwinds, where the state of the system is represented by both an acoustic pressure field and a flow velocity field, the latter carrying the velocity excitation incoming from the reed. In similar contexts, the input/output state mapping (including the excitation mechanism) is not trivial to approximate via a neural network, and the methodologies discussed in the literature cannot be applied in a straightforward manner.

Novel network designs and training strategies may be explored that model these aspects of musical acoustics via deep learning. The fundamental requirement for the implementation of such algorithms is the availability of large datasets, that carry all the information needed to frame both the acoustic behavior of the simulated instrument and the inner workings of the solvers. This translates into storage of full state fields and excitations, along with standardized access methodologies for the extraction of data points as part of training sets.

Dataset Generation for Benchmarking and Model Training

The Neuralacoustics framework stems from the necessity to generate musical acoustics datasets that could be easily employed in deep learning. It consists of a collection of Python implementations of numerical models of musical instruments, embedded in a modular structure that facilitates extensibility and allows for the application of a model-independent workflow. Its overall structure and some of its features were inspired by the repository that in 2021 accompanied the work of [3]. While building on this previous work, as discussed in the previous section we propose a framework that is specifically tailored to the case of musical instrument modeling and designed for extensibility. From a data-centric perspective, the design specifications of our framework adheres to the following constraints: the output of the acoustics simulations must be organized in data structures that are compatible with standard machine learning frameworks; such data structures must be easy to move between local and remote machines; and the output of each simulation must be easy to replicate.

The proposed framework can be accessed here. It is written in Pytorch and requires the installation of a few additional libraries, mainly for the visualization of the acoustics simulations and for logging purposes. Pytorch was chosen for its flexibility and scalability on parallel architectures, yet the resulting datasets are not tied to this specific language (more details in Section Dataset Generation Workflow). The dataset generation workflow that we propose acts upon three main types of components/scripts: solvers, numerical models and

dataset generators. In the following subsections, we detail these components and then we introduce the workflow in each of its steps.

Synthetic Datasets for Musical Acoustics

Solvers. Solvers implement numerical solutions of highly parametric PDE systems, capable of modeling entire families of musical instruments. Regardless of the numerical method employed (e.g., finite difference, finite elements), all solvers have both a set of specific acoustics parameters, that depend on the implementation details, and a set of common parameters (e.g., domain size, simulation duration).

Currently, the framework includes five solvers, all based on finite-difference time-domain schemes. The first one was originally proposed by [32] and solves a PDE system capable of the simulation of damped transverse wave propagation in a two-dimensional medium. It can be used to model the basic acoustics of membranes, thin plates and rudimentary cymbals. We also included a linear and a non-linear variant of this simpler solver that include loss terms [33]. The last two solvers tackle acoustic pressure propagation in 2D, and were ported from the OpenGL implementations proposed by [12]. The former is linear and it can be used to approximate woodwind bores; the latter includes non-linearities typical of brass instruments.

Numerical Models. Numerical models simulate specific musical instruments. To do so, each of these scripts loads a solver and sets some of its acoustics parameters, imposing for example constraints on the shape of the domain (i.e., spatial boundary conditions) or on the acoustic properties of the simulated materials. Furthermore, numerical models are characterized also by an excitation algorithm. Excitations can work as initial conditions simply aimed at setting the model into motion (e.g., initial displacement of a membrane), or as continuous models that excite the instrument throughout the whole duration of the simulation.

By setting the parameters of the underlying PDEs and defining an excitation input, numerical models can be used to simulate not only different instruments, but also different playing configurations of the same instrument (e.g., a membrane hit by a stick or by a mallet).

Each numerical model exposes controllable parameters. These include those pertaining to the excitation algorithm implemented in the script (e.g., the location of a hit, the area where an initial condition is applied), as well as any parameter of the solver that is not hard-coded by the model. These controllable parameters provide the ability to “tune” the behavior of the instrument and allow for the generation of datasets via the mechanism described in the next paragraphs.

For now, we included into the framework only six models, based on the aforementioned solvers. In particular, we implemented three models that simulate vibrating membranes, using as excitations impulses, noise and single spatial frequencies that match the horizontal modes of the surface, respectively. The fourth and fifth model simulate a stiff memebrane and a thin plate respectively, both excited via impulses. The last model leverages the linear acoustic pressure propagation solver to excite woodwinds bores with air wave pulses.

Dataset Generators. Dataset Generators consist of algorithms that load a specific model and automatically sample its parameters' space, effectively running large numbers of simulations of the same instrument/configuration. The framework includes two types of generators: random generators and grid-based generators. Random generators explore the parameters' space of the model using Python/Pytorch pseudo-random algorithms, driven by an arbitrary seed. This ensures determinism while avoiding clear patterns, and facilitates the reproducibility of results across different machines[2]. The number of random samples to take (i.e., the size of the resulting dataset) is passed to the generator as an input parameter. Grid-based generators do not rely on any random calculation, rather they sample the parameters' space in a linearly fashion. Each parameter to sample is assigned a range and a sampling step, then all the possible combinations of parameters are automatically computed—forming a “grid”. Differently from the case of random generators, the total number of samples is not arbitrary but depends on the defined grid, and the resulting datasets need to be shuffled before being used for training.

Much like the case of numerical models, dataset generators expose a set of parameters. Example generator parameters include ranges and steps of the model's parameters to sample (for grid-based generators), the requested number of dataset entries and the current seed (for random generators).

Dataset Generation Workflow

The main component scripts are not designed to be run directly. The framework includes instead a launcher dataset generation script, that allows for the correct use of generators, models and solvers, and partially hides the complexity of the underlying code and dependencies. Once launched, this script collects all the simulation data computed by a chosen generator script into an actual dataset. More specifically, dataset entries represent complete simulations, each associated to a different set of parameters. In line with what discussed in Section Deep Learning and Musical Acoustics, they consist of dictionaries containing all the inputs to the model (excitations and variable acoustics parameters[3]) and output solutions encompassing all state fields (rather than the acoustic output only), for every simulation time step.

The way in which datasets are structured and stored reflects the first two data-centric constraints we introduced at the beginning of this section, i.e., compatibility and portability. The launcher dataset generation script outputs *MAT-files* (“.mat” extension), one of the de-facto standard data formats in machine learning. The generation of acoustics datasets may yield very large files, especially when simulations span big domains and long time windows. To avoid exceeding the maximum size supported by the native file system where the code runs, the launcher dataset generation script is capable of splitting the dataset into several “chunks”, each represented by an individual MAT-file. This solution comes in handy also when moving datasets between remote locations, for the transfer of large files may be subject to failures due to connection instability. Eventually, when a dataset is loaded in memory (to train a network or to visualize its content), all the chunk files are transparently combined together back into a single dataset (more details in the next subsection).

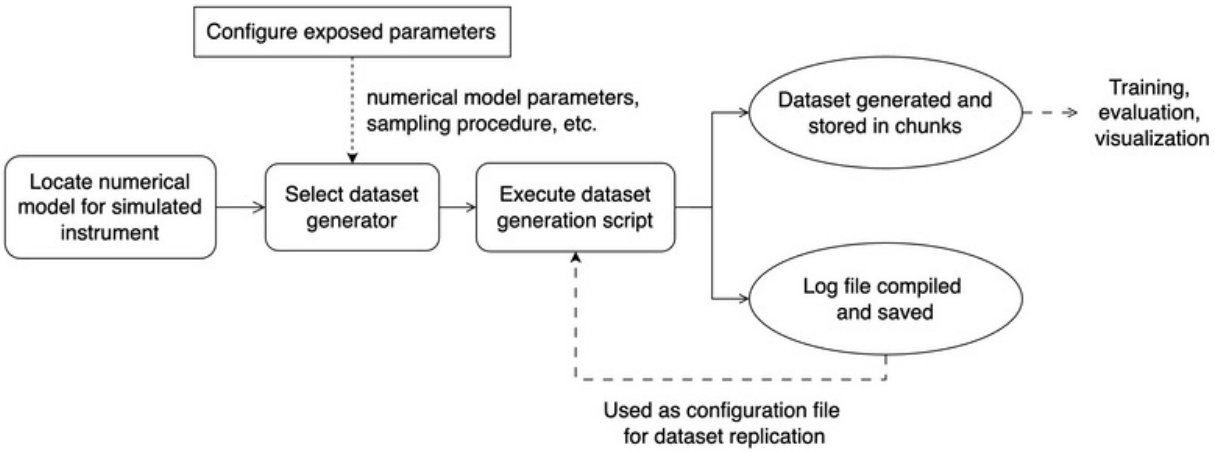

Figure 3
Flowchart for dataset generation process.

The complete dataset generation workflow can be summarized in Figure 3. First, the user has to locate a numerical model that represents the specific instrument the dataset will exemplify. Then, a dataset generator needs to be chosen, that samples the numerical model of interest. In this step, the user shall adjust the exposed parameters to make sure that the sampling procedure will result in data that well represent the instrument and its specific playing configuration, depending on purpose and application of the model. Finally, the user can set and run the launcher dataset generation script and the resulting dataset will be computed and stored according to the requested settings.

The launcher script compiles a log file too. It contains a summary of the content of the dataset and reports location and all parameters of the employed scripts. Any log file can then function as a unique configuration file that, when passed to the launcher script, allows users and remote collaborators to automatically obtain an exact copy of the original dataset, avoiding the hassle of moving large files or going through the full workflow again. The only caveat is that the same version of the framework has to be installed on both ends. This mechanism was designed to respect the third data constraint (replicability). Moreover, every step of the workflow can be carried out via command line, making it straightforward to check out the framework on remote machines and generate datasets on high performance computing clusters.

Utility Tools: Visualization and Data Points Extraction

At the root of the framework, we have two specific approaches, one designed to test numerical models, the other to visually inspect the content of datasets. Both scripts sequentially visualize the solution fields as plots, showcasing the propagating acoustic waves as an animation. The main difference between the two approaches is that the former computes the frames in real-time by means of running the tested model, while the latter extracts them from the inspected dataset.

The dataset visualization approach implements a windowing system that allows for the extraction of diverse sets of data points from the same dataset. To understand this mechanism, it is necessary to emphasize the difference between an “entry” within the dataset and a “data point” extracted from it. Each entry in a Neuralacoustics dataset consists of a time series, representing the simulation of an instrument over a certain number of time steps. In the most general sense, a Neuralacoustics data point can be any sub-series of consecutive time steps found in a dataset entry. More than one data point can be extracted from a single dataset entry and the maximum size of a data point is equal to the total number of time steps of the simulation. In the latter case, only a single data point is extracted, which coincides with the full entry.

The windowing algorithm is part of a data point extraction tool, that retrieves data points by means of repeatedly windowing the entries' time series (collecting a data point per window). The process is depicted in Figure 4.

The main windowing parameters are: the size of the windows, the stride applied between consecutive windows and the dataset entry where the windows are applied. A further parameter allows to repeat the extraction over a number of consecutive entries, increasing the total number of frames visualized. To simply visualize the full simulations within each entry, the user can set either the size of the window equal to the number of time steps of each entry, or the stride equal to the size of the window.

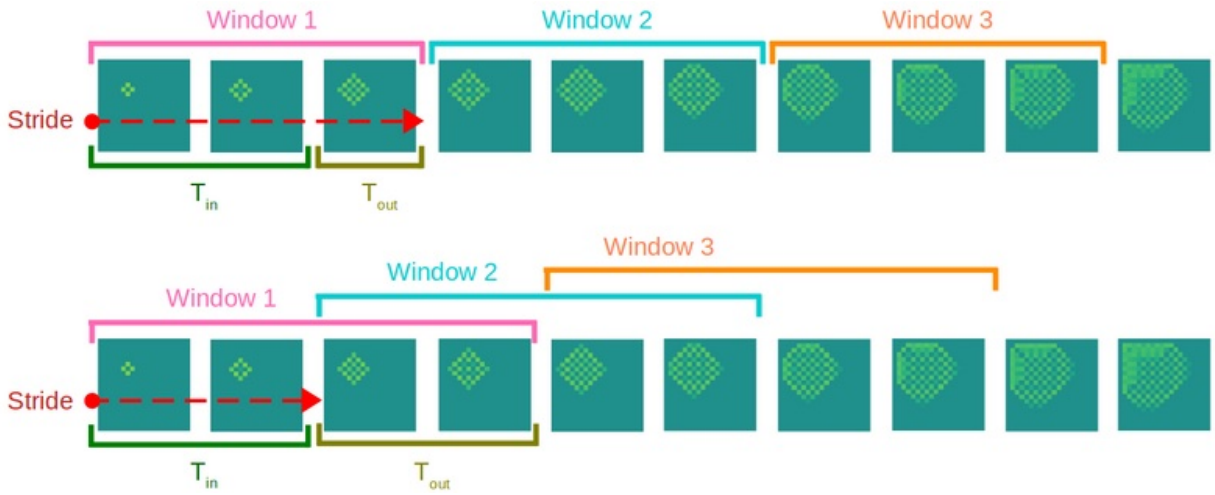

Figure 4

Windowing algorithm used to extract data points from Neuralacoustics datasets. The same dataset entry is processed via two different sets of parameters, to exemplify the extraction of different data points. The diagram depicts only the solution time steps (acoustic fields) stored in the entry's dictionary.

When used to collate a training set, the extraction tool keeps applying the windows across each entry of the dataset, until the number of data points requested is obtained. It is important to notice, though, that the values returned by the tool contain more information than simple feature vectors, for two reasons. First, the size of the window is defined as $T_{win} = T_{in} + T_{out}$; hence, the extracted values contain both the states pertaining to

the initial T_{in} time steps (i.e., the features) and the states pertaining to the following T_{out} time steps (i.e., the label to predict). Second, the values stored in each dataset entry may include heterogeneous inputs and outputs. When extracted, these pieces of data may be combined in different ways to represent the actual state of the model in each time step, depending on the specifics of the problem and of the solver. This means that some extra logic is necessary to define features and labels and be able to employ the data points for training. While increasing the overall complexity of the framework, this design detail makes data handling as generic as possible, allowing for the application of the same workflow even when very different numerical models/simulations/solvers and neural architectures are in use. We share the details of a simple example of data handling logic for training in the next section.

Training Neural Operators

The Neuralacoustics framework also aims to facilitate the design of neural architectures capable of solving acoustics PDEs. Similarly to numerical models, neural architectures are implemented in Pytorch and expose hyperparameters. A two-dimensional Fourier neural operator as introduced in [3] is available in the framework. This was chosen to be ported into the framework because of its ability to solve time-based problems, as well as for its somewhat simple internal structure[4]. The hyperparameters exposed by the port include spectral layer stack number, hidden size and Fourier transform modes. The network performs prediction one time step at a time and produces consecutive results in an auto-regressive manner.

The hyperparameters of training includes dataset and network selection, size of training and validation sets, data points extraction details (e.g., T_{in} and T_{out}, learning and optimization parameters. This separation of training parameters and network hyperparameters ensures generalizability, potentially supporting the training of networks that diverge from the structure of the implemented Fourier neural operator. The training operates on the dataset via the data point extraction tool described in Section *Utility Tools: Visualization and Data Points Extraction*. The logic used to combine inputs (excitations) and outputs (solution fields) into state tensors (i.e., definition of features and labels). Per each dataset entry, features are defined as the first T_{in} state tensors within the extracted time window, each computed as the sum between the current acoustic solution and the excitation at the next time step. This is the simplest way to embed continuous excitation into the input state of a numerical model. The label of each data entry consists of the last T_{out} state tensors, simply defined as the acoustic solutions in those time steps. Once both data points and network are set, the training of the neural model carries on with the specified learning parameters. Weight checkpoints are also saved, using a customizable step interval.

Our framework includes features to achieve reproducibility. Similarly to the data generation workflow, the training compiles a log file that serves as a summary of the training details. All parameters related to the training process, alongside dataset generation parameters and network hyperparameters, are recorded so that the log file itself could be used as a unique configuration file for thoroughly replicating the neural model.

The framework additionally allows an ease in evaluation of trained models by providing intuitive insights into a trained network's performance. The users can select the network's checkpoint to evaluate, as well as the exact dataset and data entry for running inference on. The evaluations include visualizations of the predicted domain state along with the ground truth acoustic solution (computed by the original numerical model and stored in the dataset) and their difference for a chosen number of time steps.

Conclusion and Future Work

In this paper, we introduced the Neuralacoustics framework, an open access and open source collection of tools designed to facilitate the application of deep learning in the context of acoustics simulations and musical instrument modeling. In particular, the framework responds to the need for standards to combine the output of diverse acoustics simulations into datasets, and to use them for training.

The main components of the framework are numerical models and acoustic PDE solvers. These are arranged in a modular structure, that permits the application of a robust workflow for the generation of heterogeneous musical acoustics datasets. The generation process outputs data structures compatible with standard machine learning frameworks, and is designed to maximize portability and reproducibility. The Neuralacoustics framework features also a section dedicated to the training and the evaluation of neural operators using the generated acoustics datasets. While still in progress, this part of the workflow is functional and leverages a modular structure similar to the one proposed for the dataset generation process.

At its current stage, the framework includes Pytorch implementations of six numerical models and five solvers. These first implementations are designed to work as blueprints for the porting of additional models and solvers. With the release of this work, we aim to empower all the researchers working in this emerging field with new tools for the development and the sharing of their own implementations. The release of frameworks, common practices and benchmarks have long benefited the advancement of machine learning as well as its application in various domains [34][35][36][37]. We believe that our effort can have a similar impact on the development of novel deep learning approaches to acoustics modeling, and can facilitate the onset of collaborations among researchers from both fields.

Ethics Statement

This research did not involve any human participants or animals. The datasets utilized in this work are synthetically generated using mathematical models. Thus, there are no data related issues such as copyright, privacy, or consent. The computation required to create this work was comparable to daily personal computer usage; and thus, we predict the environmental impact of this work as minimal. This work shares a reproducible open-source code to replicate the study results with sustainability considerations. Accessibility of our work is dependent on the general accessibility to computers and its development frameworks. Our work is mainly mathematical and theoretical, and our bibliography covers a global network of researchers. There is no potential conflict of interest related to this work within our knowledge.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program—Humanity and Society (WASP-HS), funded by the Marianne and Marcus Wallenberg Foundation and the Marcus and Amalia Wallenberg Foundation.

Footnotes

1. These numerical models can be deemed as “simple” only if compared to the complexity of actual acoustic instruments. ↩

2. We though noted that Pytorch deterministic algorithms may result in slightly different output numbers when the same generator is executed on different computer architectures. ↩

3. Currently, all the models ported to the framework sport fixed acoustics properties, hence the dictionaries generated from these models do not include variable acoustics parameters as part of the input tensors. ↩

4. The source code of the Fourier neural operators is available here. ↩

References

1. Bilbao, S. (2009). *Numerical sound synthesis: finite difference schemes and simulation in musical acoustics*. John Wiley & Sons. ↩
2. Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., & Kaushik, S. (2019). Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, *64*(2), 525–545. ↩
3. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020). Fourier neural operator for parametric partial differential equations. *arXiv Preprint arXiv:2010.08895*. ↩
4. Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., … Anandkumar, A. (2021). Physics-informed neural operator for learning partial differential equations. *arXiv Preprint arXiv:2111.03794*. ↩
5. De La Vega Martin, C., Sandler, M., & others. (2023). Physical Modelling of Stiff Membrane Vibration using Neural Networks with Spectral Convolution Layers. *Forum Acusticum 2023*. ↩
6. Causse, R. E., Bensoam, J., & Ellis, N. (2011). Modalys, a physical modeling synthesizer: More than twenty years of researches, developments, and musical uses. *The Journal of the Acoustical Society of America*, *130*(4 journal=Computer music journal, volume=16, number=4,), 2365–2365. ↩
7. Smith, J. O. (1992). Physical modeling using digital waveguides. *Computer Music Journal*, *16*(4), 74–91. ↩

8. Castagné, N., & Cadoz, C. (2003). 10 criteria for evaluating physical modelling schemes for music creation. *DAFx03*, 7-p. ↩

9. Yokota, T., Sakamoto, S., & Tachibana, H. (2002). Visualization of sound propagation and scattering in rooms. *Acoustical Science and Technology*, *23*(1), 40–46. ↩

10. Arnela, M., & Guasch, O. (2014). Two-dimensional vocal tracts with three-dimensional behavior in the numerical generation of vowels. *The Journal of the Acoustical Society of America*, *135*(1), 369–379. ↩

11. Sosnick, M., & Hsu, W. (2010). Efficient finite difference-based sound synthesis using GPUs. *Proceedings of the Sound and Music Computing Conference*, 42–44. ↩

12. Allen, A., & Raghuvanshi, N. (2015). Aerophones in flatland: Interactive wave simulation of wind instruments. *ACM Transactions on Graphics (TOG)*, *34*(4), 1–11. ↩

13. Zappi, V., Allen, A., & Fels, S. S. (2017). The Hyper Drumhead: Making Music with a Massive Real-time Physical Model. *Proceedings of the 2017 International Computer Music Conference, ICMC 2017, Shanghai, China, October 16-20, 2017*. Michigan Publishing. Retrieved from https://hdl.handle.net/2027/spo.bbp2372.2017.026 ↩

14. Renney, H., Willemsen, S., Gaster, B., & Mitchell, T. (2022). HyperModels-A Framework for GPU Accelerated Physical Modelling Sound Synthesis. *NIME*, *22*. PubPub. ↩

15. Wang, Z., Puckette, M., Erbe, T., & Bilbao, S. (2023). Real-Time Implementation of the Kirchhoff Plate Equation Using Finite-Difference Time-Domain Methods on CPU. *2023 Sound and Music Computing Conference (SMC)*. ↩

16. Bilbao, S., Ducceschi, M., & Webb, C. (2019). Large-scale real-time modular physical modeling sound synthesis. *Proc. Int. Conf. On Dig. Audio Eff.(DAFx 2019), Birmingham, UK*. ↩

17. Bilbao, S., & Chick, J. (2013). Finite difference time domain simulation for the brass instrument bore. *The Journal of the Acoustical Society of America*, *134*(5), 3860–3871. ↩

18. Blechschmidt, J., & Ernst, O. G. (2021). Three ways to solve partial differential equations with neural networks—A review. *GAMM-Mitteilungen*, *44*(2), e202100006. ↩

19. Khoo, Y., Lu, J., & Ying, L. (2021). Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, *32*(3), 421–435. ↩

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

21. Rudy, S. H., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2017). Data-driven discovery of partial differential equations. *Science Advances*, *3*(4), e1602614. ↩
22. Font, B., Weymouth, G. D., Nguyen, V.-T., & Tutty, O. R. (2021). Deep learning of the spanwise-averaged Navier–Stokes equations. *Journal of Computational Physics*, *434*, 110199. ↩
23. Cai, S., Mao, Z., Wang, Z., Yin, M., & Karniadakis, G. E. (2022). Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 1–12. ↩
24. Guan, S., Hsu, K.-T., & Chitnis, P. V. (2021). Fourier neural operator networks: A fast and general solver for the photoacoustic wave equation. *arXiv Preprint arXiv:2108.09374*. ↩
25. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778. ↩
26. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 234–241. Springer. ↩
27. Wang, R., Kashinath, K., Mustafa, M., Albert, A., & Yu, R. (2020). Towards physics-informed deep learning for turbulent flow prediction. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1457–1466. ↩
28. Sosnick, M. H., & Hsu, W. T. (2011). Implementing a Finite Difference-Based Real-Time Sound Synthesizer Using GPUs. *NIME*, 264–267. Citeseer. ↩
29. Rosenberg, A. E. (1971). Effect of glottal pulse shape on the quality of natural vowels. *The Journal of the Acoustical Society of America*, *49*(2B), 583–590. ↩
30. Guasch, O., Arnela, M., Codina, R., & Espinoza, H. (2016). A stabilized finite element method for the mixed wave equation in an ALE framework with application to diphthong production. *Acta Acustica United with Acustica*, *102*(1), 94–106. ↩
31. Bilbao, S., Torin, A., & Chatziioannou, V. (2015). Numerical modeling of collisions in musical instruments. *Acta Acustica United with Acustica*, *101*(1), 155–173. ↩
32. Adib, A. B. (2000). Study notes on numerical solutions of the wave equation with the finite difference method. *arXiv Preprint Physics/0009068*. ↩
33. Torin, A. (2016). *Percussion instrument modelling in 3d: Sound synthesis through time domain numerical simulation* (PhD Thesis). University of Edinburgh. ↩

34. Downie, J. S., West, K., Ehmann, A., & Vincent, E. (2005). The 2005 music information retrieval evaluation exchange (mirex 2005): Preliminary overview. *6th Int. Conf. on Music Information Retrieval (Ismir)*, 320–323. ↩

35. Schedl, M., Gómez, E., Urbano, J., & others. (2014). Music information retrieval: Recent developments and applications. *Foundations and Trends® in Information Retrieval*, *8*(2–3), 127–261. ↩

36. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., … Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems*, *33*, 22118–22133. ↩

37. Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., … others. (2021). Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv Preprint arXiv:2102.04664*. ↩