CS6P05 Final project report

Exploring the Optimisation of High-Quality Reverb Simulations using Fourier Neural Operators

Name:	Gabriel Penman
ID Number:	22030670/1
Date:	11/03/2024
First Supervisor:	Dr Maitreyee Dey (m.dey@londonmet.ac.uk)

Abstract

This project concerns the optimisation and dynamic generation of high quality reverb, using Fourier Neural Operators (Li et al., 2021) for more efficient performance on lower-end hardware.

This artefact has been produced using the agile Cowboy software methodology (Hollar, 2006). I have used this throughout the development of this program to adapt to difficult hurdles in this more speculative area, whilst maintaining as many of my original objectives as possible.

The program is a functioning implementation of a FNO architecture which is able to quickly synthesise accurate room impulse responses, RIRs, based on unseen parameters, faster than traditional hard coded methods. It also contains the software needed to generate the required training data and evaluate the results of the model.

Though I was not able to get the model to converge to a meaningful degree, I believe that the software is capable of it with the right hyperparameters and better hardware.

Introduction

Project topic and rationale

The project focuses on optimising the dynamic generation of high-quality reverb using Fourier Neural Operators (FNOs). Reverb is a continuous and complex calculation that accounts for diffusion, absorption, spatial geometry, and time variance. While convolutional reverb is a relatively efficient method, it cannot perfectly model real-world reverberation involving complex non-linear interactions with the environment.

This project aims to leverage FNOs, a type of neural network architecture designed to learn operators that map between infinite-dimensional function spaces governed by partial differential equations (PDEs) like the wave equation. FNOs have the potential to model the intricate patterns of reflections, diffractions, and interference that create rich, enveloping ambiance more accurately than traditional methods, while maintaining efficient performance on lower-end hardware.

The motivation behind this project is to address the computational bottleneck associated with high-fidelity reverb generation, which is computationally intensive, especially when modelling complex 3D environments. By employing FNOs, the project aims to optimise the dynamic generation of high-quality reverb, enabling improved performance on lower-end hardware compared to traditional methods.

Project Aims and Objectives

- Implement Fourier Neural Operator (FNO) architectures for efficient room impulse response (RIR) synthesis
- Generate and visualise a diverse and high-quality dataset of RIRs and corresponding room parameters for training the FNO models.
- Train FNO models to accurately predict RIRs based on input room parameters, such as dimensions, materials, and acoustic properties.
- Optimise the FNO models to find the best parameters
- Evaluate the trained FNO models by comparing their generated RIRs with ground truth data and assessing the quality of the simulated reverb through auditory and visual analysis.

Methodology

The project followed the agile Cowboy software methodology, which emphasises iterative development and frequent delivery. This approach allowed for breaking down the overall goals into smaller, manageable iterations, enabling early testing, evaluation, and course correction based on feedback and insights gathered at each stage.

Report content and structure

1. Background Research

Foundational research

Challenges in Optimising Convolution Reverb using ML

The initial stages of the project aimed to optimize resource-intensive plugins using novel ML techniques, inspired by the success of ML in complex tasks like upscaling in video games and solving the LPN problem using MLP neural networks. However, the irony and logical fallacy of using neural networks, such as convolutional neural networks (CNNs), to optimise a convolution process were not immediately apparent to me.

Further investigation into optimising CR using ML revealed additional challenges due to the efficiency of existing processes like Fast Fourier Transforms (FFTs). The exploration of smaller neural networks for possible quality-speed trade-offs was constrained by their higher time complexity compared to FFTs, $O(n^2)$ vs $O(n\log n)$ (CPU and GPU Implementations, n.d.b), and their inefficiency in approximating FFTs. This led to a shift in focus towards other forms of high-quality reverb suitable for ML optimization.

High-Quality Reverb Plugin Efficiency

Recent advancements in the field of graphics had demonstrated to me that neural networks could be leveraged to solve complex processes faster than traditional methods. Examples include deep learning for upscaling in video games and solving the learning parity with noise problem (LPN) using MLP neural networks (Jiang et al., 2023).

Through community consultation, as part of the software methodology, it was found that reverb could be exceptionally resource-demanding in some contexts. Reverb is a continuous and complex calculation that accounts for diffusion, absorption, spatial geometry, and time variance. Initially, the project aimed to optimise a small number of plugins and integrate them with the second part of the project, which would leverage natural language processing (NLP) to interface with them in original ways. However, realising the speculative and technical nature of this task in the context of reverb, I shifted the focus entirely to the optimization process.

Convolution reverb (CR) is generally used as a relatively high-quality and efficient method for computing reverb, but it cannot perfectly model real-world reverberation, which involves complex non-linear interactions with the environment that linear processes like CR cannot capture.

Following this, I sought to optimise the highest quality reverberation since there was more room for this kind of work.

Accurate simulation of the complex echo patterns in real-world acoustic spaces requires more advanced modelling techniques like convolving the dry audio signal with very long/detailed room impulse responses (RIRs). The increased complexity means these high-fidelity reverbs put more strain on CPU/GPU resources.

Acoustic wave field

The propagation of sound waves within an acoustic environment is governed by the wave equation - a linear partial differential equation (PDE) that describes how factors like air pressure fluctuations transmit through a medium over time. (Feynman, 2006/1963)

Solving the wave equation numerically allows simulating the intricate patterns of reflections, diffractions, and interference that create the rich, enveloping ambiance we perceive as reverb. However, conventional numerical techniques for solving PDEs can be computationally intensive, especially when modelling complex 3D environments.

This computational bottleneck for high-fidelity reverb motivated me to read about novel machine learning approaches, where I came across the work on Fourier Neural Operators (FNOs). FNOs are a type of neural network architecture designed specifically for learning operators that map between infinite-dimensional function spaces governed by PDEs like the wave equation.

Fourier Neural Operators (FNOs)

Fourier Neural Operators (FNOs) have emerged as a promising approach for optimising the dynamic generation of high-quality reverb. FNOs are a class of neural networks that leverage the power of Fourier transforms to learn complex mappings between function spaces. By operating in the Fourier domain, FNOs can efficiently capture long-range dependencies and model non-linear interactions, making them well-suited for tasks like reverb generation.

The key advantages of FNOs include their ability to handle high-dimensional data, their computational efficiency, and their capacity to learn intricate patterns and relationships. These properties make FNOs an attractive choice for optimising reverb generation, as they can potentially model the complex interactions between sound waves and the environment more accurately than traditional methods while maintaining efficient performance on lower-end hardware.

In the context of this project, the objective is to employ FNOs to optimise the dynamic generation of high-quality reverb. By training FNOs on a diverse dataset of room impulse responses (RIRs) and corresponding reverb parameters, the aim is to create a model that can efficiently generate realistic and high-quality reverb based on unseen parameters, ultimately leading to improved performance on lower-end hardware compared to traditional methods like convolution reverb.

RIRs

RIRs are essentially a way of capturing the unique acoustic characteristics of a space in a single recording, providing a comprehensive representation of how sound propagates and interacts with the environment.

The importance of RIRs lies in their ability to encapsulate the complex acoustic phenomena that occur within a space, such as reflections, diffractions, and absorption. When an impulse signal, such as a short click or snap, is emitted in a room, the resulting sound waves interact with the surfaces and objects present in the environment. These interactions give rise to the distinct reverberations and echoes that we perceive as the room's acoustic signature.

RIRs capture this acoustic signature by recording the room's response to the impulse signal at a specific receiver (microphone) location. The resulting recording contains information about the direct sound path from the source to the receiver, as well as the subsequent reflections and reverberations that follow. By analysing the RIR, we can extract valuable insights about the room's geometry, materials, and acoustic properties, such as reverberation time (RT60) and early decay time (EDT).
(https://personalpages.surrey.ac.uk/p.jackson/ee2.lab/XY_rir/)
(https://www.gearank.com/impulse-responses)

Data generation

Having settled on the implementation of an FNO for the means of RIR synthesis. I understood that in order to train a high-speed model effectively, appropriate training data is of course essential.

The first step was defining the scope and characteristics of the desired dataset. This involved considerations like the range of room dimensions, materials, and acoustic properties to cover, ensuring the FNO learns to generate realistic and diverse reverb outputs. I also needed to determine target quality metrics like sampling rate, bit depth, and RIR length to meet high-fidelity audio requirements.

Traditionally, RIRs are generated through computational acoustics techniques like the image source model or ray tracing algorithms. While accurate, these methods can be very computationally expensive, especially for complex 3D environments. This motivated exploring data-driven machine learning approaches like the FNO.
To efficiently obtain a large, high-quality RIR dataset for training, I utilised the PyRoomAcoustics package. This Python library provides a rapid simulation pipeline with key components. (https://arxiv.org/abs/1710.04196)

It provided an intuitive object-oriented interface for constructing scenarios with multiple sound sources and microphones in 2D/3D rooms. Additionally, it included fast C++ implementations of the image source model and ray tracing to efficiently generate RIRs and simulate wave propagation between sources and receivers. Furthermore, the library offered reference algorithm implementations for various signal processing operations like Short-Time Fourier Transforms (STFTs), beamforming, direction-of-arrival estimation, and more.

PyRoomAcoustics's combined functionality enabled simulating a diverse array of acoustical environments and sources to rapidly generate realistic RIR data at scale. Its optimised acoustic modelling routines produced reliable ground-truth solutions, while avoiding the computational bottlenecks of traditional methods during the data generation phase.
With this powerful tool, I could construct a large, parametrically-controlled dataset capturing a wide range of acoustic characteristics. This high-quality simulated data was then used to train the FNO architecture, providing the diverse examples needed for it to effectively learn the complex mapping between room specifications and accurate RIR predictions.

Development environment background research

A crucial aspect of the project's development was optimising the computational performance to ensure efficient training and inference with the model.

Given the model's complexity and the large dataset sizes involved, leveraging hardware acceleration through graphics processing units (GPUs) was essential.

GPUs excel at parallelizing the matrix and tensor operations that are at the core of deep learning models like FNOs. By offloading these computationally intensive workloads to the highly parallel architecture of GPUs, significant speedups can be achieved compared to running solely on CPUs.

To tap into this GPU acceleration, I had to ensure compatibility with the appropriate software frameworks and APIs. The two primary ecosystems for GPU-accelerated computing are CUDA, developed by NVIDIA, and ROCM (Radeon Open Compute), AMD's open-source alternative.

CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and application programming interface (API) that allows developers to harness the computational power of NVIDIA GPUs. (NVIDIA, 2021) It provides a comprehensive toolkit, including a parallel computing platform and programming model, libraries, and development tools. By leveraging CUDA, I could take advantage of NVIDIA's hardware optimization and performance tuning, enabling efficient execution of the FNO model's computations on NVIDIA GPUs.

On the other hand, ROCM (AMD, 2024) is AMD's open-source solution for GPU acceleration. It offers a similar set of tools and libraries to CUDA, but specifically designed for AMD GPUs. ROCM aims to provide a more open and portable ecosystem, allowing developers to write code that can run on various hardware architectures, including CPUs, GPUs, and other accelerators.

Research on tuning

Following the initial research and exploration phase, I turned heavily towards the creation of the artifact itself, including generating the training data, developing visualisations for that data, and implementing the Fourier Neural Operator (FNO) models. After encountering suboptimal performance with the initial model configurations, I delved deeper into research on effective hyperparameter tuning strategies and visualisation techniques to better understand and optimise the models.

Selecting the appropriate hyperparameters for a machine learning model like the FNO is crucial for achieving optimal performance. Ineffective hyperparameter values can lead to issues such as overfitting, underfitting, or inefficient convergence. To address this challenge, I researched and explored several hyperparameter tuning techniques.

Grid search involves defining a set of values for each hyperparameter and exhaustively training and evaluating models for all possible combinations. While comprehensive, it can become computationally expensive as the number of hyperparameters and their respective value ranges increase. (Tam, 2023)

Random search, on the other hand, randomly samples hyperparameter values from defined search spaces. This technique can be more efficient than grid search, especially when dealing with a large number of hyperparameters, as it explores a broader range of the search space with fewer iterations. I settled on this technique for its simplicity.

A more advanced technique is Bayesian optimization, which uses a probabilistic model to guide the search for optimal hyperparameters. (Koehrsen, 2018) It balances exploration and exploitation by selecting hyperparameter configurations that are most likely to improve the model's performance based on previous evaluations. This approach can be computationally efficient and effective, particularly for high-dimensional search spaces. With numerous experiments involving different hyperparameter configurations, it becomes crucial to log and track the results systematically.

Visualisation

Effective visualisation of the training data, model behaviour, and performance metrics played a crucial role in understanding and optimising the FNO models. I explored various visualisation techniques, including plotting room impulse response (RIR) waveforms and spectrograms to visually inspect the simulated acoustic data used for training, visualising model architectures and layer activations to gain insights into the internal representations learned by the FNO, generating visual representations of the predicted RIRs and comparing them with ground truth data to assess the model's accuracy, and plotting loss curves, accuracy metrics, and other performance indicators during training to monitor convergence and identify potential issues like overfitting or instability. These visualisation tools and techniques provided valuable insights into the model's behaviour, enabling informed decisions about architectural changes, hyperparameter adjustments, and other optimization strategies.

Additional research under extended scope

In addition to the core research on Fourier Neural Operators (FNOs) for optimising high-quality reverb generation, I explored several potential techniques that could enhance the project further, although they were not incorporated into the final artefact due to scope constraints.

I investigated butterfly matrices, which could potentially accelerate components like Fast Fourier Transforms (FFTs) used in reverb processing. (Dao, 2019) However, I was unable to implement this technique in the current project.

I also explored using natural language processing (NLP) to modify synthesiser parameters based on text input, inspired by prior work on mapping audio qualities to semantic labels. While this approach could enable more organic parameter mapping, it was not included in the final artefact.

Finally, I considered employing NLP systems to integrate user text commands with the model components, allowing for an interactive GUI driven by natural language input. Although too expansive for this project's scope, it represents an intriguing future direction to enhance usability.

While these additional research threads did not make it into the final artefact, they provided valuable perspective and context.

2. Requirements Analysis

Functional requirements

After reviewing the scope and my background research, I was able to outline these seven functional requirements.

1. Data Generation

Generating diverse and high-quality room impulse response (RIR) data is crucial for training accurate and robust machine learning models for reverb simulation. The ability to generate RIRs for different room configurations, with varying dimensions, absorption coefficients, and source positions, ensures that the model is exposed to a wide range of acoustic environments during training. This diversity helps the model generalise better to unseen scenarios, improving its performance in real-world applications. Additionally, supporting parallel data generation through techniques like multithreading or multiprocessing can significantly accelerate the computationally intensive process of RIR generation, enabling the creation of larger datasets in a more time-efficient manner. Finally, saving the generated RIRs, along with their corresponding room parameters, to files is essential for persistently storing the data, enabling reproducibility, and facilitating data management tasks like splitting the dataset into training, validation, and testing sets.

2. Data Visualization

Visualizing the generated data is a crucial step in understanding and validating its quality. Plotting the waveforms or spectrograms of the generated RIRs for each microphone can reveal potential issues or anomalies in the data, enabling corrective actions to be taken. Visualizing the room dimensions, microphone locations, and source positions can aid in verifying the correctness of the simulated acoustic environment and identifying potential correlations between room parameters and RIR characteristics. Furthermore, comparing the generated RIRs with the original, ground-truth RIRs is essential for evaluating the accuracy and performance of the machine learning models, as well as identifying areas for further improvement.

3. Audio Playback

Convolving the generated RIRs with an input audio signal and playing back the resulting convolved audio is a critical step in aurally evaluating the quality of the simulated room acoustics. While numerical comparisons and visualisations provide valuable insights, the ultimate goal is to achieve perceptually accurate and high-quality reverb simulations. By listening to the convolved audio, subjective assessments can be made regarding the realism and quality of the simulated acoustics, potentially revealing nuances or artefacts that may be difficult to detect through numerical or visual analysis alone.

4. Model Training

Training machine learning models, such as Recurrent Neural Networks (RNNs) or Fourier Neural Operators (FNOs), to predict RIRs based on input room parameters is the core objective of the project. Supporting different model architectures and configurations allows for exploring and comparing different approaches, enabling the selection of the most effective and efficient solution. Performing k-fold cross-validation during training helps in evaluating the generalisation performance of the models and mitigating overfitting. Visualising the training loss over epochs is a crucial step in monitoring the model's convergence and identifying potential issues, such as divergence or instability.

5. Model Evaluation

Evaluating the trained models is essential for assessing their performance and determining their suitability for real-world applications. Loading trained models from files enables persistent storage and reuse of models, avoiding the need to retrain from scratch for every evaluation. Generating RIRs using the trained models and comparing them with the original RIRs provides a quantitative measure of the model's accuracy and performance. Additionally, hearing the generated RIRs by convolving them with an input audio signal allows for subjective evaluation of the simulated room acoustics, complementing the numerical comparisons with perceptual assessments.

6. User Interface

Providing a menu-based command-line interface for interacting with the different functionalities of the program enables users to easily navigate and access the various components, such as data generation, model training, and evaluation. Allowing users to select specific room configurations, models, and input audio files further enhances the usability and flexibility of the system, catering to diverse use cases and experimentation scenarios.

7. Logging and Error Handling

Logging relevant information, warnings, and errors during program execution is essential for debugging, troubleshooting, and understanding the program's behaviour. Clear and informative logs can aid in identifying and resolving issues, as well as providing a record of the program's execution for future reference or analysis. Proper error handling ensures that the program gracefully handles exceptions and errors, preventing crashes or unexpected behaviour, and providing meaningful feedback to the user.

By carefully considering and justifying each requirement, I ensured that my system was designed to meet the project's objectives effectively, while also accounting for important factors such as performance, usability, and robustness. This level of detail and reasoning demonstrates a structured approach to requirement gathering and serves as a solid foundation for the subsequent development and implementation phases.

Non functional requirements

I also outlined five non functional requirements.

1. ### Performance

   Optimising the code for efficient data generation and model training is crucial in this project due to the computationally intensive nature of simulating room acoustics and training machine learning models. Generating room impulse responses (RIRs) for a large dataset with varying room configurations can be a resource-intensive process, especially when dealing with complex room geometries and high-fidelity simulations. Similarly, training machine learning models like Fourier Neural Operators (FNOs) or Recurrent Neural Networks (RNNs) on large datasets can be computationally expensive, particularly during the iterative training process. Poor performance in these areas can lead to prohibitively long training times, hindering the ability to explore different model architectures and configurations effectively. Therefore, optimising the code's performance is crucial for enabling efficient experimentation and ensuring timely progress in the project.

2. ### Scalability

   As the project progresses, the amount of data and the complexity of the room configurations and models may increase. Ensuring that the system is scalable is essential for accommodating these growing demands without compromising performance or functionality. A scalable system can handle larger datasets and more intricate room configurations without significant degradation in performance or requiring major architectural changes. Additionally, the ability to support more complex machine learning models, such as deeper or wider neural network architectures, can be crucial for achieving higher accuracy and better generalisation performance.

3. ### Usability

   A user-friendly and intuitive interface is essential for enabling seamless interaction with the different functionalities of the system. Clear instructions and meaningful error messages can greatly improve the user experience and reduce frustration, especially for non-technical users or those unfamiliar with the system. A well-designed interface can also facilitate efficient navigation and operation, allowing users to focus on their tasks without unnecessary distractions or complexities.

4. ### Portability

   Developing a platform-independent codebase is important for ensuring that the system can be deployed and utilised on a wide range of operating systems and hardware configurations. This portability can be particularly valuable in collaborative projects or when sharing the system with others, as it eliminates the need for users to have specific hardware or software environments. Additionally, a portable system can facilitate easier integration with other tools or workflows, expanding its potential applications and increasing its overall utility.

5. Maintainability

Well-structured, modular, and documented code is essential for ensuring the long-term maintainability of the system. As the project evolves and new features or enhancements are required, maintainable code can significantly simplify the process of making changes, fixing bugs, or adding new functionality.

Modular design promotes code reuse and improves code organisation, while clear documentation helps future developers understand the system's architecture, design decisions, and implementation details. Maintainable code can also facilitate easier collaboration and knowledge transfer within development teams, reducing the overhead associated with onboarding new team members or transitioning the project to a different group.

3. Software Design

The Cowboy methodology's emphasis on iterative development and frequent delivery has been instrumental in driving my project's progress. By breaking down the overall goals into smaller, manageable iterations, I have been able to define a more focused scope and concentrate on early testing, evaluation, and course correction. This iterative approach has facilitated an efficient development process while enabling me to gather valuable feedback and insights at each stage, further refining the solution to ensure its alignment with the project objectives.

Design of the Data Generation

The primary goal in designing this component is to generate a diverse dataset of room impulse responses (RIRs) for various room configurations using the pyroomacoustics library.

The component should be designed to generate the dataset at a variety of smaller grid resolutions, starting with an initial grid resolution and a specified number of configurations to generate. It should attempt to leverage parallel processing for faster data generation.

For each individual room configuration, the design should incorporate adding microphones to the room at calculated grid points and setting up an impulse signal. The room acoustics will then be simulated using the simulate method from pyroomacoustics, which calculates the RIRs for each microphone position.

Once all configurations have been created for a specific resolution, the design should include functionality to save them to disk according to a predefined file structure.

Design of Visualization

Visualising the room itself should be a crucial aspect of the program design. This functionality should be designed to load the room dimensions, microphone locations, and source location from the configuration files. It should then create a 3D plot of the room, displaying the wireframe of the room boundaries, the microphone positions, and the source position. This visual representation will allow users to verify the correctness of the simulated room geometry and the placement of microphones and sound sources, ensuring that the configurations align with the intended specifications.

Additionally, it will be useful to design functionality for visualising the entire wavefield. This function should be designed to load the RIR data for each microphone position and plot them in a grid of subplots, with each subplot representing the signal captured by a specific microphone over time. By visualising the waveforms or spectrograms of the RIRs, users will be able to inspect the characteristics of the simulated acoustic waves, such as amplitude and frequency content, at different microphone positions. This visualisation can aid in identifying potential issues or anomalies in the data and can provide valuable insights into the simulated acoustic environment.

Designing the ability to hear audio convolved with microphone RIR is a crucial feature that will enable users to aurally evaluate the simulated room acoustics. This functionality should be designed to allow users to select a specific microphone position and convolve a given audio file with the corresponding RIR. The resulting convolved audio will then be played back to the user, simulating the acoustic effect of the room on the input audio signal. By listening to the convolved audio, users will be able to subjectively assess the realism and quality of the simulated room acoustics, complementing the numerical and visual analyses.

Design of Model

In terms of design, it’s important the model architecture stays relatively simple as the goal of this project is to demonstrate the capabilities, not roll out a production implementation. I do not plan to adapt the layers of the FNO at all and plan to use existing implementations.

In implementing the model, one of the crucial aspects I aim to incorporate is Tucker factorization (Balažević et al., 2019). Tucker factorization is a tensor decomposition method that enables the efficient representation and computation of high-dimensional data. By applying Tucker factorization to the FNO model, I intend to significantly reduce the number of parameters and computational complexity while maintaining the expressive power of the model. This factorization technique will enable the FNO model to capture the complex relationships between the input parameters and the generated RIRs in a more compact and efficient manner.

Furthermore, I plan to employ batch normalisation layers in the model architecture. Batch normalisation is a technique that normalises the activations of each layer, reducing the internal covariate shift and improving the stability and convergence of the training process. By incorporating batch normalisation, I aim to enhance the model's ability to learn and generalise from the training data, leading to more accurate and reliable RIR generation. (Ioffe & Szegedy, 2015)

To optimize the training process and prevent overfitting, I intend to implement gradient clipping and learning rate scheduling. (Zhang et al., 2020) Gradient clipping will ensure that the gradients remain within a specified range, mitigating the risk of exploding gradients and promoting stable training. Learning rate scheduling will allow for the dynamic adjustment of the learning rate during training, enabling the model to converge more effectively and avoid getting stuck in suboptimal solutions.

Design of Tuning

In designing the tuning component, my goal is to create a flexible and efficient approach. Instead of exhaustively searching through all possible combinations of hyperparameters (grid search), I plan to implement a random search approach. This approach will randomly sample hyperparameter values from a defined search space, which can be more efficient than grid search, especially when dealing with a large number of hyperparameters.

***

By following the Cowboy software methodology and incorporating adaptive and flexible designs, I aim to address potential challenges and constraints effectively while maintaining a focus on the core objectives of the project.

4. Implementation

Data generation

The data generation process aims to create a diverse dataset of room impulse responses (RIRs) for various room configurations, which will be used for training and evaluating the machine learning models.

The `create_config` function is the core component responsible for generating the RIRs for a single room configuration. It takes several parameters, including the grid resolution, configuration ID, room dimensions, target reverberation time (RT60), and source position.

Create config process

Room Creation

The function creates a room object using the `pra.ShoeBox` class from the *pyroomacoustics* library. The room dimensions, absorption coefficients, and maximum simulation order are specified during this step.

Microphone Placement

A grid of microphone locations is calculated based on the provided grid resolution and room dimensions. These microphone positions are then added to the room object.

Source Signal

An impulse signal is created, which will be used as the source signal for the simulation.

Simulation

The room acoustics are simulated using the `room.simulate()` method, which calculates the RIRs for each microphone position.

Data Saving

The generated RIRs, room dimensions, microphone locations, source position, and other relevant parameters are saved to disk in a specific directory structure. This allows for persistent storage and ease of data management.

Further process

The `generate_dataset` function orchestrates the data generation process by calling `create_config` multiple times with different configurations. It supports parallel processing using the `concurrent.futures` module, which can significantly speed up the data generation process by utilising multiple CPU cores. However, if a `MemoryError` is encountered during parallel processing, the function handles the error by reducing the number of worker processes or switching to sequential processing.

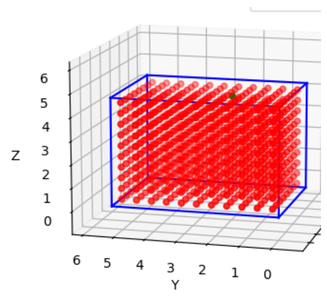
The code also includes logging functionality to track the progress of the data generation process and record any errors or warnings that may occur.

Overall, the data generation component demonstrates a structured and flexible approach to creating a diverse dataset of RIRs, which is essential for training accurate and robust machine learning models for reverb simulation.

Visualisation functions

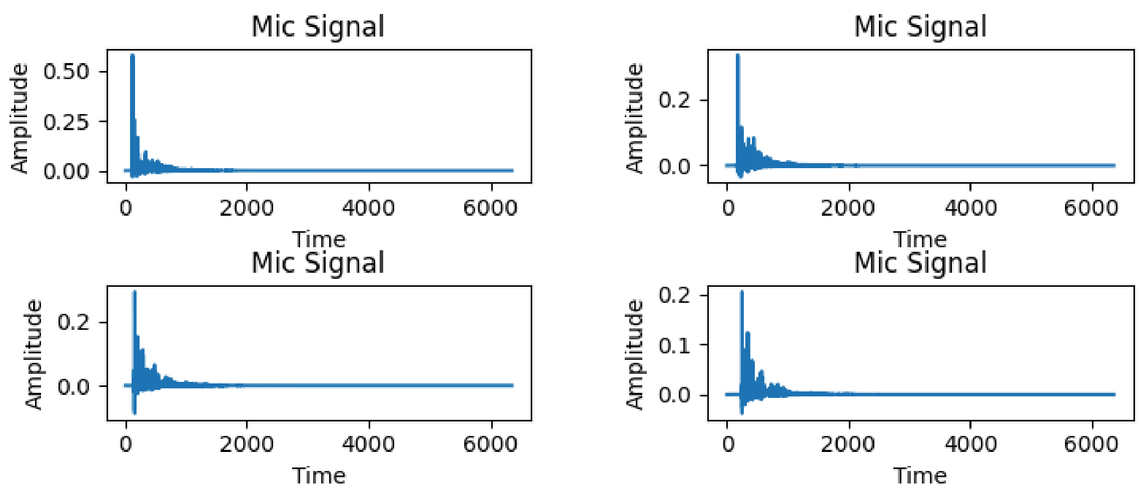
`Visualize_wavefield_by_source_proximity` this function provides a unique 3D visualisation of the microphone signals, where the z-axis represents the proximity of the microphone to the sound source. This visualisation allows me to observe patterns and relationships between the signals and their distance from the source, which can be valuable for evaluating the accuracy and realism of the simulated room acoustics.

`visualize_room`: This function creates a 3D plot of the simulated room itself, displaying the room boundaries as a wireframe, the microphone positions as red dots, and the source position as a green dot. As the developer, this visualisation helps me verify the correctness of the simulated room geometry and the placement of microphones and sound sources, ensuring that the configurations align with the intended specifications.



Visualising at 0.5 resolution

`visualize_wavefield`: This function provides a grid of subplots, where each subplot represents the waveform or spectrogram of the microphone signal at a specific microphone position. This visualisation allows me to inspect the characteristics of the simulated acoustic waves, such as amplitude and frequency content, at different microphone positions within the room. By examining these plots, I can identify potential issues or anomalies in the data and gain insights into the simulated acoustic environment.



`hear_wavefield`: This function enables me to aurally evaluate the simulated room acoustics by convolving a given audio file with the room impulse response (RIR) of a specific microphone position. The resulting convolved audio is then played back, simulating the acoustic effect of the room on the input audio signal. By listening to the convolved audio, I can subjectively assess the realism and quality of the simulated room acoustics, complementing the numerical and visual analyses.

`hear_reference_audio`: The `hear_reference_audio` functionality allows me to listen to the original audio file without any convolution. This feature serves as a reference point for comparing the convolved audio and evaluating the impact of the simulated room acoustics on the input signal. By providing both the convolved and reference audio, I can better appreciate the effects of the simulated acoustic environment and make informed judgments about the quality of the simulation.

Model implementation

The implementation of the machine learning models is a crucial component of my project, as it aims to train models capable of accurately predicting room impulse responses (RIRs) based on input room parameters. I started with a Recurrent Neural Network (RNN) model to test the feasibility of the approach and then proceeded to implement a Fourier Neural Operator (FNO) model, which is the main focus of the current artefact.

RNN Model Implementation

I first implemented an RNN model to serve as a baseline and to test the viability of using neural networks for this task. The `RIRModel` class defines the RNN architecture, which consists of an RNN layer followed by a fully connected layer. The `__init__` method initialises the model with the specified input size, hidden size, number of layers, and output size. The `forward` method defines the forward pass of the model, where the input is passed through the RNN layer, and the resulting output is then passed through the fully connected layer to produce the final output.

FNO Model Implementation:

Building upon the initial success with the RNN model, I implemented the FNO model using the library from (Li et al., 2021). This is by the same authors as the FNO paper.

The `FNOModel` class defines the FNO architecture, which is an example of the Fourier Neural Operator (FNO) introduced by Li et al. (2021). The `__init__` method initialises the model with the specified number of modes, hidden channels, input channels, output channels, factorization method, and rank. The `forward` method defines the forward pass of the model, where the input is first reshaped to match the expected input format for the FNO, then passed through the FNO layer, and finally processed by a fully connected layer to produce the final output.

Dataset and Data Loading

To train and evaluate the models, I implemented data loading and preprocessing functionality. The `RIRDataset` class inherits from `torch.utils.data.Dataset` and is responsible for loading and preprocessing the RIR data. The `__init__` method loads the configuration data, calculates the maximum RIR length, and pads or truncates the RIRs to ensure a consistent length. It also normalises the RIR data by subtracting the mean and dividing by the standard deviation.

Training and Evaluation

The `train_model` function is responsible for training the model using the provided training data loader, loss criterion, optimizer, and learning rate scheduler. It performs the training loop, computing the loss, backpropagating the gradients, and updating the model parameters. The function also implements early stopping by tracking the validation loss and saving the best model weights.

The `compare_rirs` function is used to evaluate the trained model by generating RIRs using the model's predictions and comparing them with the original RIRs. It allows for auditory evaluation by convolving the generated RIRs with an input audio signal and playing back the convolved audio. Additionally, it visualises the original and generated RIRs, as well as the room configuration, for visual inspection.

To persist and reuse trained models, I implemented functionality for loading and saving models. The `load_model` function prompts the user for a model file path and loads the corresponding model checkpoint. It supports loading both RNN and FNO models by checking the model type in the checkpoint and creating the appropriate model instance.

The `generate_model` function is responsible for training a new model from scratch. It performs k-fold cross-validation by splitting the dataset into training and test sets for each fold. The function trains the model on the training set and evaluates its performance on the test set. After training, it prompts the user for a model name and saves the trained model as a checkpoint file.

To facilitate interaction with the different functionalities of the program, I implemented a menu-based command-line interface. The `menu` function presents a list of options to the user, such as loading a model, generating a new model, or comparing RIRs using the loaded model. Based on the user's choice, the corresponding function is executed, allowing for seamless navigation and experimentation.

In the current artefact, the focus is primarily on the initial readings and performance of the FNO model. While the RNN model was implemented first to test the feasibility of the approach, the FNO model is expected to provide better performance and accuracy due to its ability to leverage the spatial and spectral inductive biases inherent in the problem domain.

By implementing both the RNN and FNO models, I have established a solid base for evaluating and comparing their performance in predicting RIRs. The modular design of the codebase, along with the data loading and preprocessing components, facilitates further experimentation and refinement of the models. Additionally, the user interface and model loading/saving functionality enable seamless interaction with the system and promote reproducibility and reusability of trained models.

Tuning implementation

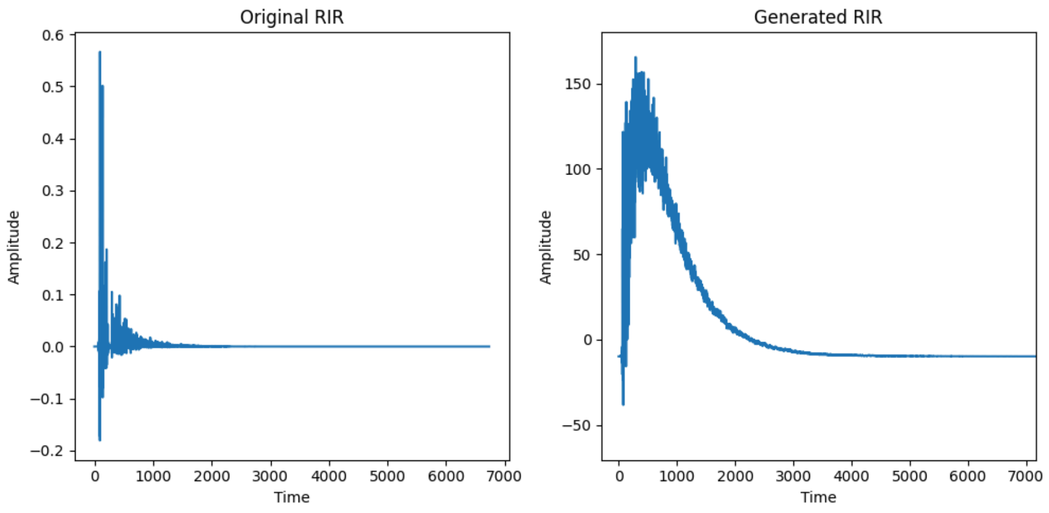
The approach randomly samples hyperparameter values from predefined search spaces, which can be more efficient than grid search, especially when dealing with a large number of hyperparameters.

The `random_search` function plays the central role in the tuning process. It takes the model type, resolution, and the number of search iterations as input parameters. Within each iteration, it randomly selects hyperparameter values from specified ranges based on the model type (RNN or FNO). For the FNO model, the search spaces include the number of modes, hidden channels, rank, learning rate, and batch size. The function constructs the model using the sampled hyperparameters and trains it using the `train_model` function.

After each iteration, the `random_search` function compares the validation loss of the trained model with the best loss obtained so far. If the current model achieves a lower validation loss, it is considered the best model, and its weights are saved. This process is repeated for the specified number of iterations, allowing for the exploration of different hyperparameter configurations.

To facilitate the analysis and understanding of the tuning process, the display_model_info function was also implemented. It displays the model info simply. It is notable that the `random_search` does not save the models it creates.

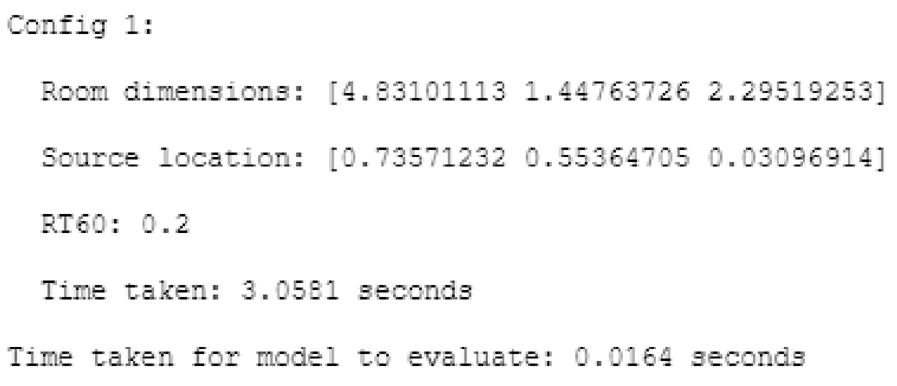
5. Evaluation of Results



Initial synthesis using the RNN

Deployment

From the outset, the evaluation of the Fourier Neural Operator (FNO) model proved to be much faster than traditional room impulse response (RIR) generation methods. This initial finding was promising, as it aligned with the project's goal of optimizing the dynamic generation of high-quality reverb for improved performance on lower-end hardware.



```
Config 1:
  Room dimensions: [4.83101113 1.44763726 2.29519253]
  Source location: [0.73571232 0.55364705 0.03096914]
  RT60: 0.2
  Time taken: 3.0581 seconds
Time taken for model to evaluate: 0.0164 seconds
```

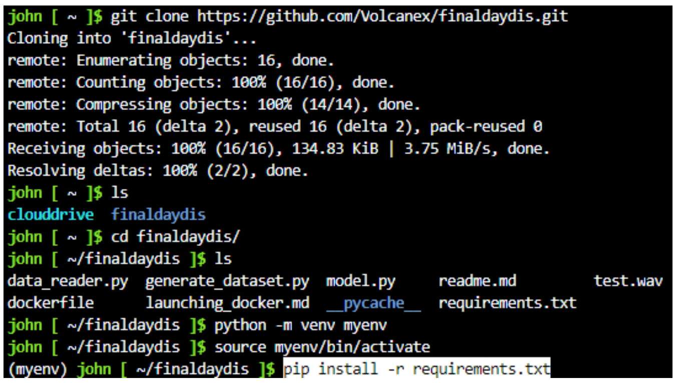
The model usually evaluated at least an order of magnitude faster than the hard coded methods

However, the evaluation process itself presented significant challenges. Following the submission of the project artefact and the implementation of the FNO, efforts were focused on accelerating the model training process using available resources. Since CUDA, the parallel computing platform developed by NVIDIA, requires an NVIDIA GPU, and my available hardware was an AMD Radeon RX 5700XT, the focus shifted towards exploring ROCm, AMD's open-source alternative.

Attempts were made to mount the model code on a Docker container, but this approach proved unsuccessful. Efforts then turned to utilising cloud computing resources, with Azure Cloud machines being explored. Unfortunately, this endeavour was hindered by quota restrictions, preventing the successful deployment of the necessary compute resources.

Eventually, CUDA compatibility was achieved by leveraging a Google Cloud virtual machine and a dedicated virtual environment. However, even with this setup, persistent errors emerged that could not be resolved, leading to a decision to abandon GPU acceleration and instead rely on high-performance CPU models for model training and evaluation.

The difficulties encountered in obtaining the necessary computational resources and the associated time constraints meant that the additional functions designed to automate the tuning process became less relevant. As a result, efforts were redirected towards manually optimising the model's performance, rather than randomly, through careful hyperparameter tuning and architectural adjustments.



```
john [ ~ ]$ git clone https://github.com/Volcanex/finaldaydis.git
Cloning into 'finaldaydis'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 16 (delta 2), reused 16 (delta 2), pack-reused 0
Receiving objects: 100% (16/16), 134.83 KiB | 3.75 MiB/s, done.
Resolving deltas: 100% (2/2), done.
john [ ~ ]$ ls
clouddrive  finaldaydis
john [ ~ ]$ cd finaldaydis/
john [ ~/finaldaydis ]$ ls
data_reader.py  generate_dataset.py  model.py     readme.md        test.wav
dockerfile      launching_docker.md  __pycache__  requirements.txt
john [ ~/finaldaydis ]$ python -m venv myenv
john [ ~/finaldaydis ]$ source myenv/bin/activate
(myenv) john [ ~/finaldaydis ]$ pip install -r requirements.txt
```

Running the virtual machine

Limited convergence

Partially because it was so resource intensive to run large searches and implement hyperparameter optimisation, I was not able to get the FNO model to meaningfully converge despite extensive efforts and experimentation. Each training session took a significant amount of time, due to the complexity of the model and the size of the training data. I primarily focused on using the 2.0 resolution dataset for training, as it provided a good balance between computational feasibility and capturing the essential characteristics of the room impulse responses (RIRs).

During the training process, I closely monitored the mean squared error (MSE) as the primary metric to assess the model's performance. However, despite trying various optimization techniques and hyperparameter adjustments, at best, the MSE often hovered around 6500, indicating a significant discrepancy between the generated RIRs and the ground truth. This high MSE value suggested that the model struggled to accurately capture the complex relationships between the input parameters and the corresponding RIRs.

Moving forward, additional investigations into alternative architectures, loss functions, and training strategies may help improve the FNO model's convergence and generation quality. Exploring techniques such as transfer learning, data augmentation, or incorporating domain-specific knowledge could potentially bridge the gap between the model's efficiency and the desired level of accuracy.

Particular findings

While optimising the model, I had avoided using the finer resolutions of training data. I did this due to the lack of computational resources and the fact that using the higher resolution data took longer when I first started to optimise the model.

Best run

Near the end of my attempts to reduce the MSE, I tried the higher quality and the loss of most models dropped to 1505.

```
 Modes: (32, 16)
 Width: 32
 Rank: 0.1
 Learning Rate: 0.000252
 Batch Size: 32

Epoch [1/10], Val Loss: 1795
Epoch [2/10], Val Loss: 1772
Epoch [3/10], Val Loss: 1730
Epoch [4/10], Val Loss: 1655
Epoch [5/10], Val Loss: 1538
Epoch [6/10], Val Loss: 1521
Epoch [7/10], Val Loss: 1505
Epoch [8/10], Val Loss: 1491
Epoch [9/10], Val Loss: 1476
Epoch [10/10], Val Loss: 1462
```

For reference, running this model took around ~15 mins.

Larger models, more simulations, finer data

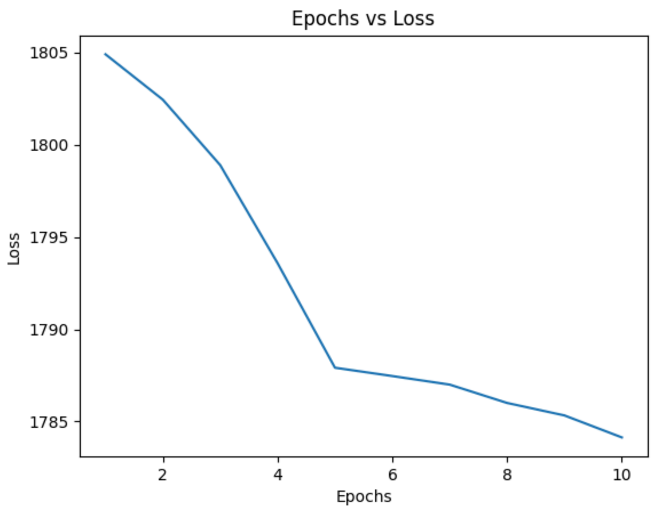
I believe that the current program, though flawed may be able to demonstrate meaningful convergence and meet the original goals of the program if we were able to run the simulations at a larger scale.

Increasing Model Capacity

The Fourier Neural Operator (FNO) architecture employed in this project has the capacity to capture the intricate relationships between input room parameters and the corresponding room impulse responses (RIRs). However, the model sizes explored thus far may have been constrained by computational limitations. By leveraging more powerful hardware accelerators such as GPUs or distributed computing resources, it would be possible to train significantly larger FNO models with increased depth and width. These larger models could potentially better represent the complex spatial and spectral characteristics present in the RIR data.

More Training Iterations

The checkpointing functionality implemented in the codebase allows for training to be resumed from previously saved model states. With access to greater computational resources, it would be feasible to train the FNO models for a substantially higher number of epochs. Longer training times and more epochs ran through could enable the model to more thoroughly explore the parameter space and converge to a better optimum, resulting in improved accuracy in RIR prediction.



Loss over epochs

Expanded Hyperparameter Search

The random search approach for hyperparameter tuning currently explores a limited range of values due to resource constraints. By expanding the search ranges and increasing the number of random search iterations, a more comprehensive exploration of the hyperparameter space could be conducted. This broader search could uncover configurations that yield superior performance, further enhancing the model's capabilities.

Higher Resolution Training Data

While the 2.0 and 1.0 resolution dataset provided a good balance between computational feasibility and capturing essential RIR characteristics, utilising even higher resolution data could potentially lead to improved model performance. The finer-grained details present in higher resolution datasets may enable the FNO model to better capture the nuances and subtleties of the RIRs, resulting in more accurate predictions.

Conclusion of results

By leveraging larger model architectures, conducting more extensive simulations with increased training iterations, expanding the hyperparameter search space, and incorporating higher resolution training data, the project's goals of dynamically generating high-quality reverb for improved performance on lower-end hardware could potentially be achieved. These enhancements would enable the FNO model to more effectively learn the intricate mappings between room parameters and RIRs, ultimately leading to better convergence and higher quality RIR predictions.

Conclusion

In conclusion, this project aimed to optimize the dynamic generation of high-quality reverb using Fourier Neural Operators (FNOs) for efficient performance on lower-end hardware. The journey began with a focus on improving resource-intensive plugins using novel machine learning techniques, but as the project progressed, the scope narrowed to specifically address the computational bottleneck associated with high-fidelity reverb generation.

The development process followed the agile Cowboy software methodology, which allowed for iterative development, early testing, and course correction based on insights gathered at each stage. This approach proved invaluable in navigating the challenges and complexities encountered throughout the project.

The core components of the project included data generation, visualisation, and the implementation of FNO models. The data generation process utilised the PyRoomAcoustics library to simulate diverse acoustic environments and generate realistic room impulse responses (RIRs) at scale. Visualisation techniques were employed to inspect the generated data, assess model performance, and gain insights into the simulated acoustics.

The implementation of the FNO models involved extensive research and experimentation. Various optimization techniques, such as Tucker factorization and batch normalisation, were incorporated to enhance the models' efficiency and generalisation capabilities. Hyperparameter tuning and architectural adjustments were made to improve performance and convergence.

Despite the efforts to optimise the FNO models, the evaluation process presented significant challenges. The pursuit of GPU acceleration through CUDA and ROCm proved difficult due to hardware compatibility issues and resource constraints. Ultimately, the models were trained using high-performance CPUs, but the training process remained time-consuming due to the complexity of the models and the size of the training data.

The evaluation results revealed that the FNO models struggled to converge meaningfully, with the mean squared error (MSE) often remaining high despite various optimization attempts. While the models demonstrated faster evaluation times compared to traditional methods, the accuracy of the generated RIRs fell short of the desired level.

Despite the challenges faced, this project has laid the groundwork for further research and development in the field of efficient reverb generation using machine learning techniques. The insights gained from this endeavour will guide future iterations and improvements.

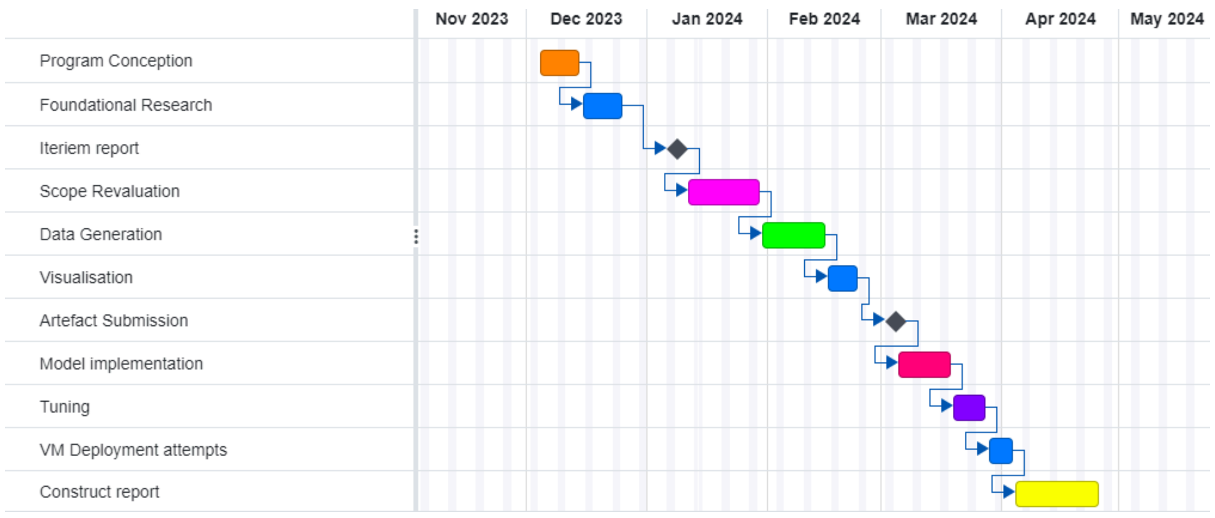
Moving forward, exploring alternative architectures, loss functions, and training strategies may help improve the FNO models' convergence and generation quality. Techniques such as transfer learning, data augmentation, and incorporating domain-specific knowledge could potentially bridge the gap between efficiency and accuracy.

Furthermore, the project's modular design and the development of key components, such as data generation and visualisation, provide a solid foundation for future work. These components can be leveraged and extended to support ongoing research and experimentation in the field of audio processing and machine learning.

While the project faced challenges in achieving the desired level of accuracy and convergence, it has made significant strides in exploring the potential of FNOs for efficient reverb generation. The development environment created provides a solid base for further research into this topic which I would like to continue after this paper, hopefully with more resources.

Appendices

Timeline



This is a rough timeline of how I completed the program.

Community consultation

What new opportunities do you think machine learning could bring to plugin development, especially for Ableton?

Running the program

The code is hosted here alongside my submitted artefact.
https://github.com/Volcanex/finaldaydis

You can find instructions in the *readmd.md* in the program directory.

References

Feynman (2006) *The feynman lectures on physics vol. I ch. 47: Sound. the wave equation*. Available at: https://www.feynmanlectures.caltech.edu/I_47.html (Accessed: 07 May 2024).

AMD (2024) *amd-rocm-6-brief* . Available at: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/product-briefs/amd-rocm-6-brief.pdf (Accessed: 07 May 2024).

Balažević, I., Allen, C. and Hospedales, T.M. (2019) *Tucker: Tensor factorization for knowledge graph completion*, *arXiv.org*. Available at: https://arxiv.org/abs/1901.09590 (Accessed: 07 May 2024).

Dao, T. (2019) *Butterflies are all you need: A universal building block for structured linear maps*, *Butterflies Are All You Need: A Universal Building Block for Structured Linear Maps · Stanford DAWN*. Available at: https://dawn.cs.stanford.edu/2019/06/13/butterfly/ (Accessed: 07 May 2024).

Hollar, A.B. (2006) *An agile programming methodology for a solo programmer*. Available at: https://core.ac.uk/download/pdf/51292464.pdf (Accessed: 07 May 2024).

Ioffe, S. and Szegedy, C. (2015) *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, *arXiv.org*. Available at: https://arxiv.org/abs/1502.03167 (Accessed: 07 May 2024).

Jiang, H., Wen, K. and Chen, Y. (2023) *Practically solving LPN in high noise regimes faster using neural networks*, *IACR Cryptology ePrint Archive*. Available at: https://eprint.iacr.org/2023/372 (Accessed: 07 May 2024).

Koehrsen, W. (2018) *A conceptual explanation of Bayesian hyperparameter optimization for Machine Learning*, *Medium*. Available at: https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (Accessed: 07 May 2024).

Li, Z. *et al.* (2021) *Fourier neural operator for parametric partial differential equations*, *arXiv.org*. Available at: https://arxiv.org/abs/2010.08895 (Accessed: 07 May 2024).

Tam, A. (2023) *How to grid search hyperparameters for pytorch models*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/how-to-grid-search-hyperparameters-for-pytorch-models/ (Accessed: 07 May 2024).

Zhang, J. *et al.* (2020) *Why gradient clipping accelerates training: A theoretical justification for adaptivity*, *arXiv.org*. Available at: https://arxiv.org/abs/1905.11881 (Accessed: 07 May 2024).

Bibliography

Additional inspiration

https://sohl-dickstein.github.io/2024/02/12/fractal.html
https://www.youtube.com/watch?v=lLUcOFwZvyY&ab_channel=TomScott

Special thanks to the talented authors of the Fourier Neural Operator paper and the authors of the room acoustics library to which without my project wouldn't exist and my supervisor Dr Maitreyee Dey for keeping me on track.