Machine Learning models for audio generation with embedded devices

- Affording control of latent parameters for real time audio synthesis -

Master's Thesis

Hannah Jensen Tyedmers

Aalborg University
Electronics and IT

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Machine Learning models for audio generation with embedded devices

Theme:
Audio synthesis, Machine Learning, Embedded devices

Project Period:
Fall Semester 2023

Participant(s):
Hannah Jensen Tyedmers

Supervisor(s):
Daniel Overholt

Copies: 1

Page Numbers: 46

Date of Completion:
December 20, 2023

Abstract:

Generative machine learning models for audio synthesis provides several interesting approaches to shaping of the sound in ways that are usually difficult to achieve with other forms of synthesis. Assembling of musical instruments capable of merging between widely different directions in sound can be made possible when considering deep learning models. However, such an endeavor is not immediately apparent, and a long line of work has been behind getting deep learning models to efficiently work in real time scenarios for audio and to be deployed for smaller devices. A question that could be posed in this context is how to afford intuitive control over the learned representations of generative models for creation and performing of music, which this project has sought to address.

Contents

Chapter 1

Introduction

Machine learning models and especially deep neural networks have become increasingly more commonplace within the last decade. A particularly useful feature of neural networks is their ability to find correlations across vast amounts of data. The advancement of deep neural networks means they have become more proficient for a broader variety of tasks of a high complexity, such as speech recognition, image classification, and, as of more recently, synthesizing new instances of data [14].

Audio synthesis with deep neural networks enables interesting possibilities, such as timbre transfer, where a recording of a voice or an instrument the model has never encountered before can be made to sound like it was played by an instrument known by the network, such as the recording of a singing voice merging into a violin [11]. Sounds can also be synthesized entirely by the model itself, by drawing samples from the latent representation the model has learned from observing a large dataset of different sounds. Yet, doing so has been proven to usually either be costly in terms of quality of the audio or being computationally demanding, though recent progress has made it possible to find some middle ground [4].

This project has been concerned with considering how this might be leveraged for real time use in the context of creating music and for the construction of instruments that might be considered fitting for interacting with the latent spaces of machine learning models. Additionally, smaller, low-powered devices have been employed for the task of inference to the models as well as for mapping the gestures of a user for them to gain control of the sound and its parameters in alternative ways.

Chapter 2

Background

In this chapter, overall details are given about generative models, specifically differentiable generative networks, that have achieved promising results for audio synthesis. Some examples of models used for audio are presented, as well as how they contributed to the state of the art that are compatible with embedded devices.

2.1 Generative models

A desirable feature of machine learning models is their ability to learn and synthesize features based on a large set of examples they've been presented to. A model can hereby learn to produce output in a structured manner that still has some variation from sample to sample; and so, is without a single one "correct" result from a given input [14]. In other words, the objective of training the model is to learn a probability distribution $q(x)$ — which describes possible parameters to create new data from — by repeatedly introducing a random sample x from a real data distribution $p(x)$.

An early approach to learning a probability distribution was to attempt to make generative models learn the density of the distribution from a "family" of densities. The one that is most likely to be associated with the data (i.e. maximizes the probability) could theoretically be found by minimizing the Kullback-Lieber divergence[1] $D_{KL}(p(x)|q(x))$ with respect to the density of the real distribution. [1]. However, in most cases the intersection between the two distributions will be so negligible that a divergence cannot be found. For this reason, models found in classical machine learning literature will have a noise term (in most cases Gaussian noise) added to the distribution of the model. This makes the approach of using maximum likelyhood in order to make the models converge possible, but the noise term also degrades the quality of the generated samples [1].

[1]The divergence describes the distance between the distributions, and so measures how likely they are to be similar.

2.1.1 Differentiable generative networks

Many recent generative networks can be thought of as using part of it as a differentiable function that can transform a latent variable, z, into an output sample, x. A model that can by itself be referred to as the generator is often paired with another network in order to learn how to transform, or decode, the latent variables. This is also known as a feedforward network, which represents a parametric family of non-linear functions (or conditional distributions) that can be learned through gradient descend (such as minimizing distances like D_{KL}), and essentially, for the model to learn to arrange the latent space so that it may easily map from z to x [14].

Variational auto-encoders

Variational auto-encoders (VAEs) seek to find the approximate likelihood of the real distribution $p(x|z)$ and consists of two combined smaller models for encoding and decoding latent parameters. Usually, a Gaussian or Bernoulli distribution [2] is used as a prior for the encoder, which, through training, becomes a variational approximation $q(x|z)$ of $p(x|z)$ [30]. In order to generate a data sample, x (e.g. image, audio sample), the encoder draws a parameter z from the latent space of $q(x|z)$, which is passed to the decoder that makes up the actual differentiable function for transforming z to x [14].

Each data point x from $p(x|z)$ will have some marginal likelihood that can be summed together to find the marginal likelihood of the entire data distribution. This is called the variational lower bound, or evidence lower bound (ELBO), which evaluates the data points with the following loss function, $\mathcal{L}$:

$$\mathcal{L}_{\phi,\theta}(x) = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathcal{D}_{KL}[q_\phi(z|x)||p(z)] \tag{2.1}$$

VAEs, like the classic examples of generative models, relies on gradient descend to approximate the maximum likelihood. The important difference is that because VAEs was made to allow for some deviation from the real distribution, they can approximate a maximum likelihood that would otherwise be intractable [19]. However, there is a likely correlation between the maximum likelihood approach and noisy output samples (e.g. such as seen in images generated by VAEs, which tend to be blurry). This is possibly because this method (along with modelling the distribution as a Gaussian function) causes the model to assign high probability across many different points that are unlikely to all be included in the real distribution [14].

[2] These two distributions often used as they are applicable in many different cases.

Generative adversarial networks

Goodfellow et al. introduced the Generative adversarial network (GAN) structure in 2014, which was highly proficient at image generation, among several potential usages. GANs have since proved to give promising results for tasks related to audio synthesis, examples of which will be explained further in section 2.2.1.

Many variants of GANs, including the original proposed by Goodfellow et al. [15], does not include an encoder, as it was introduced to mitigate the difficulties with estimating probability through the maximum likelihood approach [29]. Instead, the principle of a GAN is to have two networks that compete to respectively maximize and minimize some objective function. This is also referred to as a minimax-game, where the objective of both networks is to improve their performance over time. The network that seeks to maximize the term is the generator, while the other network, the discriminator, minimizes the function in order to evaluate how similar the output of the generator is to the data from the real distribution. The latent space of the generator is initialized as a set of random noise variables, that will start to represent (or recover) the distribution of the dataset [15].

While GANs have been noted to produce output of higher quality than VAEs, they are also prone to be more unstable and difficult to get to converge to the training data. A common problem encountered is "mode collapse", where the GAN model is only able to capture a subset of the variation of the real data distribution [29]. However, proposed variants of the GAN structure, such as by sampling the latent space using auto-encoders and fine-tuning them through adversarial training have been shown to be beneficial [10]. Another consideration is finding an appropriate method for measuring the distance — D_{KL} and the similar Jensen-Shannon distance, D_{JS}, provides strong distances, but might also cause the gradients (i.e. parameters) of the generator to "vanish", and no longer be useful in describing the data distribution of the generator [29].

2.2 Models for audio generation

In order to accurately generate audio waveforms, one has to account for the fact that audio generally requires very high temporal resolution. Modelling of raw waveforms was first achieved with autoregression (i.e. by modelling a time series — in this case, a complex waveform — by predicting the next step with a regression equation based on the previous sample). These models have mostly received attention for their capability of text-to-speech generation, in particular the WaveNet model (2016) introduced by Oord et al. [25]. In order to accurately capture the broader temporal structures of waveforms, different measures have to be taken alongside autoregression for these models: WaveNet relies on causal and dilated convolutions [25] and SampleRNN (2016), a recurrent neural network, ad-

ditionally works at higher temporal frames across its modules [21]. The models do yield accurate results, yet, since they have to generate each individual sample of the waveform one by one, they are computationally inefficient and ill-suited for operation in real-time systems [4].

More recently, different measures have been considered to make the process of modelling the waveform less intensive. Using the spectral representation, and hereby treating the spectrograms as images has had the advantage of making it easier to borrow from the much more well-accomplished task of image generation. An example of an audio-generating model that was a rather straightforward reworking of an existing model structure is Riffusion[3] (2022), inspired by a popular state of the art text-to-image diffusion model, Stable Diffusion (2022) [6]. (As well as Stable Audio, a very recent model created by the same company behind Stable Diffusion). Using the spectral representation, additional preprocessing of the data to obtain the STFT is required. Since the phases are discarded in the process (or simply does not exist for the generated spectrograms), the output will likely suffer some amount of phase distortion, and some method of phase reconstruction has to be considered as well [7] [9].

2.2.1 GAN-network structure

The use of a GAN-structure along with the spectral representation made for more efficient models that were better able to capture the global structure of the waveforms than the aforementioned auto-regressive models and in turn works at a much faster rate [12]. They did however present a new set of challenges, as to how to get the convolutional filters to efficiently capture the features of the audio.

A notable example is found in [9], were Donahue et al. present two models, SpecGAN and WaveGAN, both of which are reconfigurations of a deep convolutional GAN (also known as a DCGAN), which has been successful at generating realistic images [27]. These models were targeted for short audio segments, such as percussive sounds and spoken digits. For the first net, SpecGAN, the DCGAN was reconfigured slightly to work on spectrograms that would be then be transformed into waveforms and uses Griffin-Lim inversion for phase reconstruction. WaveGAN was made by flattening the structure to one dimension in order to create an end-to-end model that could be applied directly to waveforms in turn.

The two representations each have their own advantages and drawbacks. SpecGAN, which requires more preprocessing of the data, was found to converge at a faster rate and was seemingly able to capture more of the underlying variance. The output was labelled more accurately by evaluation participants compared to samples made by WaveGAN — however, the quality of the sound was perceived to be poorer. The authors note that this could be attributed to the choice of the

[3] `https://www.riffusion.com/`

phase reconstruction algorithm rather than it being a problem inherently tied to the method of using the spectral respresentation.

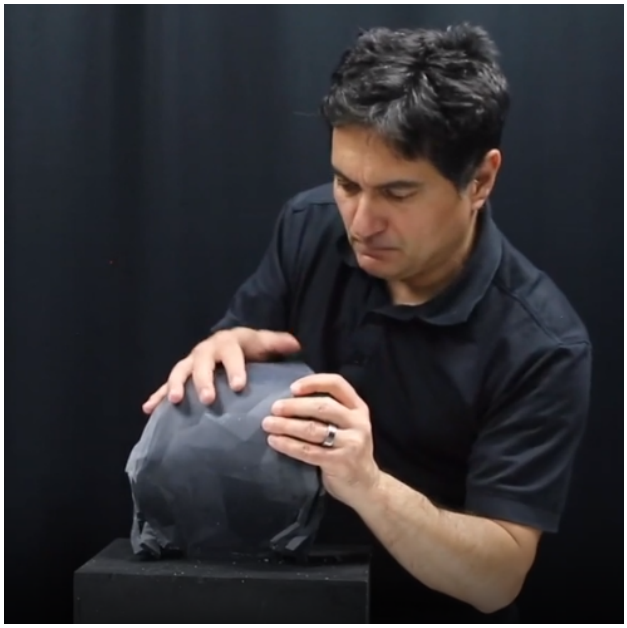

Figure 2.1: Showcasing of *AI-Terity* [28] during the performance of *Uncertainty Etude* #2.

Convolutional networks

A challenge when using convolutional networks for audio synthesis is that in order to capture the alignment of the phase of a signal, and hence, the periodicity of the waveform as a whole, the stride of the convolutional filter's frame would need to be of corresponding length. However, since an audio signal usually contain many different frequencies, this condition rarely holds [12]. The irregularities that arise as a result highly influences the perception of the signal. One solution has been to place restrictions on the model, such as pitch conditioning, as seen in GANSynth (2019) which was built upon WaveGAN by using it as the basis. Generating output where the fundamental frequency is fixed ensures that the frame of the convolutional filter can be optimized to accurately capture the harmonics (i.e. their phases).

Another similar problem arises when streaming the inference of the model (i.e. performing inference in real time). Convolutional filters rely on padding to have the output tensor fit the right dimensions. Usually, zero-padding is used, however, when several buffers are used to sequentially process the data, it will cause their output to be misaligned and create phase discontinuities [5].

Though, as an upturn, a prominent feature of convolutional networks is that the

an audio sample may be processed in parallel by its filters. This makes inference extremely efficient with GPU-devices, as opposed to the autoregressive models that restrict the processing device to work with one sample at a time.

A comparison made between WaveNet and a conditioned variant of GANSynth showed the latter to be about 53,800 times as fast when producing a 4-second audio segment[4] [12], making these models much more suitable for real-time synthesis.

Indeed, GANSynth has been used to provide the sound for a musical instrument, the *AI-Terity*. The instrument is a flexible, flat structure suited with pressure sensors to control the parameters of granular synthesis depending on amount and speed of applied pressure, as well as bending, as depicted in figure 2.1. The model was inferred to on a CPU-device with about 5 seconds of latency. The authors went on to extend GANSynth to gain better control of the sampling to the latent space by observing the parameters with Principal Component Analysis, and by using a GPU-device for inference, they reduced the latency to somewhere between 1.2 seconds and 65 milliseconds [28].

2.3 Embedded AI

The field dedicated to developing machine learning models for embedded and edge devices, TinyML, is an emerging one that is still scarce on learning material and platforms for non-experts to help get started on developing models [2] — especially within the context of audio development [26]. Application of TinyML-models has typically been seen in IoT (internet of things), such as for devices that uses voice detection, where the model listens for and activates the device with a keyword — but also for monitoring of potential emerging deficiencies in industrial equipment like wind turbines or to detect diseases in agricultural plants [2].

Within the NIME (New Interfaces for Musical Expression) community, there is a considerable interest in machine learning models that may be used for audio generation on embedded devices. A review of the utilization of machine learning in publications for NIME conferences within the last ten years[5] shows a gradual shift from shallow learning models toward deep learning in the latter half of the decade [17]. The authors explain this trend as linked to deep learning models becoming increasingly better at performing tasks of higher complexity, rather than emergence of new applications for machine learning models — they specifically point out the capability for sound generation as a contributing factor of the shift.

[4] WaveNet took almost 18 minutes to complete the task, whereas the GANSynth variant completed in 20 milliseconds.

[5] Starting from 2012 with the advent of the "breakthrough" in regards to the capabilities of deep neural networks

2.3.1 Sound generation for embedded systems

Currently, audio generation still requires models of a complexity that severely limits which environments they may operate in, especially for real time inference, without compromises such as low sample rates that degrade the quality of the sound. For embedded devices, the ideal would be to develop lightweight structures that are compatible with CPU, and are otherwise low on power consumption in order to avoid draining a battery-powered device quickly and avoid the device getting overheated [18].

However, some work has been done to further reduce the size of audio-generating networks and speed up their processing. These audio-generating models that may run on embedded hardware have been targeted for more powerful single-board computers like the Raspberry PI and NVIDIA's Jetson Nano. Alternative hardware such as Field Programmable Gate Arrays have also been noted for their proficiency in terms of running operations in parallel, but requires more expertise in terms of low-level programming skills than most embedded devices [18] [26].

DDSP

As a way of overcoming some of the challenges mentioned in regards to neural synthesis models in general, the Differentiable Digital Signal Processing (DDSP) library was created by yet another approach as to processing the sound. The DDSP-models are based on oscillators and linear filters, and uses convolutional layers as FIR-filters along with some conditioning (such as to only generate samples of a fixed fundamental frequency, f_0, and amplitude) to approximate the desired behavior of the filter [11]. Some of the usages for DDSP is timbre transfer of monophonic sounds (which is achieved with interpolation between latent parameters as well as extrapolation to pitches that have not been included in the training data), reverberation and dereverberation. The models are based on deterministic autoencoders[6].

In [11], both supervised and unsupervised models are considered for performing spectral modelling synthesis. During training, the models extract the amplitude and pass it directly to the decoder. The encoder extracts f_0 as well as the residual data, which is stored as a latent parameter, based on the MFCC-coefficients[7] of the audio, using 30 features to describe each frame.

While DDSP is not specifically targeted to run on embedded devices, its lightweight structure could make it a suitable candidate for some devices[8] — such as for the NVIDIA Jetson Nano, which was attempted for an instrument called the

[6]Deterministic meaning the model produces the same result when given the same input sample.

[7]The Mel-Frequency Cepstrum coefficients, which is a small set of features to describe the spectral envelope of a signal.

[8]`https://acids-ircam.github.io/workshops/nime2022/resources.html`

raivBox, which targeted DDSP as part of a timbre transfer pipeline[9].

RAVE

RAVE is a combination of a VAE (as the overall model structure) and a GAN (for training of the decoder network) for real-time synthesis. The authors have drawn inspiration from several different audio-generating models in order to optimize it for real time use. In addition, RAVE was also created to be able to run (without compromises such as lowering the sample rate) on less powerful devices such as CPUs found in standard laptops, as well as embedded devices [4], making these more accessible than the works presented previously in section 2.2. Another feature of RAVE is that it performs unconditional generation, meaning high-level parameters of sound such as pitch and amplitude is entirely dependent on the model's distribution.

Inspired by Multi-band MelGAN, a real-time text-to-speech generator by Yang et al. [31], RAVE decomposes waveforms into multiple smaller frequency bands in order to speed up processing[10]. The frequency bands are further compressed into a latent representation of a smaller dimensionality. This process was in turn also found to make the model converge faster during training [4].

The difference between real and generated waveforms, respectively x and $\hat{x}$, is measured with a loss function called multiscale spectral distance (difference between magnitude spectrograms $||x_n - \hat{x}_n||$ and log magnitude, both L_1 with an FFT size of n). This distance was introduced by Engels et al. who argue that functions that measure point-wise loss is not suitable for waveforms, as two different waveforms that are perceived to be similar may have very little correlation in terms of point-wise distance [11]. An addition by Yang et al. is minimizing the spectral convergence (using the Frobenius norm $||\cdot||_L$ which is defined as the square root of the absolute squares of the elements within a matrix) [31]. The resulting distance $S(x, \hat{x})$, based on the STFT with a window size n for a given set of scales, N:

$$S(x,\hat{x}) = \sum_{n \in N} \frac{||STFT_n(x) - STFT_n(\hat{x})||_L}{||STFT(x)_n||_L} + log(||STFT_n(x) - STFT_n(\hat{x})||_1) \quad (2.2)$$

The Frobenius norm[11] of a $n \times m$ matrix A:

$$||A||_L = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2$$

[9] `https://github.com/jacktipper/raivBox`

[10] The decomposition consists of 16 bands, which the authors found sufficient for reconstructing the audio with a sample rate of 48kHz.

[11] `https://mathworld.wolfram.com/FrobeniusNorm.html`

This distance is then integrated in the ELBO loss function 2.1 in order to refine both the encoder and the decoder.

$$\mathcal{L}_{vae}(x) = -\mathbb{E}_{\hat{x}\tilde{}p(z|x)}[S(x|\hat{x})] + \beta \times \mathcal{D}_{KL}[q_\phi(z|x)||p(z)] \tag{2.3}$$

RAVE divides training into two stages; during the first stage, the network learns to encode the audio data into latent representations of increasingly more compact dimensionality. This is usually the objective considered when training VAEs [4]. The second stage is devoted to further improve the quality of the audio with adversarial training, hereby introducing a discriminator to determine between x, and $\hat{x}$, from the distribution (latent space of the encoder). In order for this approach to work, the encoder should be able to make accurate representations and should not need to be optimized further. Therefore, only the decoder is trained during the adversarial stage. A feature matching loss is considered along with the spectral distance to measure the divergence between x and $\hat{x}$.

Unless configured otherwise, RAVE-models are non-causal during training. They are composed of convolutional filters that use zero-padding to avoid reducing the dimensionality of the output[12]. This would pose a problem for inference, as the zero-padding causes phase discontinuities between consecutive audio buffers (i.e. meaning the model would not be suitable for real-time use). Therefore, once training is finished, the model is reconfigured to be causal by using strided convolutions with no zero-padding, as well as adding cached padding to retain the structure of the audio [4].

Neurorack

Neurorack is a modular synthesizer that uses the NVIDIA Jetson Nano for neural audio synthesis. It features a lightweight model structure that is more similar to traditional DSP-like elements; most notably a neural source filter model (SincNet) that consists of trainable filters[13]. This model is able to generate impact sounds, which are usually hard to produce with traditional synthesis methods (apart from sampling, which does not allow for generating new variations of a sample) [7]. This is achieved by using two sources for the sound, Gaussian noise and a sinusoidal harmonic source, that are passed into the trainable filters (SincNet in the case of the noise, and a neural filter block for the harmonic sample). Before merging the two sources, they are respectively further processed by a bandpass and a lowpass filter. This approach requires relatively little data and few hours of training compared to what is usually necessary for a convolutional network[14]. It also allows control over

[12]Which will happen if the kernel of the filter is larger than 1. The discriminator part of the RAVE-models have convolutional filters have a kernel size of up to 15.

[13]SincNet was first developed for models capable of speaker recognition.

[14]Though, it should be noted that the training was carried out on a powerful, state of the art GPU, the NVIDIA Titan V, which in part accounts for the efficiency of the training.

high-level features of the resulting sound, such as brightness, richness, noisiness and percussivity along with amplitude and pitch [7] [11].

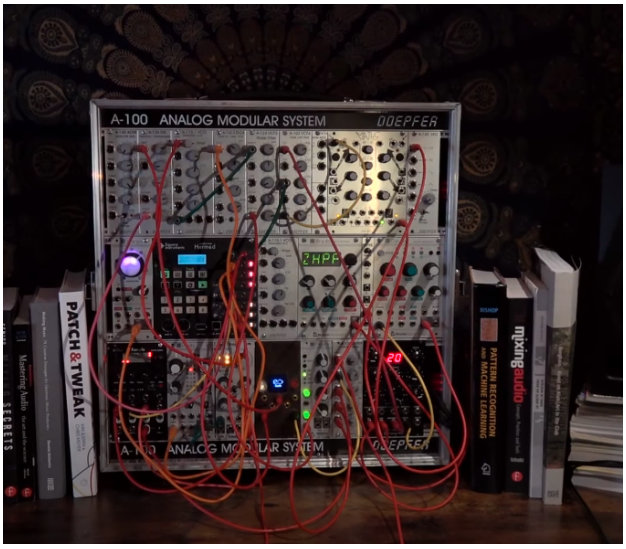

Figure 2.2: The Neurorack module [23] (containing a Jetson Nano inside).

Bela deep learning pipeline

While RAVE and Neurorack are state of the art models that can run on specific embedded devices, Pelinski et al. point out that generally there is a lack of resources (and support) that allow non-expert programmers to compile deep learning models targeted for the purpose [26]. They provide a pipeline for training of neural networks that might run on the Bela board, which is a single-board computer often used by audio developers. For inference on the Bela board, the models would need to be targeted for CPU — which can be seen as a constraint, insofar that many deep learning models rely on matrix multiplications and therefore run more efficiently with GPUs. On the other hand, Pelinski et al. also argue in favour of CPUs, as they are less power-consuming, making them more suitable for use in embedded systems, and may be the better choice for certain types of models.

Model	Structure	Audio representation
WaveNet (2016)	CNN	Waveform
SampleRNN (2016)	RNN	Waveform
SpecGAN (2018)	DCGAN/WGAN	Spectrogram
WaveGAN (2018)	DCGAN/WGAN	Waveform
GANSynth (2019)	GAN	Spectrogram
DDSP (2020)	Auto-encoder	Spectrogram (decomposed)
Neurorack (2021)	Neural Source Filter	Spectrogram (decomposed)
RAVE (2021)	GAN/VAE	Spectrogram (decomposed)

Table 2.1: Overview of mentioned model structures and which audio features they learn to produce.

Chapter 3

Development

Developing generative models that are able to run on embedded platforms in real time is of great interest, since it would allow for building musical instruments based on these models, making musicians able to explore the possibilities for the alternative ways of synthesizing sounds that they present. Yet, as mentioned already, the complexity of most generative models places a rather big constraint on which platforms they can run on, as they are resource-heavy and so, often not suitable for embedded microcontrollers without severely limiting the quality of the generated sound output.

This project has been concerned with finding an appropriate middle ground between the contrasting requirements of generative models and resources available on embedded platforms. This chapter describes many of the considerations that had to be taken along the way to build a device — a preliminary for a musical instrument — based on a generative model.

3.1 Requirements

Inference

- An embedded device capable of processing audio as well as inferring from a (lightweight) neural network structure.
- A device that might support different modes of interaction, such as reading measurements from various kinds of sensors.

Training

- A neural network structure that is able to generate audio samples in real time.

- Datasets containing different sounds that might be desirable to synthesize new variations of in a musical context.
- A cloud platform or a device with resources enough for training (preferably with a GPU).

By far the most computationally demanding part of creating a model is the training stage — fortunately, this does not have to take place on the embedded device the model is to be deployed on. Usually for TinyML, the model is trained on a host machine from which the model is transferred to its target device; using tools such as `TensorFlow Lite` and `PyTorch`, trained models can be compressed and pruned in order to take up less memory.

3.2 First iteration

3.2.1 ESP-32 Boards

Early considerations for an embedded platform was for a board that would be able to support existing frameworks for machine learning. Within the recently established field of TinyML (machine learning on embedded devices), a recommendation was to use `TensorFlow Lite`, which is targeted for edge/embedded devices. The "building blocks" of neural networks could then be assembled and compiled to a compatible board, such as a board with an ESP-32 System-on-chip. And indeed, the "Hello World"-example[1] to get started with TinyML can be utilized for generating a (simple) waveform — the example teaches one to train and deploy a model that can predict a sine wave based on an incoming stream of values.

The model was first tested on SparkFun's ESP-32 Thing Plus, where a DAC-pin was connected to an oscilloscope to display the resulting sine wave. In order to generate the sine wave as audio, the model was deployed to AI-Thinker's ESP32-Audio-kit, which was more suitable for audio development. Using an example script called TfAudioStream[2], the data stream from the model was converted to I2S for audible output. The board was able to produce real time audio without drop-outs when the sampling rate was set below 20k Hz.

3.2.2 Building and training of DCGAN

The first attempt at creating a suitable network was, similar to the `SpecGAN` model suggested by Donahue et al. [9], building upon the architecture from image generating models. An example of a DCGAN capable of generating low-resolution images (64x64 pixels) was modified to be able to generate spectrograms in the

[1] `https://www.tensorflow.org/lite/microcontrollers/get_started_low_level`
[2] `https://github.com/pschatzmann/arduino-audio-tools`

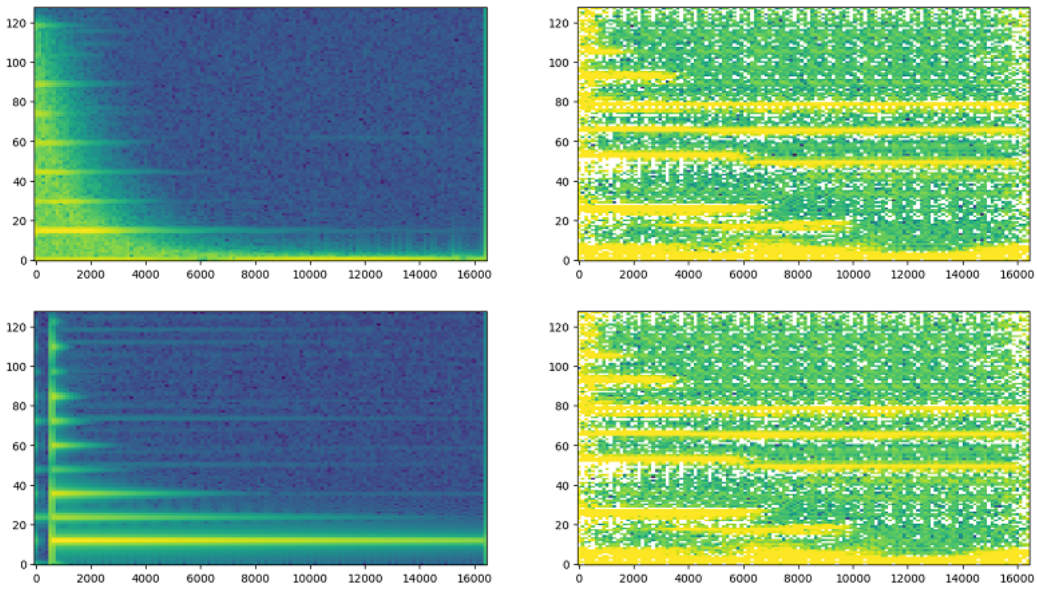

Figure 3.1: Comparison of examples from the *NSynth* dataset (left) and results obtained from the modified DCGAN (right).

dimensions of 128x128. A subset of the Magenta *NSynth* dataset [13] was used, containing bass notes of different pitches. Validation was performed for every 100 iterations, with each epoch lasting about 2 hours. However, the model was not found to converge much further to the data after about 5-6 epochs, with some of the best results displayed in figure 3.1. The generator was able to capture some of the structure of the harmonics in a decaying note, but the output was marred by empty white specs, which would prevent them for being suitable for conversion to waveforms with inverse Fourier Transform. Additionally, the model was also found to suffer severely from mode collapse, with no variance in the sample output — as if the entire learned distribution amounted to just a single sample. While the distribution would change after each epoch, the model was not found to get any better at creating varying or even more realistic features within the spectrograms.

3.2.3 Conclusion

The task of creating a generative model from scratch to run on the ESP-32 boards was deemed to be out of scope for this project. Instead, relying more on one of the existing state of the art structures as well as a more powerful device, more emphasis might be placed on the interaction with the deployed model — essentially, creating something approximating a NIME featuring a generative model, rather than attempting to build a novel lightweight model structure.

3.3 Second iteration

3.3.1 NVIDIA Jetson boards

Two aforementioned state of the art models for sound synthesis on embedded devices, namely RAVE and Neurorack, has been succesfully deployed in real-time on the Jetson Nano board. Inference with very low latency is possible, as the GPU featured in the board can be used to speed up inference. In the case of RAVE, for comparison, a lightweight model can run on a Raspberry Pi 4 (with 8GB RAM) with about 200 milliseconds of latency, while a regular RAVE model can run on the Jetson Nano with about 25 milliseconds of latency[3].

The Jetson series is a selection of single board computers that was created specifically for developing machine learning applications and includes both appropriate hardware and software tools for the purpose[4]. Notably, as opposed to the vast majority of embedded devices[5], they include GPUs. As mentioned in 2.3.1, deep learning models tend to achieve better performance when targeted for GPUs, since they are equipped for running tasks in parallel [22] [26]. The Jetson boards also include a distribution of Linux, which makes them compatible with a wide range of software tools, such as CUDA (a toolkit for NVIDIA GPUs) and Python packages for machine learning (PyTorch, TensorFlow, Flask, etc.).

RAVE on Jetson

RAVE-models can be deployed for real-time use in Max/MSP and Pure Data, which are popular environments for musicians who rely on audio processing as part of their performance. Both environments are visual, and allows for patching together functional objects, such as filters, to process incoming data streams. Pure Data is compatible with Linux, and so, it can run on embedded systems such as Raspberry Pi and Jetson Nano.

In order to import the RAVE models into Pure Data or Max, an extension developed alongside RAVE abbreviated as "`nn~` " (nn_tilde) can be used as a container for the model for inference[6]. The encoder and decoder can be separated, which allows for the latent space of the model to be manipulated. The encoder has one inlet, which allows for letting it parameterize an input signal according to it's learned distribution. The decoder has a number of inlets corresponding to its latent parameters for passing in any signal, encoded or not, in order to activate or amplify

[3] `https://www.matrixsynth.com/2022/05/acids-rave-realtime-audio-variational.html`

[4] `https://elinux.org/Jetson`

[5] For use on mobile devices, CPUs have several practical advantages over GPUs, such as being less power extensive, more portable and offering better software support. GPUs, notably good for parallel processing, is rarely an optimal choice for the tasks performed by embedded devices.

[6] `https://github.com/acids-ircam/nn_tilde`

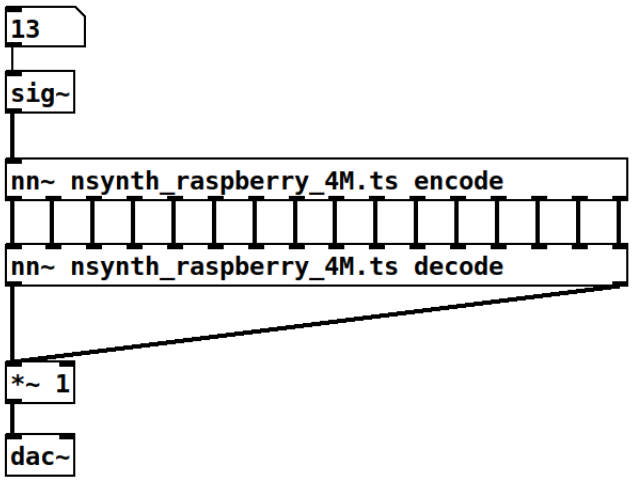

Figure 3.2: Simple implementation of a RAVE-model named `nsynth_raspberry` in Pure Data. A noise signal (a number sent to the "sig~"-object) is passed into the encoder, whose outlets are connected to the 16 latent parameters of the decoder. The output of the decoder is sent as output to the sound card by "dac~".

certain parameters.

3.3.2 Setting up for inference

It was decided to use the existing "pipeline" of RAVE with nn_tilde as a baseline for a model to be deployed on the Jetson Nano. This board, being already known as a suitable candidate, was first considered. However, compiling nn_tilde for Pure Data using CMake failed to build the extension, which was later attributed to RAVE's most recent dependencies on certain Python modules (most likely CUDA and PyTorch) being incompatible with the Jetson Nano[7].

Luckily, a more recently launched and more powerful device was made available, the Jetson AGX Orin. Indeed, `nn_tilde` was built succesfully upon first try, since newly updated versions of CUDA and PyTorch could be installed. Due to its size, the Orin moved the project slightly beyond the sphere of embedded machine learning, but was an acceptable compromise given that inference from the RAVE-models was found to be working for generating audio in real time. Likely, with earlier versions of the software used for RAVE, this would also have been the case for the Nano, albeit with more latency than on the Orin.

[7]Possibly, after 5 years since the launch of the Jetson Nano, it might be considered a slightly outdated piece of equipment.

Figure 3.3: The Jetson AGX Orin and its desktop setup, which included a monitor, keyboard, mouse and an external sound card.

3.3.3 Training

While it is possible to train RAVE using CPU only, the creators do not recommend it, as it takes significantly longer and might also affect the accuracy of the model. However, setting up the required Python-packages for RAVE in a GPU-environment proved difficult with the hardware and cloud-solutions available initially.

Summation of the difficulties in setting up modules for RAVE-training:

- When testing remote options for training (such as UCloud), a work-around could not initially be found in terms of making a Python script where the functions found in the RAVE package could be accessed. Another trade-off to consider was also whether access to GPUs could even be supported through the university, which meant priority was for finding alternatives instead.
- Python packages were incompatible with the architecture on the Jetson Orin: RAVE uses PyTorch version 1.13.1. This was both a problem on the Jetson Nano (CUDA could not be fully upgraded to be compatible) and the Jetson Orin (no existing wheels was found for building the required version of PyTorch along with GPU-support for CUDA specifically on aarch64-devices).

Datasets used for training

It was recommended by the creators of RAVE to use a minimum of 2-3 hours worth of audio to avoid overfitting when training the models, although, ultimately,

reaching convergence varies depending on other attributes about the dataset, such as coherency, that might not be easy to predict before training. The following dataset was selected:

- Part of the Magenta *NSynth* dataset (the subset usually used for validation), introduced by Engels et al. in [13], consisting individual notes played by 10 different instrument groups (e.g. strings, brass, guitars), both acoustic and electronic, ranging in all 88 pitches found on a standard MIDI piano. It amounted to a total of 9.5 hours.
- Part of *SASS-E*, a set of one-hit samples of tenor steel drums at different pitches and of varying timbres [20]. For training, this dataset was combined with bass notes from NSynth, including which it amounted to nearly 4 hours of data.
- *ff1010bird*, a dataset containing recorded audio with birds, as well as ambient noise [8]. Unlike the two musical datasets, the quality of the audio was varying from sample to sample. The whole set was used, which totaled to a little more than 19 hours.

Jetson AGX Orin

Given the presence of fairly powerful hardware components, a NVIDIA GPU (with 2 GB of dedicated memory) and CPU with 12 cores, the Orin could also potentially have been able to perform training of RAVE-models with decent results — and it made for an appealing idea to have a "self-contained" device that could perform both the inference and the training. However, training of RAVE (as opposed to inference) required a specific version of PyTorch, which could not be supported by both the AArch64-architecture for the Orin's CPUs as well as compatible versions of CUDA for the GPU. Training could therefore only be carried out using CPU.

RAVE has different configurations to make it optimal for different kinds of usage (e.g. the model can be made to be discrete instead of continuous, to support a open-source framework called onnx, or lightweight enough to do real-time inference on a Raspberry Pi). For running on the Orin, the standard model (version 2) was found to operate in real time. However, training proved to be tediously slow — what amounted to several hours (12-18) worth of training resulted in models that could only produce a single pitch at best, but for the most part simply produced noise.

[8] The dataset was located at `https://archive.org/details/ff1010bird`.

Stationary Windows Computer

An alternative that appeared later in the process was a stationary windows computer with a NVIDIA GPU. The setup of a compatible version of PyTorch (11.7) was straightforward; so was creating a miniforge environment with the required packages for training. The same structure as on the Orin was initially used, version 2 of a standard RAVE-model along with the *NSynth* dataset[9]. The GPU on the stationary computer — an NVIDIA GeForce RTX 3080 — proved to speed up progression of the training significantly and give better sounding results much quicker. However, the audio, when playing back audio in Pure Data, was occasionally glitching slightly. This was not found to be improving after a long stretch of training for the recommended 2-3 million steps. The lightweight "raspberry" configuration was tried instead. In addition to training at a much faster rate[10], these models also didn't suffer the problem with glitching, although the audio it produced was perceived to be at a lower resolution, making the sounds have a more obvious synthetic quality to them.

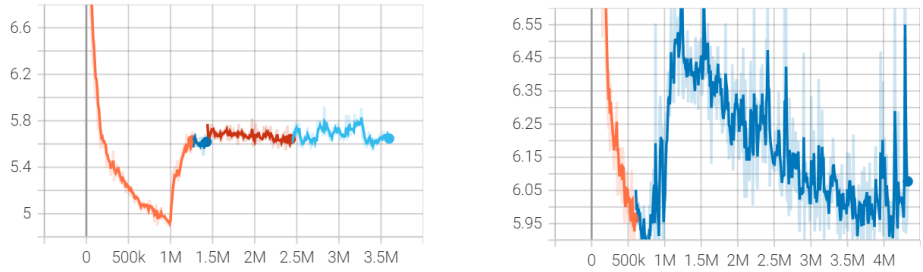

Figure 3.4: Measured distance for validation (every 10.000 steps) of the two instrument models (Left: `subset_raspberry`, Right: `nsynth_raspberry`). Notice that the increase at 1M steps was expected, due to the onset of the second phase of the training. Although the average value of the loss was lower for the `subset_raspberry` model, `nsynth_raspberry` was found to converge better, as the value decreased steadily in relation to its starting point. This was also reflected in the quality of sound, with `nsynth_raspberry` being able to produce much more varying sounds with more fidelity to the real data.

Model architecture

The initial testing proved that latency of the larger models was not a concern. Comparably, getting the models to learn to produce variance in frequency, both in

[9]Both the ELBO and the Wasserstein optimizers were tried for about 100.000 training steps for comparison, with much divergence from one another. The Wasserstein regulator was chosen for further training.

[10]Measuring the number of steps within the first hour revealed it to be performed 31 times faster than CPU training on the Orin.

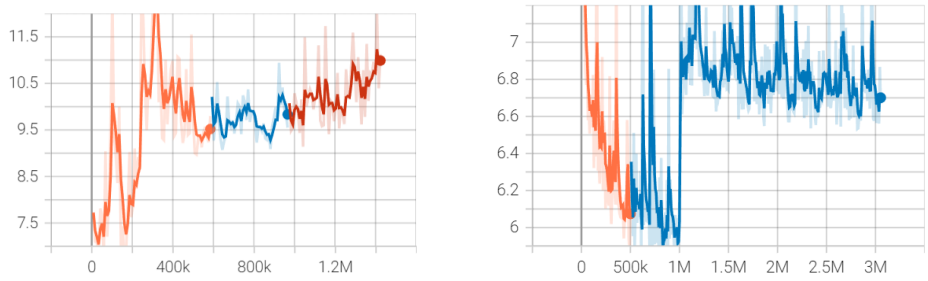

Figure 3.5: Measured distance (every 10.000 steps) for validation of the bird models (Left: `birdamb_v2`, Right: `birdamb_raspberry`. Note the second training phase started at 200k for `v2` and at 1M for `raspberry`. The `v2`-model was found to produce sound at a slightly better quality, yet neither was found to improve much during the second phase of training — which correlates with the lack of decrease in loss estimation pictured here.

terms of pitch and timbre, required several iterations.

Two configurations (defined by the developers behind RAVE) was tested: v2 (version 2 of the default RAVE-model) and raspberry (a lightweight structure more suitable for embedded inference). Both configurations are based on non-causal convolutional filters. Training can be carried out using causal convolution to reduce latency, however, latency was not deemed to be a problem during assessments of the progress of the models, and was inferred in Pure Data with a delay of 20 milliseconds.

Best results for the setup:

- `subset_raspberry`: After attempts at applying the *SASS-E* dataset (along with the *NSynth* bass notes) to a v2-model had given slightly better sounding results, the raspberry configuration was given a try, to see if a smaller model might be a better fit for a smaller dataset. Training for this configuration went faster; the model reached 3.6 million steps after 66 hours. The latent space had the default configuration of 128 latent parameters, giving more than plenty of ways to try and vary the sound, which was found to be more nuanced than the v2-model.

- `nsynth_raspberry`: The *NSynth* dataset was also tried for this configuration, and very quickly gave significantly more promising sounding results than its earlier v2-counterpart. As shown in figure 3.4, the model seemed to converge very well with the data. This model was given 20 latent parameters and was trained for a total of 4.3 million steps over 81 hours.

Additional models:

- `birdamb_raspberry`: This raspberry model was trained on a much larger dataset than the previous two, the *ff1010bird* dataset. It was trained for 3 million steps for 58 hours with 20 latent parameters. This model was, however, found to suffer from mode collapse, possibly because it was underfitting to the vast amount of data.

- `birdamb_v2`: This default v2-model, with Wasserstein regularization, trained for 1.4 million steps over 80 hours, with 16 latent parameters, was better able to replicate the *ff1010bird* dataset, although, the model was not found to improve much after starting the second phase of training, as also depicted in figure 3.5, and further training was dropped.

3.4 Prototype for interaction

Although the initial idea was to train a RAVE-model capable of timbre transfer, another intriguing option was to create sound by interacting directly with the latent space of the decoder network. This would allow the musician (i.e. user) to select which latent parameters to activate.

Since none of the configurations provided by the creators of RAVE places any conditions (apart from sample rate and number of latent parameters) on the sound, the mapping of sound parameters (i.e. amplitude, pitch, timbre, etc.) to the latent space is entirely dependent on the model. Thus, activation of different latent parameters resulted in the sound being shaped differently, in a way that was obscured to the user. In order to evaluate the potential for interaction with the trained RAVE-models, a prototype was developed to afford the user control of the activations.

3.4.1 Design

Size of the interface

The Orin, due to its larger size and the fan sitting on top of the board, might have been an appropriate choice as the processing unit for a typical synthesizer or an instrument kept in a stable position on a desk. In other words, while it was portable, it could not be embedded into an instrument that one might pick up, move and tilt around. Another restriction was that while the GPIO-pins might support digital sensors with some modification, it was not the case for analog sensors.

The ESP-32 Thing Plus, however, while not powerful enough to perform audio processing, had a wide range of other features that seemed like a good supplement to the Orin. The system-on-chip has both a bluetooth and WiFi-transceiver,

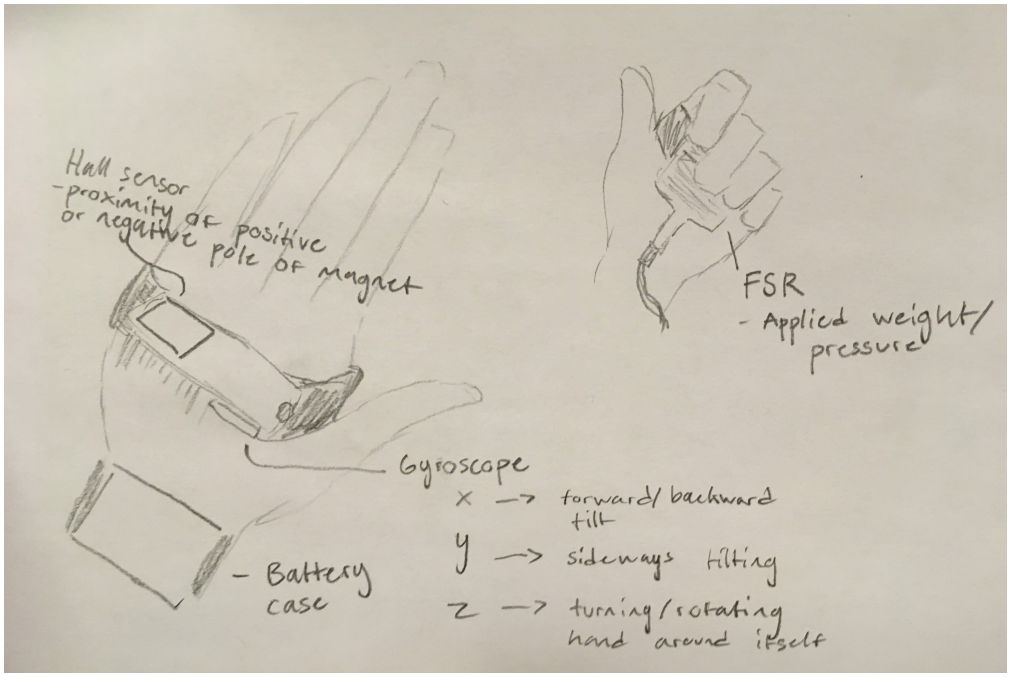

Figure 3.6: Design sketch showing the chosen positioning of components and sensor readings that would activate the latent parameters of the RAVE-models.

meaning the board could be used to communicate with the Orin wirelessly, and transmit data from sensors connected to any of the 21 GPIO-pins on the board. The ESP-32 was in turn also very light and compact, ideal for fitting onto the body of an instrument or for using as part of a small or wearable device.

Form of interaction and choice of sensor

The Orin could thus be the main device for processing — receiving an incoming stream of data from the ESP-32, mapping it to the latent parameters of the RAVE-models and send the audio output to a sound card connected by USB.

The lack of direct high-level sound parameters had obvious implications for the question of how to afford control of the latent space of the models. A small controller-like instrument to be mounted on one hand of the user was decided to be appropriate for nevertheless exploring the concept of controlling the RAVE-models with an embedded device (as a compromise to the original idea, interaction would thus be handled by two separate devices).

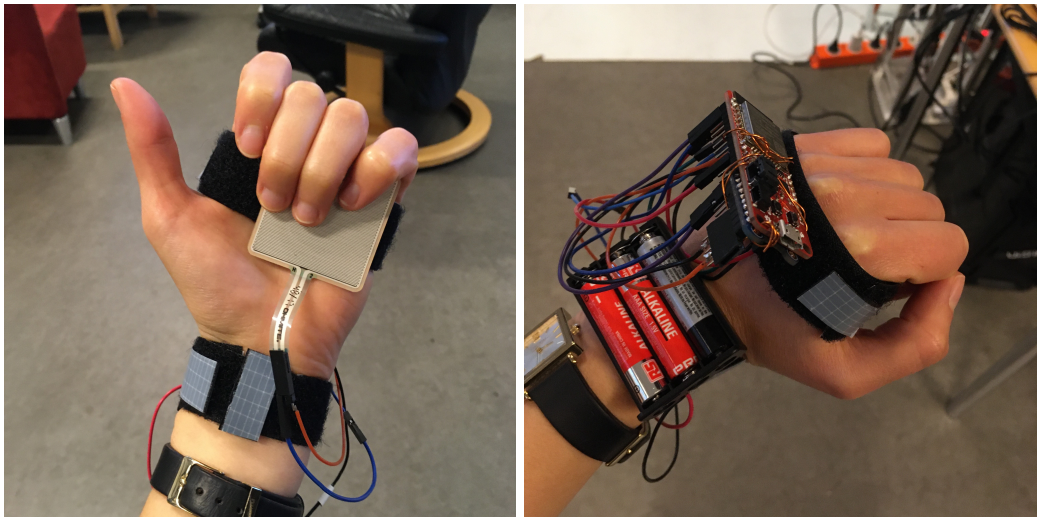

Figure 3.7: Prototype of the controller. Attached with strips of velcro, a FSR was placed in the palm of the hand, the ESP-32, batteries, gyroscope and most of the circuit was compact enough to be able to fit on the back of the hand.

Three different sensors was used for the purpose of building a compact controller:

- **Hall sensor**: A sensor that was built into the chip of the ESP-32. It could sense changes in the magnetic field around it — such as proximity of a magnet. The voltage of the sensor was changed accordingly. For the prototype, it was be placed on the back of one of the hands of the performer/musician.
- **Gyroscope**: A digital sensor which could measure the velocity of rotational movements in three dimensions (x-axis, y-axis and z-axis). The gyroscope was mounted underneath the ESP-32 to keep it in a steady position (and thus measuring in relation to the performer's movements).
- **Force sensitive resistor (FSR)**: An analog sensor whose resistance depended on applied force, thus varying how much voltage passed through it. This sensor would be placed in the palm of the hand as pictured in figures 3.6 and 3.7. The sensor was also connected to a voltage divider using a 1.5K resistor placed in series to read the values through an analog pin on the ESP-32. This circuit (which included the power inlet to the gyroscope) was small enough to be fitted next to the gyroscope underneath the ESP-32.

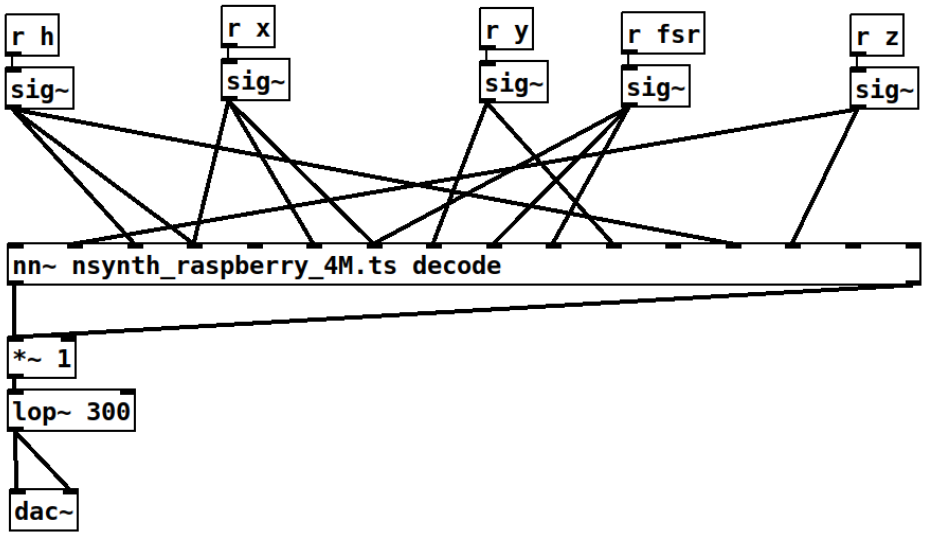

Figure 3.8: The decoder of `nsynth_raspberry` in Pure Data. The data from the sensors was received and passed to the inlets of the latent parameters. A one pole low-pass filter was applied to the generated output as the model could produce sounds that were noisy and grating at times.

3.4.2 Implementation

Mapping

The incoming stream of values from the sensors was scaled and applied directly to the latent space of the decoder part of the models in Pure Data. It was found that activating different latent parameters separately or together made for a way in which the timbre and amplitude of the sound could be controlled in a predictable manner once the latent mappings had been recognized. The pitch of the sound could only be varied slightly in the same manner. The gyroscope felt like a fitting choice, insofar it could allow for more variation in the reading, responding to more subtle changes in the movement, as well as adding a performative element to the interaction. It would also better allow for making measurements (thus, more variation in the sound) simultaneously with both the FSR and the hall sensor, which both required one hand each to operate.

Setup

The ESP-32 was programmed with the open-source Arduino IDE, which included libraries used for reading from the gyroscope and setting up a UDP-server to communicate wirelessly to the Orin over a WiFi-connection. In the `setup()`-function of the Arduino script, a soft access point was configured to be hosted on the ESP-32.

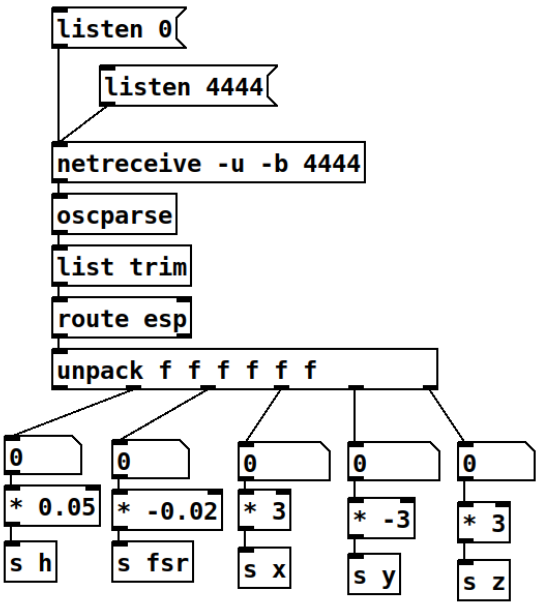

Figure 3.9: The pipeline created for receiving OSC data from the ESP-32 in Pure Data. By default, the `netreceive`-object listens for UDP-messages from port 4444, the port that the ESP-32 was set to stream from. `oscparse` is able to interpret an OSC-packet from the UPD-message. Directing to the first part of the message "esp" with the `route`-object, the data from the sensors could then be unpacked as separate entities, in this case, floating point values. The values was scaled before being sent to the inlets of the decoder, as depicted in figure 3.8 above.

Inside `loop()`, the values read from the sensors was updated every 60 milliseconds and sent as an OSC-message inside a UDP-data packet to be unpacked in Pure Data on the Orin when it was connected as a client to the "server" set up on the ESP-32.

Both the readings from the FSR and the gyroscope would have a value close to zero when no interaction was happening. The hall sensor, on the other hand, would always give a reading somewhere above zero without a magnet in proximity. A threshold was thus set to define when the reading from the sensor would be valid and thus sent through the UDP-connection. This worked to ensure that when the controller was kept static by keeping the hand still, the readings be zero or negligible (and could therefore be used to determine onset and offset of the sound on its own).

Chapter 4

Evaluation

The evaluation consisted of a usability test of the prototype and a questionnaire. This assessment was carried out in order to determine how the interaction afforded by the prototype, as well as the RAVE-models (or, indeed, for a similarly functional model) would be received by its intended target group, people who play and compose music.

The testing was carried out by eight participants in a quiet laboratory setting, where they got to try and get familiar the mappings of the prototype for at least 15-20 minutes after some instruction on how the sensors worked. This was followed by a questionnaire, where participants were asked to evaluate both the experience of playing the instrument and their impression on the sounds the RAVE-models were capable of creating. Half the questions had participants rate their agreement with the sentiment on a 7-point Linkert scale[1] inspired by [3], while the rest were intended for them to elaborate on their opinion. These questions are included in appendix A.

Prior to the test, information on musical background of the participants was also gathered. Experience with playing one or more instruments ranged from 10-40 years (Avg. 21,5 years), experience with production of sound or music ranged from 1-30 years (Avg. 11 years). Seven out of the eight participants had professional experience in working with sound or music. Two were familiar with RAVE, having trained models of their own, and at least three others had experience in working with machine learning for other tasks.

The Orin was connected to a set of loudspeakers through a mixer by USB, so the sound was played out loud in better quality than the initial sound assessments that were done intermittently throughout training the models and assembling the prototype. Three models were selected; two as a result from the training, `subset_raspberry` and `nsynth_raspberry`. Since the bird sound models neither produced particularly pleasant nor varying sounds, for the experiment, a bird

[1] With the lowest value 1 indicating strong disagreement and highest, 7, strong agreement.

Questions	Q1	Q2	Q3	Q4	Q5	Combined avg.
Participant 1	6	5	7	6	6	6.00
Participant 2	5	6	6	5	3	5.00
Participant 3	7	7	7	5	3	5.80
Participant 4	4	4	7	3	7	5.00
Participant 5	3	4	6	3	2	3.60
Participant 6	6	6	6	5	5	5.60
Participant 7	5	3	7	3	6	4.80
Participant 8	3	5	4	3	3	3.60
Average score	4.88	5.00	6.25	4.13	4.38	4.93

Table 4.1: The ratings obtained from the 7-point Linkert-scaled questions listed in appendix A. The test participants were asked how well they agreed with the sentiment of each question, with 7 being the highest rating.

model named `Pluma` recently released by the Intelligent Instruments Lab of Iceland University of the Arts [16] was employed instead, so that participants would be able to try and synthesize sounds that are normally much more difficult to achieve with other forms of synthesis.

The participants noted the sound quality of the setup to be decent, but also had varying perspectives; some found the different sounds to be realistic, while others responded that though they liked them, they also perceived them to suffer from artefacts such as clipping or in need of more conditioning.

4.1 Feedback

Overall, the participants seemed to be mostly positive about the prototype. When making the distinction between evaluating the functionality of the RAVE-models and functionality of the controller itself, the participants seemed to have a preference for the latter. Possibly, this could be explained as the assessment of the controller as a physical artefact is more tangible. Noting their points of improvement, the participants turned their attention to the sound rather than the way of interacting with the controller.

4.1.1 Sound

Though some participants mentioned during the test that they thought the sound to be very simplistic or untreated, most also noted they liked the sound selection and the way it interchanged. Most had suggestions for ways to improve the sound

that pointed in different directions; some noted the current selection of sounds to be fitting, but that they could be improved through additional processing, such as by having filters applied to better shape the sounds. Assessing the results of the Linkert-scaled questions 4.1, the attribute of the prototype which were rated the lowest was how well the musician were able to control of the pitch and timbre of the sound (Avg. 4.13). Ability to control the amplitude (Avg. 4.38) was only rated slightly better. This was reflected in their answers from the subsequent questions:

> *I felt the high frequencies were a little uncontrollable and it felt hard to transition between the high and the lows.*

> *I think [the sound of] this instrument has a lot of bass and it would be better with a balance to the treble.*

> *I liked the sounds a lot, maybe a bit too boomy sometimes, but I thought it sounded cool for soundscapes and experimental music.*

Others seemed to be more concerned with the selection of and transition between parameters in the latent space of the models, stating they would have liked to be able to have a more clear distinction between the directions of the sound or instruments included (and further, suggestions about sounds from nature or percussive instruments were also mentioned by several participants):

> *Maybe some of the sounds provided could be separated a bit more, instead of using a model based on a lot of instruments together. At the same time, it creates a nice blend!*

> *The sounds where a bit alternative for my taste. I think it could be interesting to synthesize e.g. a symphony orchestra playing, or just a single instrument, like a violin.*

As mentioned, the unconditional generation that RAVE is capable of means more direct mappings are hard to achieve, and the attributes that generates different nuances in the sound are scattered across the latent space. The participants seemed divided as to whether the preference would be to have more "concise" directions of sound or being able to get more varying results with the merging of the latent parameters:

> *I believe that you are able to create some very unique transitions between the sounds which might sound really good by themselves.*

> *Like most instruments, it affords an exploration of its possibilities, a dialogue to uncover performative fields. At the same time it feels like possibilities are too many and too few: too many sounds hidden somewhere in the latent space, with only few options being easily accessible, and the others popping up sometimes.*

> *It's a more 'blended' kind of sound [...] This could maybe be achieved with sampling, but since you're not synthesizing it, the control capabilities become quite limited. Whereas with your device, you could hear the sounds blended together and control them in a fluid way.*

4.1.2 Control

Throughout testing, the participants seemed intrigued by the modalities of the controller, especially when switching between the models[2]. Most participants preferred to stand still in front of the loudspeakers, though some also started wandering about in the room while trying different gestures. For the most part, they got to vary their interaction in subtle ways that could bring forth the nuances in the latent space of the models — though some also had to rely a lot on repetitive movements in order to establish a link to what aspect of the sound they were controlling. While it could be interpreted as a sign of confusion about the indirect means of control, often they would also seem focused and engaged, as some mentioned themselves to be:

> *It's innovative. However, operating it takes practice. Even small movements can has a big influence on the volume and dynamics.*

> *It was very interesting and I got absorbed and kind of hypnotized to try out and experiment with all the sensors, the possible movements and details.*

The more positive attitude towards the interaction itself was reflected in the answers to the questions. Being able to change the sound towards a certain desired direction (Avg. 4.88) and merging the sounds (Avg. 5.00) was rated as easier than changing the sound parameters themselves. The majority of the participants also found the changes to the sound was influenced by their movements (Avg. 6.25), some noting it to be intuitive and fitting for performing. On the other hand, one stated finding the interaction unsuitable as a means of controlling the latent spaces of the models and that latency affected their experience:

> *Gestures are often too coarse to navigate the subtle structures in big, unstructured latent spaces. Latency also makes it difficult to have a more rhythmic approach, as a performer has to slow down to a more delayed feedback time with the instrument.*

Another participant, whose primary musical expertise was drumming, also noted latency (as well as reverb) as being a slight problem while testing the prototype. It was only addressed by two of the participants, perhaps because their

[2]Since Pure Data was not able to infer from more than one model at a time, they had to be manually switched. Participants were therefore not able to use the models in combination with one another nor switch between the models by themselves.

musical backgrounds, especially in the case of drumming, made them more perceptible to it.

4.1.3 Functionality

Participants were lastly asked to imagine what kinds of settings they might see the controller being suitable for, or even just the RAVE-models by themselves. Many was of the conviction that the incorporation of movement could make it a great asset for performances, either as part of an installation or experimental live music. One participant could also envision it as a way for story tellers or actors in a theater play to control music to back up their storytelling. A few noted the potential of the models for creating interesting and varying timbres made them (along with the controller) suitable as tools for music production, especially when considering the fluency with which they make one able to switch direction in the sound:

> *I could imagine it being a part of most home studios etc, as some sort of MIDI-controller that has the benefit of fluent x and y axis movements.*
>
> *It can be used to create the foundation to a song and to add dynamics to different sounds.*
>
> *Yes, [I could imagine using the models] in a performance setting because they are quite expressive and offer a range of timbres.*

One of the participants who already had experience with using RAVE for creating music was of the opinion that RAVE-models in general tended to be more proficient for re-synthesis (i.e. timbre or style transfer) when provided with existing material, rather than generating the sound by themselves. This was attributed to a dissatisfaction with the obscurity of how the latent spaces of the models were sampled.

Chapter 5

Discussion

This project has for the most part been dedicated to setting up for the training and inference of the machine learning models and working through the challenges it imposed along the way. For further development, more time could have been spent on the prototype, and what kinds of interaction with the RAVE-models it could possibly afford that might be intuitive, engaging and useful for musicians or producers.

The evaluation of the prototype roughly followed the line of thought presented in works such as O'Modhrain's framework [24] in terms of what might constitute a reasonable way of evaluating usability of a digital musical instrument. Central here is that what a proper evaluation entails depends on perspective, since composers/performers, audiences and designers (the primary "stakeholders") will usually attribute different factors that make for a "successful" instrument. Most emphasis must however be placed with the composer/performer, which is why they were sought out for the usability test; additionally, a few participants had experience with designing instruments themselves. Even though more participants would be needed for a more comprehensive conclusion, some important points have been collected and could be considered for directions of further development for the prototype and the use of RAVE-models for musical instruments.

5.1 Usefulness of the prototype

As pointed out by some participants, in order to produce sound with the prototype, no "leverage" from existing musical instruments are needed — one does not need to be a skilled player of existing instruments to be able to grasp how to generate sound with it. As targeted for a composer or producer, it might be a more expressive alternative to a typical MIDI-controller, as one pointed out, that could offer a more fluent means of control.

Three participants spoke of the possibility of using the prototype to mimic

drumming gestures as well as producing percussive sounds with the models. Had the prototype been targeted for drummers, modifying the controller itself could be done so straightforwardly, by replacing the gyroscope with an IMU, that would also allow for measuring the acceleration of the hand, rather than simply the rotational speed. A more difficult issue would be reducing the latency, which should be kept at a minimum, as instantaneous timing is more critical within this scenario. Possibly it could be achieved through faster readings from the ESP-32. For model inferring, the delay in Pure Data could not be reduced to less than 15-10 milliseconds, where audio dropouts would start occurring.

5.2 Unconditional audio generation

RAVE was chosen as it is the current state of the art in terms of "low-powered" real time audio generation that can be inferred on many devices. Another advantage is that it does not target a very specific kind of sound, as otherwise seen with models presented in chapter 2, such as the granular GANSynth-model for the *AI-Terity* instrument or the impact sounds of Neurorack, making it a more flexible means of synthesis. The unconditional approach to sound generation, as part of a more (but not fully) unsupervised approach to the learning, does have its trade-offs, as touched upon in earlier chapters; namely, the lack of control over high-level parameters. One one hand, the idea of developing models that are able to sample the latent space without intervention is intriguing, and allows for many kinds of interesting alternative directions to arise when e.g. the model is able to generate polyphonic sounds instead of being conditioned to a certain fundamental frequency. This approach also allows for the algorithm to make errors that might turn out to be unique and desirable when occurring in the right context — the "happy accidents" that can initially result from constraints or limitations of new technologies.

5.2.1 Intuition of the interaction

On the other hand, the lack of overview or direct casualty of unconditional sound generation can also be a point of confusion, or simply viewed as ill-suited for many tasks related to sound synthesis, where comprehensive means of control is often desirable. When playing music, having a reliable way to adjust the amplitude of a specific sound or being able to recreate it in another pitch are fundamental and often presented in a more or less intuitive way, though it should be left to the musician to develop the virtuosity to be able to bring forth change in increasingly expressive ways. According to O'Modhrain, causalty is important for both audiences and performers, but most fundamental, from the perspective of a performer, is reliability [24].

Casualty was in part addressed through the mappings created in the scripts for the models in order to evaluate the prototype. Amplitude was found to be correlated with a (numerically) higher activation of a given latent parameter, meaning intuitive ways of control, such as by applying more pressure to the FSR could be used to increase it. Changes in fundamental frequency was harder to achieve, happening by chance when activating one or more additional latent parameters while one was already active. This thus became a more "hidden" modality of the instrument, that required more subtlety to achieve.

Some of the questions the participants were asked was aimed at getting them to consider whether they felt in control of pitch and timbre, as well as amplitude of the sound. Further evaluation could entail asking them more specifically about how they felt the indirectness of interaction with these parameters affected the way they engaged with the sound — whether they could still find the interaction intuitive and engaging, and whether they saw it as a shortcoming or not being unable to directly parameterize the sounds they were creating. Their responses gave some idea, but an approach to evaluating that would shed more light on this aspect would be needed to draw a more certain conclusion.

Further work

A way to gain another perspective on the interaction could have been to have found a way for participants to try a generative modelling scheme other than the unconditional generation method that RAVE offers. A DDSP-model could possibly have been included as an alternative to have participants try and interact with a model that is capable of merging between timbres and is also built to consider high-level parameters (i.e. fundamental frequency, amplitude, harmonics) as the primary way of creating changes to the sound. The conditioning of DDSP might be considered as either more structured or more restricting, but would allow participants to reflect on whether directly being able to change the sound parameters would improve their engagement with the way of interacting that is afforded by the prototype.

Lastly, there have also been made more recent efforts towards providing better control of the latent parameters of RAVE-models by mapping them to a set of high-level parameters — allowing continous control over the chosen parameters, which is also referred to as *attribute transfer*. An extension by Devis et al. [8] introduces a set of more implicit parameters (booming, sharpness, centroid, bandwidth and RMS) by re-organizing the encoder's representation during the first training stage. The models are then capable of performing either timbre or attribute transfer, giving the user access to a set of alternative parameters that might make very specified changes in loudness and timbre. Unlike DDSP, however, this extension does not offer any explicit control of the pitch (or indeed, timbre and loudness by themselves).

Chapter 6

Conclusion

This project has been concerned with analyzing what approaches there has been to audio generation with machine learning models, specifically deep learning models that might be able to produce sounds that are hard to achieve with other forms of synthesis. A certain prospect of this form of sound synthesis, the merging of and interpolation between learned representations of real sounds, has been a specific point of focus, along with inference in more computationally efficient ways, as to how sounds might be recreated in real time on less powerful devices. An overview has been given of how structures for such models might look, and how they mitigate problems that are tied to the different approaches of learning high resolution sound representations.

Considering RAVE, which is one specific state of the art approach that targets the intersection between sound of high quality, computational efficiency and a broad range of applications (both in terms possibility of sampling of its latent space and deployment in several environments), a scheme for musical interaction was developed. By leveraging its architecture and approach to learning, models based on musical instruments could be trained for the purpose of merging and interpolating between several sound directions at once. The models were deployed to a powerful, yet small and low-powered device, the NVIDIA Jetson AGX Orin, using extensions to make real time inference possible.

A way of synthesizing sound using the models was developed. By coupling the Orin with an ESP-32 Thing Plus, an embeddable device that could be fitted onto the hand of a potential user, different modes of interaction was considered to gain control of the sound. A prototype of a small controller was built that would give musicians the ability to activate the latent parameters of the models to create sound. The controller would detect rotation and tilting, pressure applied in a grasping motion and also allow for using presence of a magnet as ways of moving the sound in different directions. Evaluation was carried out by employing users from a target group — people with experience in playing and producing music —

to test the usability of the interaction with the models and potential usage of this particular method of synthesis. The responses gave several points of reflection for how further development might be carried out. They also yielded some positive and promising remarks from the test participants on both the initial impression of the prototype and the idea of using the latent spaces of a generative model as a form of sound synthesis.

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[2] Arun. "An Introduction to TinyML". In: (2020). URL: https://towardsdatascience.com/an-introduction-to-tinyml-4617f314aa79.

[3] John Brooke. "Sus: a "quick and dirty'usability". In: *Usability evaluation in industry* 189.3 (1996), pp. 189–194.

[4] Antoine Caillon and Philippe Esling. "RAVE: A variational autoencoder for fast and high-quality neural audio synthesis". In: *arXiv preprint arXiv:2111.05011* (2021).

[5] Antoine Caillon and Philippe Esling. "Streamable neural audio synthesis with non-causal convolutions". In: *arXiv preprint arXiv:2204.07064* (2022).

[6] Devin Coldewey. "Try 'Riffusion,' an AI model that composes music by visualizing it". In: (2022). URL: https://web.archive.org/web/20221216000233/https://techcrunch.com/2022/12/15/try-riffusion-an-ai-model-that-composes-music-by-visualizing-it/.

[7] Ninon Devis and Philippe Esling. "Neurorack: deep audio learning in hardware synthesizers". In: *EPFL PhD Seminar "Human factors in Digital Humanities"*. POST_TALK. 2021.

[8] Ninon Devis et al. "Continuous descriptor-based control for deep audio synthesis". In: *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5.

[9] Chris Donahue, Julian McAuley, and Miller Puckette. "Adversarial audio synthesis". In: *arXiv preprint arXiv:1802.04208* (2018).

[10] Vincent Dumoulin et al. "Adversarially learned inference". In: *arXiv preprint arXiv:1606.00704* (2016).

[11] Jesse Engel et al. "DDSP: Differentiable digital signal processing". In: *arXiv preprint arXiv:2001.04643* (2020).

[12] Jesse Engel et al. “Gansynth: Adversarial neural audio synthesis”. In: *arXiv preprint arXiv:1902.08710* (2019).

[13] Jesse Engel et al. *Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders*. 2017. eprint: arXiv:1704.01279.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[15] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[16] Intelligent Instruments Lab. *rave-models (Revision ad15daf)*. 2023. DOI: 10.57967/hf/1235. URL: https://huggingface.co/Intelligent-Instruments-Lab/rave-models.

[17] Théo Jourdan and Baptiste Caramiaux. “Machine Learning for Musical Expression: A Systematic Literature Review”. In: (2023).

[18] Chris Kiefer. “Towards lightweight architectures for embedded machine learning in musical instruments”. In: (2022).

[19] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[20] G. Malloy C. og Tzanetakis. *SASS-E: The Steelpan Audio Sample Set for Evaluation*. Zenodo, 2023. eprint: doi:10.5281/zenodo.7803316.

[21] Soroush Mehri et al. “SampleRNN: An unconditional end-to-end neural audio generation model”. In: *arXiv preprint arXiv:1612.07837* (2016).

[22] Sparsh Mittal, Poonam Rajput, and Sreenivas Subramoney. “A survey of deep learning on CPUs: opportunities and co-optimizations”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.10 (2021), pp. 5095–5115.

[23] *Neurorack - Acids IRCAM*. 2023. URL: https://acids.ircam.fr/neurorack/.

[24] Sile O’modhrain. “A framework for the evaluation of digital musical instruments”. In: *Computer Music Journal* 35.1 (2011), pp. 28–42.

[25] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

[26] Teresa Pelinski et al. “Pipeline for recording datasets and running neural networks on the Bela embedded hardware platform”. In: *arXiv preprint arXiv:2306.11389* (2023).

[27] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[28] Koray Tahiroğlu, Miranda Kastemaa, and Oskar Koli. "AI-terity 2.0: An Autonomous NIME Featuring GANSpaceSynth Deep Learning Model". In: *NIME 2021*. PubPub. 2021.

[29] Ilya Tolstikhin et al. "Wasserstein auto-encoders". In: *arXiv preprint arXiv:1711.01558* (2017).

[30] Yuhuai Wu et al. "On the quantitative analysis of decoder-based generative models". In: *arXiv preprint arXiv:1611.04273* (2016).

[31] Geng Yang et al. "Multi-band melgan: Faster waveform generation for high-quality text-to-speech". In: *2021 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2021, pp. 492–498.

Appendix A

Evaluation

A.1 Questions: Test of prototype

1. Did you find it easy to change the sounds in the directions you wanted?
2. How well did you feel you were able to merge or interpolate between different instruments/sounds?
3. To what extend did you feel the sound was influenced by your movements?
4. How well do you rate your ability to control pitch and timbre of the sound?
5. How well do you rate your ability to control the loudness?

A.2 Questions: Thoughts on the concept behind the prototype

- Did you like the sound (in terms of quality, timbre, etc) ? Are there any other kinds of sounds that you think might be interesting to be able to synthesize this way?
- Do you think this instrument/method of synthesis offers a unique kind of sound? (As opposed to alternatives like sampling, FM, granular synthesis, etc)
- Was it interesting to interact with the model with the kinds of sensors chosen?
- Can you imagine a context where you might want to use a machine learning model that behaves like these do?

A.3 Description of usability test

The purpose of this project has been to train lightweight machine learning models that might provide the sound for a musical instrument or a sound production tool that can operate in real time. A strength of using machine learning for audio synthesis is that these models are able to switch between and merge different "directions" of sound, such as interpolating between the sound of a flute and the sound of a string. Through training, the models determine these directions and store them as so-called latent parameters.

For this experiment, the latent parameters of three models trained on different datasets can be activated with the sensors of the controller. The models are presented one at a time, where you get to play the "instrument" and change the sound by either tilting and rotating your hand, press down on the sensor in your palm or hold a magnet close to the metal chip on the board on the back of your hand. Once you've gotten to test the prototype, you'll get asked about your impression on the idea.

The test and the survey should take about 20-25 minutes to complete, and your responses will be saved anonymously. No data is collected while testing the prototype.

A.3.1 Questions: Background

1. Do you play any instruments? If yes, for how many years have you played music?
2. Do you make your own music/sound productions? If yes, for how many years?
3. Have you worked with music/sound professionally?