Zürich University of Arts
Master Composition and Theory - Sound Design

A Bottleneck of Opportunity

Designing an interface for the experimental exploration of RAVE models

by Floris Demandt
May 2024

Supervised by Dr. phil. Daniel Hug

ACKNOWLEDGEMENT

I would like to express my gratefulness to Dr. phil. Daniel Hug for his guidance and support throughout the process of writing this thesis. His advice and expertise have been instrumental in helping me focus on the core aspects of my research. His encouragement and constructive feedback have significantly enhanced the quality of my work and I am very grateful for the time and effort he dedicated to mentoring me.

I also would like to express my thankfulness to Cedric Spindler, who supported me in developing my interface, helping me with programming and was always a great sparing partner for general discussions about artificial intelligence.

I would like to thank Dr. Daniel Bisig (Research Assistant at the ICST) and Florian Brugisser (Research Assistant at the Immersive Arts Space) for giving me access to a powerful computing structure to train RAVE models. Without their help, training models would have been much more difficult and time intensive.

I want to thank Eric Larrieux, who was always very interested in this thesis and provided valuable insights into existing RAVE research and applications, giving me a better understanding of the topic.

ABSTRACT

The vast improvements of generative audio AIs have created powerful new tools to generate audio content for a wide target group. Popular audio generation models, like Suno and Udio, enable users to generate high-quality music by writing a text prompt. This makes them very accessible and easy to use for the average person, but for audio professionals such as musicians and sound designers, they lack expressivity and adjustability because of their text prompt interface and non-real-time audio generation. Novel approaches to AI audio generation like IRCAM RAVE offer solutions to these problems by generating audio in real time and abandoning the need for text prompting and offering direct and precise control of the generation through the model's latent variables. This inevitably raises the question of how an interface for a real-time-generative-audio-model that abandons the need for text prompting could be designed and if this method offers potential for new sound design and music tools.

This thesis explores one of various ways of designing an interface for a real-time-generative-audio-model by giving the user methods to experimentally explore RAVE models with a variety of interaction methods. These interaction methods range from simple sliders to 3D physical models, with each method exploring a distinct way to interact and therefore generate audio content. During an expert review, the interface was evaluated to gather feedback on its perception and usability, as well as the interaction methods used. Using this expert review, the interface was enhanced with new interaction methods. This thesis concludes with an outlook for future generative audio models and what audio professionals may expect to incorporate them into their workflows.

1. MOTIVATION

Throughout my master's degree, I was always interested in projects that focused on creating interactive experiences. Be it in games, installations, or sound-design tools, I enjoy creating interactive playgrounds that reward experimentation through a balance of restriction and freedom, while providing a fun interaction experience. It is especially rewarding when these interactive experiences allow for freedom of exploration and play, resulting in unrestrictive environments for users.

The vast improvements of AI, especially generative AI, in the last years sparked my interest in potential use cases for sound-designers and musicians. Especially because I have incorporated various tools into my workflow over the years that allow a playful and explorative approach to sound design whenever I lack inspiration or am still in search of a sound aesthetic. I asked myself how AI could fit into this existing palette of explorative tools. There are many AI audio tools that offer a solution for technical problems such as noise reduction or voice enhancement AIs (i.e., IZotope's newly announced RX11), but there are yet tools to be released that help with the actual creative sound design tasks sound designers face. Recent research shows that there is a great demand from sound designers for AI tools that support the creative process (Kamath et al., 2024).

While AI models like Suno or the recent Udio enable high-quality generation of music with prompt input, the same has yet to happen for sound effects and, more importantly, a useful integration into an audio professional's workflow is yet to come. If models like Suno and Udio are someday capable of producing high quality sound effects, they still lack sufficient features that I and other sound-designers would wish for before integrating them into a workflow:

1. While this prompt-based approach is a significant technological achievement, personally I am not particularly interested in it for my workflow because creating music or sound with a text prompt feels unintuitive and the output somewhat arbitrary to me. While I find it exciting to listen to prompt created music, it almost always fails to produce my exact desired output. And on top of that, it is not possible to make fine adjustments to the output afterwards, since the only way to change it is with a new text prompt, that the model might interpret completely differently to what I might desire.
2. Suno or Udio are not designed to respond to real-time adjustments, which I find a significant disadvantage if you want to make a tool to play with and allow for quick iterations. Recent research about creative AI tools for sound designers (Kamath et al., 2024) shows that many sound designers are excited about AI tools to help with such a process.

3. They are closed-source. This means that it is not possible to get an understanding of how they exactly work and, more importantly, you cannot train your own sound models. This is a significant disadvantage if sound designers want to create generative AI models with their own source material and potentially limits use cases because sound designers would always be reliant on models that might not perfectly fit their requirements.

Because of these disadvantages, I got interested in one of the recent developments in AI audio generation RAVE (Real-Time-Variational-Autoencoder) by IRCAM. While not always successful, RAVE tries to tackle the listed critique points above, such as usability, real-time usage, possibility of a wide range of applications within audio (e.g., live performance, sound-design, music, production, audio plugins) and is not prompt based. Although RAVE is said to enable musicians and sound-designers to create their own models, for people with no prior experience in coding (In my experience, this applies to the majority of musicians and sound-designers) this poses significant challenges.

Because RAVE allows for a new approach of interacting with a model, namely not prompt-based but by either inferring[1] it with audio material or influencing the generation of audio by directly interacting the models parameters, it inevitably raises the question of how an interface for a real-time-generative-audio-model that abandons the need for text prompting could be designed and if this method offers potential for new sound design and music tools. This is the main research question of this thesis. Designing an interface for RAVE that encourages experimental exploration of models by providing the users with multiple versatile interaction methods while providing a fun experience with the freedom to combine interaction methods as desired. Because of the unsupervised model architecture of RAVE and the resulting non labeled model parameters[2], it is important to mention that RAVE models encourage explorative sound design approach rather than a very goal orientated one. While sound designers oftentimes have a very goal orientated process a certain amount of unpredictability and "Cinematic effect over accuracy" is also something that is desired (Kamath et al., 2024).

[1] Inferring (n. inference) is "…the process of running data points into a machine learning model to calculate an output…"(Google, 2024). For example, entering a prompt into ChatGPT is inference.

[2] For readers unaware about RAVE's architecture this is difficult to understand at this moment. I will explain the characteristics of unsupervised networks shortly.

By providing multiple interaction methods, my intention is to increase the chances to fully explore and exhaust the capabilities of every model. It is equally important to emphasize that I designed the interface with users in mind that have never heard of RAVE before and are not familiar with generative audio tools but are sound-designers or musicians just like me with an interest in adding another tool to their sound and music vocabulary.

Exploiting the full capabilities of RAVE also includes training a model. This has several advantages, because you are not dependent on third party models, you have control over the training process, and you can get a grasp about how the model uses your provided sounds and maybe even recognize characteristics from them. Therefore, besides my main research question, I provided a basic model training guide on the usage of IRCAM RAVE for people with no coding experience. This guide is published as a website that explains the basics. It starts with building a coding environment and later diving into training models, while providing as much information as possible about important topics along the way. I was starting to write the guide in March 2024 when there was nothing comparable available, but it should be mentioned that since then IRCAM released a video about the training process.

2. RAVE’S ARCHITECTURE

It is important to me to give an in-depth explanation about the inner workings of RAVE because they have direct implications for the later design process of the interaction methods and the interface. Therefore, in the following, I will describe the architecture of a RAVE model and lay a foundation for neural networks and discuss the two types of neural networks that RAVE is using.

2.1 TRAINING PHASE 1: VARIATIONAL AUTOENCODERS

In chapter one, I briefly mentioned that RAVE is a so called Variational Autoencoder or a VAE. A VAE is a specific method of structuring a neural network. It generates new content from the data it was given during the training stage. It is important to me that the readers understand this concept in more depth, since my interface will sit inside the architecture of this VAE and directly influence the audio generation of the model.

I will start by breaking down the basic principles of neural networks and machine learning and set them in relation to autoencoders and then later variational autoencoders. After that, I will move on to the second training phase which uses Generative Adversarial Networks (also known as GANs).

2.1.1 NEURAL NETWORKS: THE FOUNDATION

Neural networks are the fundamental structure for every machine learning project. Neural networks simulate the biological learning process, similar to our human nervous system, by creating a network of connected **neurons** (Aggarwal, 2018). Each connection has a **weight,** resulting in weighted connections between the neurons. These weights (Dotted green arrows Figure 1) correspond to the connection strength between the neurons. The neurons hold values that they pass on to the following neurons.

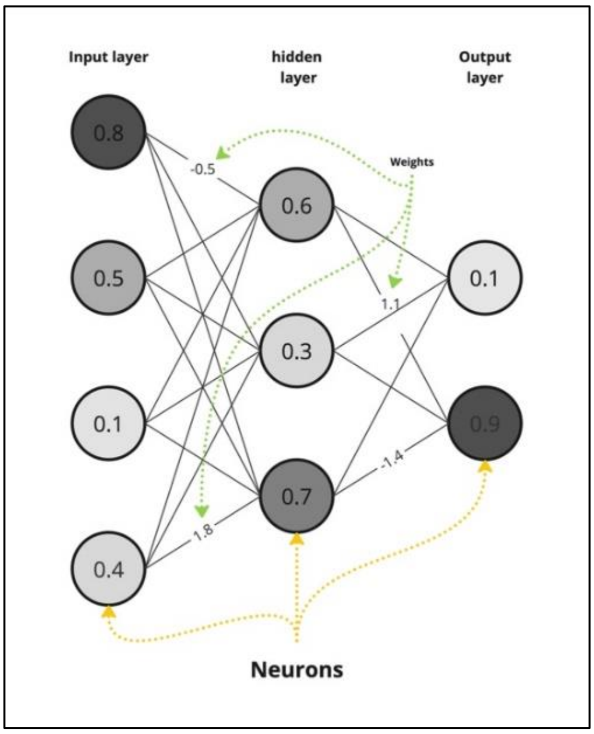


FIGURE 1 (CREATED BY THE AUTHOR)

Depending on the strength of the weight, the following neuron is more likely to get activated. The weight can be described as a factor (a real decimal number). When a neuron gets a signal from a previous neuron, the sum of all the signals then decides if the neuron gets activated (Aggarwal, 2018). This is achieved with an **activation function**. Using Figure 2 (X-axis: value from previous neuron, Y-axis: activation threshold) as our activation function, the neuron is only activated when the value is greater than zero[3].

[3] It must be mentioned that in practice there is also an additional value called the bias unit. The bias unit can be thought of as a factor sitting in front of the value inside of the neuron. This bias unit, as the name suggests, can be used to provide a bias towards a certain value. For simplicity I will not talk about this further.

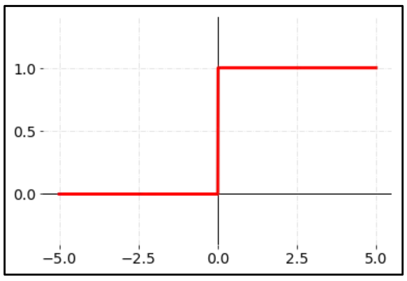


FIGURE 2 A BINARY ACTIVATION FUNCTION (CODECADEMY, 2023)

2.1.2 THE BASICS OF TRAINING A NEURAL NETWORK

Suppose we want to train a network that can distinguish between photos of dogs and cats. For training or learning such a neural network, the previously mentioned **weights are critical** because the neural network needs to adjust these weights during the training process to produce the desired output. For example, it is obviously incorrect if our imagined cat-dog-distinguishing-network receives an image of a dog but determines that it has detected a cat. Consequently, the network needs to acquire knowledge of that. The network does this through what is called **backpropagation**. Backpropagation steps through the entire network backwards (Aggarwal, 2018) to identify neurons and their weights that are most responsible for the error and therefore need to be adjusted to reduce it.

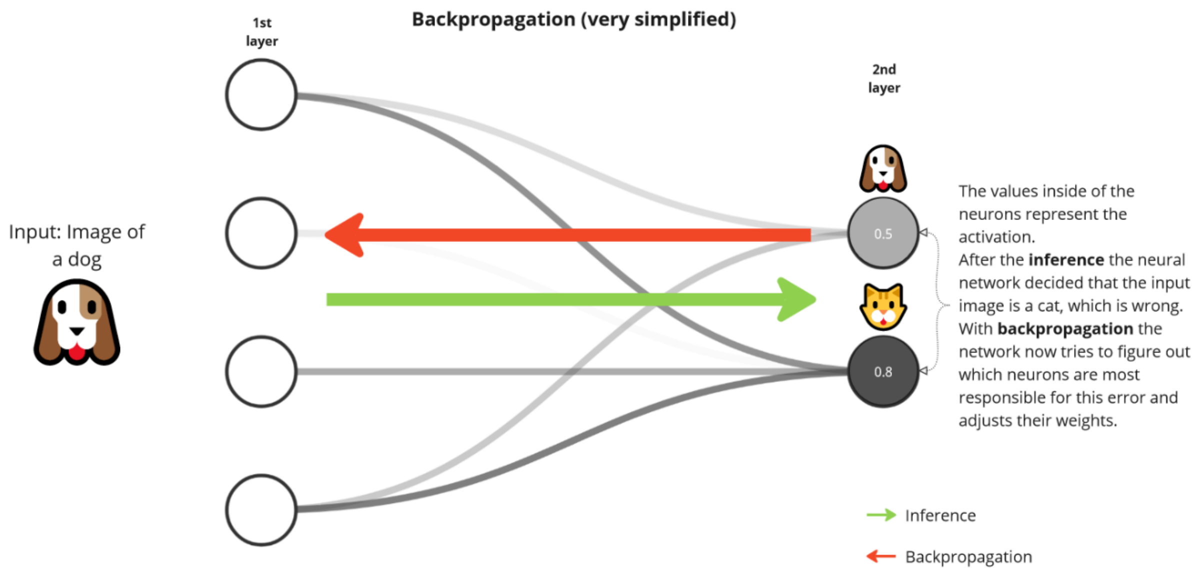

Backpropagation (very simplified)
1st layer
2nd layer
Input: Image of a dog
0.5
0.8
The values inside of the neurons represent the activation.
After the inference the neural network decided that the input image is a cat, which is wrong.
With backpropagation the network now tries to figure out which neurons are most responsible for this error and adjusts their weights.
Inference
Backpropagation

FIGURE 3: A VERY SIMPLIFIED ILLUSTRATION OF BACKPROPAGATION (ILLUSTRATION BY THE AUTHOR).

This is achieved by comparing the input and output and inputting this into a loss function. The loss function estimates the contribution of each weight to the overall error of the model (Briot et al., 2020). In other words, the loss function tries to move the weights of the current answer into the direction of minimum loss (Figure 4 on the left). If the loss function has reached the global cost minimum, this is then the best possible solution. These functions typically look much more complex and have multiple minima (Figure 4 on the right). For this reason, finding the global cost minimum of a loss function is an intensive computing problem to solve.

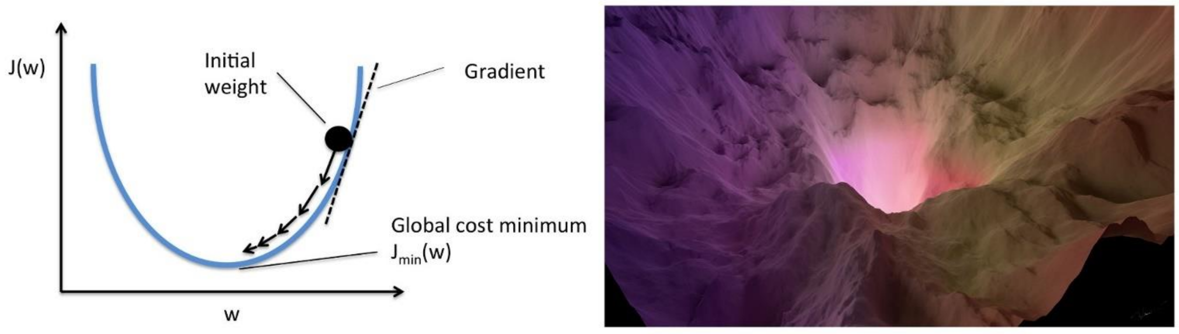


FIGURE 4 LOSS FUNCTION SIMPLIFIED ON THE LEFT. ON THE RIGHT IS A MORE COMPLEX LOSS FUNCTION (IDEAMI, N.D.; MURIUKI, 2018)

2.1.3 AUTOENCODER

Now that we have a basic grasp of neural networks and their training process, let us proceed to the topic of the Autoencoder. This may be quite a leap, but I will try my best to explain it.

Breaking down Figure 5 starting from the left, we have the Input. In Figure 5, this input is data from a popular dataset called MNIST, which is just a large collection of handwritten numbers. This data is put into an encoder. The **encoder** is a converging neural network, meaning that the layers of neurons reduce in number from input to output. In other words, it compresses the input information into fewer dimensions. The point where this reduction of dimensions is the largest is called the **bottleneck**. Following the bottleneck is the **decoder**. The decoder is a diverging neural network, meaning opposite to the encoder, the dimensions get increased in dimensions from input to output. The decoder uses the compressed representation from the bottleneck to expand it to the number of dimensions of the original input data (Briot et al., 2020). After training an autoencoder, the input and output are ideally identical, but in fact autoencoders are inherently lossy (Aggarwal, 2018) and lose some information from the input once decoded.

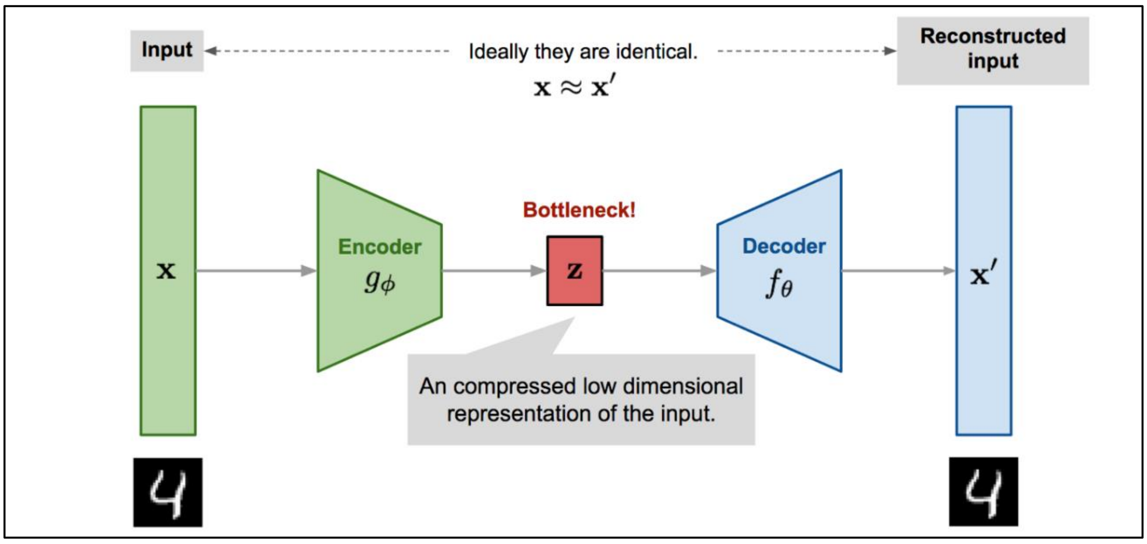

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

FIGURE 5 (ROCCA, 2019)

EXTRACTING HIGH LEVEL FEATURES

What an autoencoder does, besides recreating the original data, is to extract **high-level features** (Briot et al., 2020) from the data and reduce these features to as few as possible (Bottleneck). The autoencoder's decoder then uses only these few features to replicate the input data. It is a form of data compression of the original data, but a lossy one. A finished autoencoder can be used to extract features from an input that the autoencoder has never seen during training. For example, it can be used to decide what numbers are in an image or if an image contains a dog or a cat. The following chapter will explain this in more detail.

2.1.4 LATENT SPACE

The previously mentioned **features** that the model extracts are ordered by putting the data into a **latent space**. A latent space is a vector space that characterizes the input data with a reduction in dimensionality (Liu et al., 2019) which is the bottleneck I described above. This is especially important for this thesis because, when interacting with a RAVE audio model, we are in fact interacting with the latent space of the model. These features are not defined by the user beforehand. Instead, the autoencoder creates these features on its own. Because of this, the training process is called unsupervised learning. Training a RAVE model is also unsupervised learning, meaning that high-level features, for example loudness, centroid, pitch etc. cannot be defined beforehand. The network extracts these by itself. Therefore, we might not even be able to decipher what the high-level features exactly mean, especially with a finished RAVE model that can have dozens of dimensions/features.

Figure 6 shows the latent space of an autoencoder that is trained on the MNIST dataset which holds about 60,000 handwritten numbers. As a result of the training process, the autoencoder ordered the numbers into little islands of similar numbers because they share similar features. In the case of MNIST it is the composition of the pixel data that represents the numbers. Usually, the longer the model was trained, the more precise that vector space is ordered.

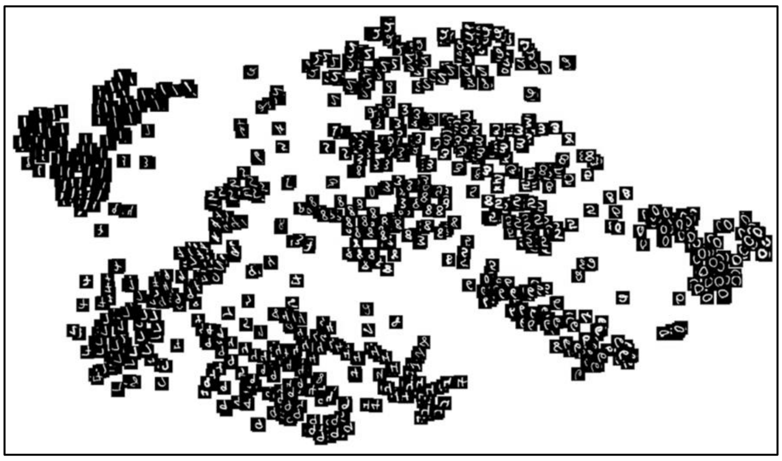


FIGURE 6: LATENT SPACE OF HANDWRITTEN NUMBERS (DESPOIS, 2017).

Another use case for autoencoder besides extracting features, as described in chapter 2.3.1, is content creation by interacting with the latent variables of the bottleneck directly. But there are a couple of steps before this interaction produces meaningful content[4] for every point in the latent space. If we scroll through the latent space of Figure 6 space with an imaginary finger that is our sampling point, we change the values of the high-level features that describe this latent space. These features can be seen as coordinates, also known as **latent variables**, for our sample point. Exactly like we would use X and Y coordinates to move through a 2D space. Examining Figure 7, the latent space on the left is not ordered in a particularly meaningful way, because points that are encoded close to each other do not have a lot in common.

[4] Meaningful refers to two things. Firstly, do close encoded points in the latent space have similar features? And secondly, the created content will not be meaningful if we have white spaces in the latent space that do not represent any of the data from the dataset.

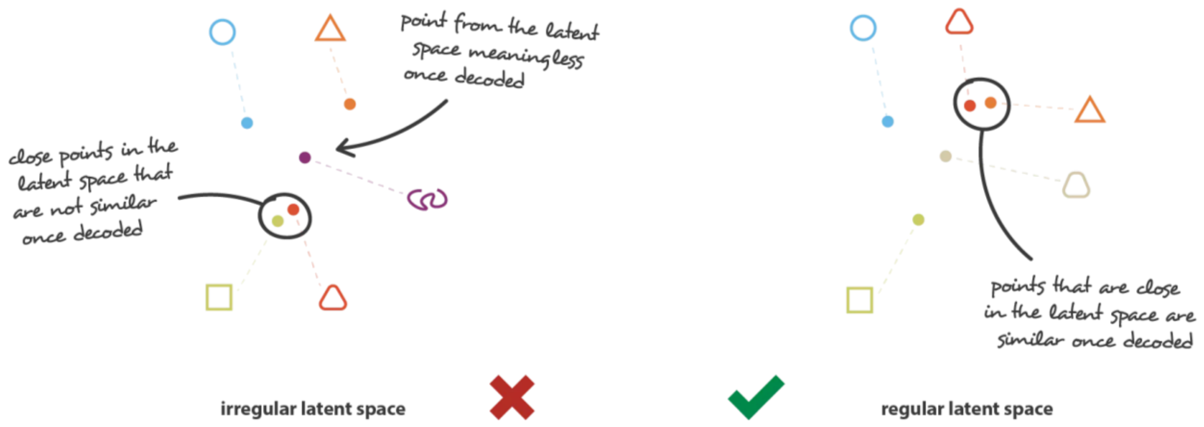

point from the latent space meaningless once decoded
close points in the latent space that are not similar once decoded
points that are close in the latent space are similar once decoded
irregular latent space
regular latent space

FIGURE 7 (ROCCA, 2019)

Whereas on the right, closely encoded objects share more similar features. For example, the triangle with rounded corners is close to the triangle with sharp corners. This is called a **regular** latent space. A regular latent space is one step closer to meaningful content creation. Because there still is no useful information encoded in the white space of the latent space, the autoencoder cannot smoothly interpolate between points at this moment. For example, between the triangle with rounded corners and the triangle with sharp corners.

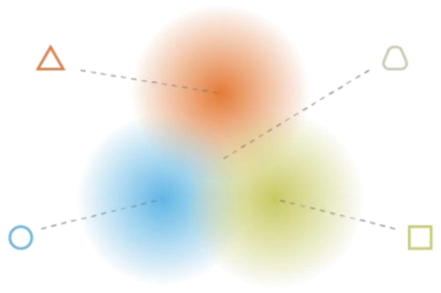


FIGURE 8 (ROCCA, 2019)

What is desired is something like shown in Figure 8, where there are no white spaces between the objects and data points with similar features are encoded close to each other. Figure 9 shows how this looks for the MNIS dataset. It also clearly demonstrates that in this case the VAE is not ordering the numbers according to their mathematical value, but rather regarding their similar pixel values.

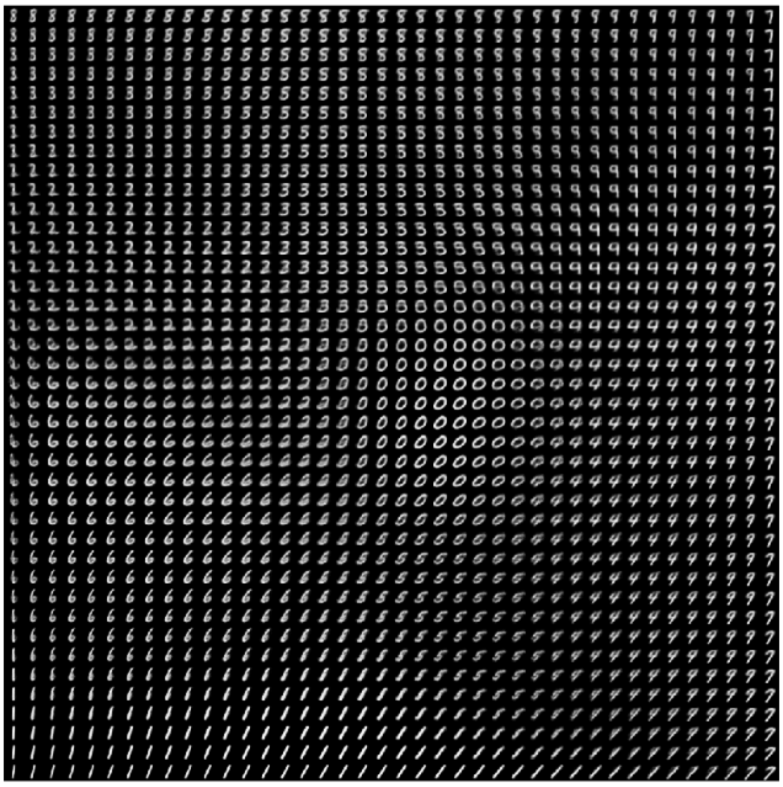


FIGURE 9 (JORDAN, 2019)

This is achieved by not encoding the input data into a single point but to a probability distribution. During the training it is then enforced that sampling from within the distribution creates a similar output (Figure 10). Often a gaussian distribution is used but in theory it can be any distribution[5]. If we now scroll through this space with our imaginary finger, the output is a meaningful smooth interpolation (Briot et al., 2020).

All the described principles of Variational Autoencoders also apply to RAVE, meaning similar audio data, according to the features the VAE decided to use, is encoded close to each other in the latent space. For example, if RAVE is trained on drum samples, it is likely that snare sounds are encoded

[5] I recommend Joseph Rocca's article **Understanding Variational Autoencoders (VAEs)** to get a deeper insight into the concept of encoding data into a latent space with probability distributions.

close to each other and Cymbal sounds are encoded at a different location. Even so, it is then possible to interpolate smoothly between snare and cymbal sounds.

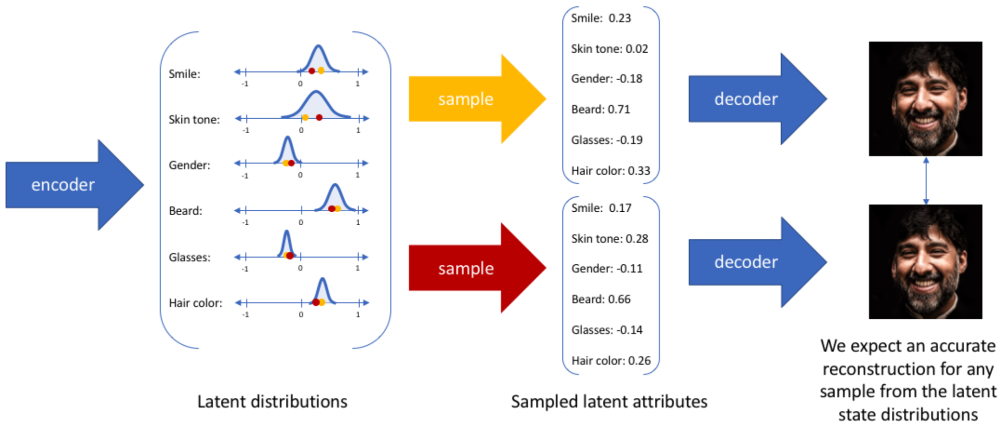


FIGURE 10 SAMPLING FROM A VAE CREATED WITH IMAGE DATA (JORDAN, 2019)

2.2 Training Phase 2: Generative Adversarial Network

In a second step, IRCAM RAVE uses the principles of **Generative Adversarial Networks**, also known as GANs, to ensure the quality of the sounds. A GAN can be described as a game between two neural networks: a generator network and a discriminator network. With RAVE, the generator network is the VAE from stage 1 after it has finished the training. The discriminator (discrimination in the sense distinction) takes two inputs. Firstly, generated audio from the generator and secondly a sample from the original dataset. The discriminator's job is to distinguish/discriminate between the generated and original data. The generator tries to create samples as realistic as possible to fool the discriminator. Over time, the generator ideally becomes so good that the discriminator cannot distinguish between real or "fake" data (Aggarwal, 2018).

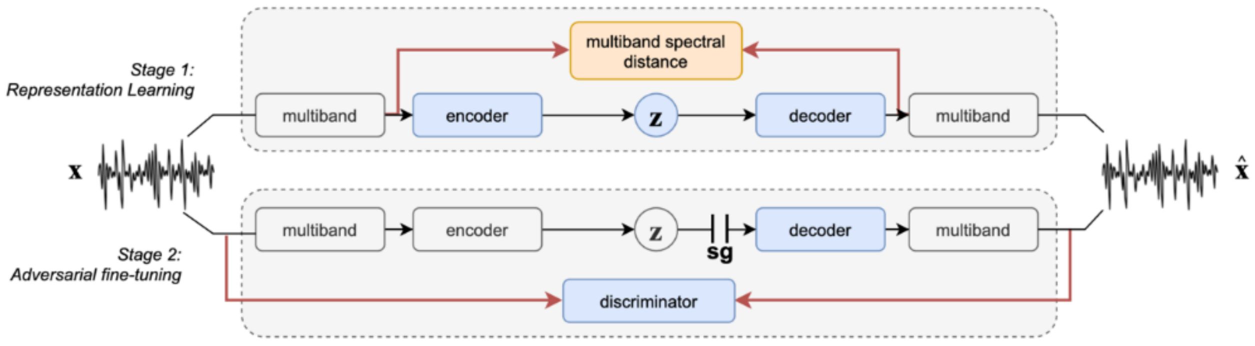


FIGURE 11 ARCHITECTURE OF RAVE (CAILLON & ESLING, 2021)

Figure 11 shows the two stages of training a RAVE model. Stage 2 only starts after stage 1 (The training of the VAE) is completed. The illustration also shows **multiband** blocks before the encoder and after the decoder. The multiband block before the encoder represents a **multiband decomposition** into sixteen audio bands that make the model more performant in terms of synthesis speed. Additionally, Figure 11 also shows a **multiband spectral distance** block. This multiband spectral distance block takes samples before the encoder and after the decoder and compares the amplitude of their Short-Term-Fourier Transforms. The less distance between the Short-Term-Fourier Transform, the better the VAE performs, because this means the generated audio sounds like the original audio data. The spectral distance is measured with a Short-Term-Fourier because this disregards the phase of the two signals and therefore, only important perceptual features are trained (Caillon & Esling, 2021; Engel et al., 2020).

2.3 Conclusion

- Neural networks work by connecting layers of neurons with weights.
- Training the neural network adjusts these weights using a loss function.

RAVE training Phase 1:

1. RAVE uses a Variational Autoencoder to compress the audio data of the given dataset to as few dimensions/high-level features using the encoder.
2. By learning distributions instead of single points and regularization, we get a latent space where you can scroll through and get meaningful data.
3. The decoder learns to recreate the original data with the compressed latent space.

RAVE training phase 2:

4. Using an adversarial stage, the model is fine tuned for perceived audio quality.

Now that a basic understanding of RAVE is established, I will move on to training a model.

3. TRAINING RAVE MODELS

IRCAM RAVE already provides trained models which are ready to use. But I wanted to train my own models for a couple of reasons:

1. The training process has multiple configurations which influence the perceived quality of the model and therefore I wanted to have control over that.
2. I was interested to discover if diverse content (ambiences, impacts, explosions, foley etc.), like my personal sound library, would still create usable models.
3. Without having access to the original audio in the dataset, it is not possible to evaluate if the model accurately recreates the original data.

When starting to train my first models, the topic of machine learning was just a buzzword for me, and I had no prior experience with it other than watching an occasional YouTube video. Therefore, the learning curve was steep, and I had to start over a couple of times. There is a brief guide on the GitHub page of RAVE and a quite active RAVE discord channel that cleared some questions, but it still was quite difficult to finish my first model.

There are two distinct ways to train RAVE.

1. Locally on a Windows PC with a Nvidia GPU that needs to support the Compute Unified Device Architecture (CUDA). CUDA is an architecture provided by Nvidia that enables machine learning on their desktop GPUs.

or

2. Using cloud services such a Google Colab, Amazon Web Services or Kaggle (owned by Google).

I quickly realized that training on my PC, while possible, would take too much time since the VRAM (Memory of the GPU) is too small and my memory bandwidth is not fast enough.

For my first attempts at training, I started using cloud services. I began with Google Colab, where you can run python code online on their server GPUs, but this is time restricted and not always guaranteed to work since the demand of these GPUs is high and can sometimes lead to getting worse GPUs than anticipated. Because of that, I switched to Kaggle. Kaggle is less restrictive about the time and resources that you can use but also has slower GPUs. Despite their less restrictive policies, the processing power of Kaggle was still not enough to finish my first training in a reasonable time (One model would have taken a couple of weeks). I had already expected this while trying out the cloud solutions and therefore contacted the ZHdK Immersive-Arts-Space and the ICST because I heard they both have powerful GPUs. Luckily, I was then able to train my first model on a Nvidia 3080Ti and later got the opportunity to train on a Nvidia 4090 from the ICST.

From that point on, I experimented with the different training configurations to tackle problems such as latency, artifacts, and bugs by RAVE. Because of my lacking experience in training models, this was just trial and error and took a lot of time since one training run still took 3-5 days of constant running to finish. If I decided to tweak the setting, I started the training all over. I later learned that you could change some settings while the model is training, but this takes experience and a more in-depth knowledge about how these settings influence the training process. To reach a satisfactory point in terms of the sound quality of the models, it took me approximately 6 weeks of training. About a dozen models had been trained after, but only a handful of them reached a usable state.

4. USING RAVE MODELS: NN~

After exporting a model, the model can be imported into Cycling 74's Max (called Max in the following) with the nn~ object developed by IRCAM. Max is a graphical IDE (Integrated development environment) where you can visually connect objects to create your own program code. You can then create a graphical user interface for the program by using sliders, dials, matrices, graphs etc, all provided within the Max software. Furthermore, you can import packages created by other users that enhance the basic functionality of Max (i.e., nn~).

You can then load a model and directly send audio through its trained network (Figure 12).

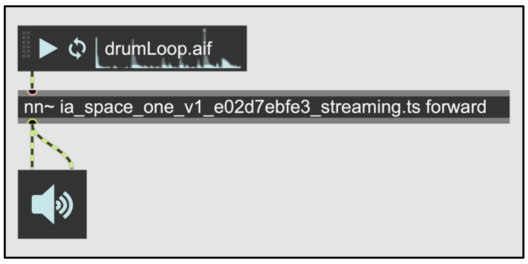


FIGURE 12

But the model can also be split into the encoder and decoder, enabling direct interaction with latent variables.

Figure 13 shows two nn~ Max objects which represent the trained encoder and decoder. The encoder outputs the values of the latent variables of the particular audio material that is sent into it. As discussed, these latent variables are essentially the coordinates of the latent space. Figure 13 also shows that it is possible to take control of these latent variables and consequently influencing the audio generation. Here, we are altering the second dimension of the latent space. By using the dial, we can explore how altering this dimension affects the sound characteristics.

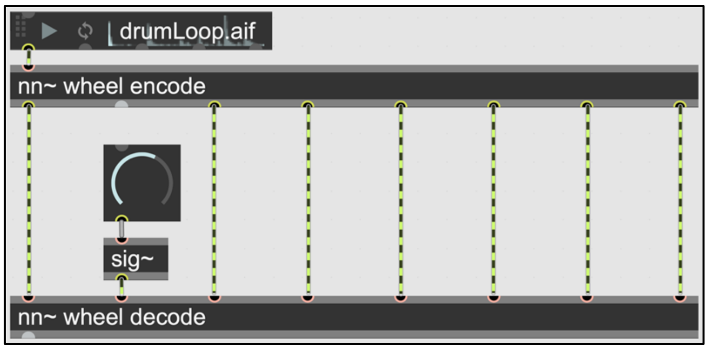


FIGURE 13 A RAVE MODEL SPLIT INTTO ENCODER AND DECODER USING THE NN~ OBJECT.

CONCLUSION

In Figure 13, the model is called wheel, but for simplicity, I'll assume that this is a model trained on trumpet samples and proceed through the important stages that occur when inferring a RAVE model from audio input.

1. The *drumLoop.aif* is sent into the encoder.
2. The encoder puts the drum sample into the latent space that was originally trained on trumpet samples. This is a challenging task because the latent space is specifically organized for trumpet sounds.
3. The latent variables are output to the outlets of the encode object and sent into the decoder.
4. The decoder then uses the latent variables to create an output.
5. To manipulate the generation, the second latent variable of the trumpet latent space is manually controlled by feeding a signal value into the second inlet. In other words, we are taking manual control of the second dimension of the latent space, which will influence the location of the latent space we sample from and consequently influence the sound of the model.

This happens in real-time or near real-time, depending on the model and processing power of the computer. The result is a trumpet RAVE model that tries to recreate the drum loop with trumpet sounds, while also being manipulated with our dial.

When inferring a RAVE model with audio samples, it is called **timbral transfer** because the model transfers its timbral/sound characteristics onto the incoming audio. We can influence this timbral transfer by taking control of latent space variables manually, as shown in Figure 13. But we do not necessarily need to infer RAVE with audio material because we can also create audio output by only manually controlling the latent variables. Sticking to the visualization of Figure 13, it would be possible to connect a dial to every latent variable and generating audio using this way.

Besides RAVE being real time, these two different methods of inference distinguish RAVE from generative AI’s such as Suno, which do not allow inference with audio nor direct manipulation of the latent variables.

5. Designing the interface

As stated in chapter one, my main research question for this thesis is to design an interface that allows a playful, experimental exploration of RAVE models by providing multiple unique interaction methods. The importance of interface design for software instruments is well established and, in my experience, sound designers and musicians are aware of the importance of good interface design because it saves time and can inspire creative decisions. However, at the beginning of writing this thesis it was not yet clear to me that the interface would play such an important role but as I progressed and especially as I first started to experiment with RAVE, I realized how important the interface itself is to understand and explore the model's capabilities. As my research on the subject progressed, I came to understand why this was necessary.

To further explore this topic, I would like to discuss the audio software company FabFilter[6] because their products show that interface design can differentiate a product from the competition even though the competitors offer similar functionality in their plugins. This becomes more evident when looking at their equalizer (Pro-Q3) that you can use inside a digital audio workstation. The audio software market is flooded with equalizers that offer similar functionality as Pro-Q3 but what sets them apart from their competition is their user interface, because it seamlessly integrates their technical innovations into a fun and easy-to-understand interface, while only introducing complexity when the user really needs it. The interface is structured in levels of complexity that flow into each other whether they are needed or not. Figure 14 shows this disparity compared to BX Digital V3 from Brainworks which is also an equalizer and a competitor to ProQ3. Digital V3 however, overwhelms the user with everything the interface has to offer whether the user needs it or not [7]. Pro-Q3 starts almost completely blank and will only reveal interface knobs and buttons when activating a frequency band. Even then, it hides these knobs and buttons again after you move your mouse away from a band. This results in a clean interface that only shows the equalization curve even if a complex eq curve is set. Additionally, at first glance it might seem that Digital V3 offers more functionalities than Pro-Q3 but in fact it is Pro-Q3 that trumps it in every category while still being easier and fun to use.

[6] Fabfilter is a software company that develops plugins for audio professionals.

[7] Both equalizers are in a setting that does not apply any processing to the incoming audio.

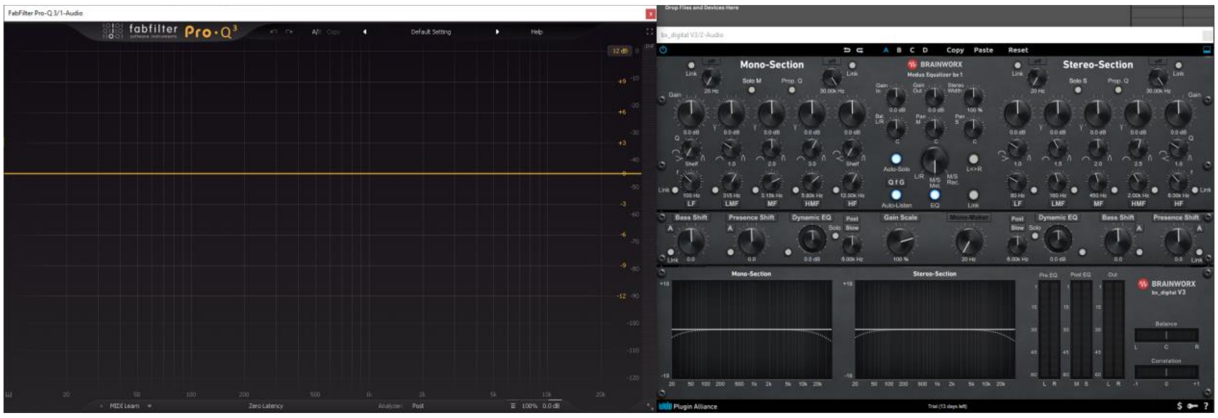


FIGURE 14 FABFILTER PROQ3 LEFT AND BX DIGITAL V3 FROM BRAINWORKS ON THE RIGHT

The point I am trying to make is that the design of an interface can have a significant impact on how we perceive and interact with something that is technically very similar.

In the context of electronic instruments, Hunt et al., 2003 describe something called *separation of sound source and control interface*. A phenomenon that emerged with electronic instruments, because unlike an acoustic instrument, the control interface is not inherently connected to the sound source and therefore the control/user interface must be defined in an additional design step. Consequently, the design of the interface does potentially have huge implications for the usage (Hunt et al., 2003), as I described with the example of ProQ3.

Because real-time generative audio models are so novel, their interface design is mostly still unexplored. And because of the inherent differences in their functional principles compared to traditional digital audio signal processing, I believe their interfaces necessitate new techniques to fully utilize the potential of this novel sound generation technology.

Therefore, the goal of my interface is not to develop a final product, rather it is the aim to provide one possible approach to an interface for real-time-generative-audio-models that purposefully provides a wide range of interaction methods that give users the possibility to playfully explore the boundaries of whatever RAVE model they desire to use. The interface can be seen as the product of the discovery phase of the double diamond design process (Figure 15), where various solutions to a problem are gathered to explore the problem from as many angles as possible.

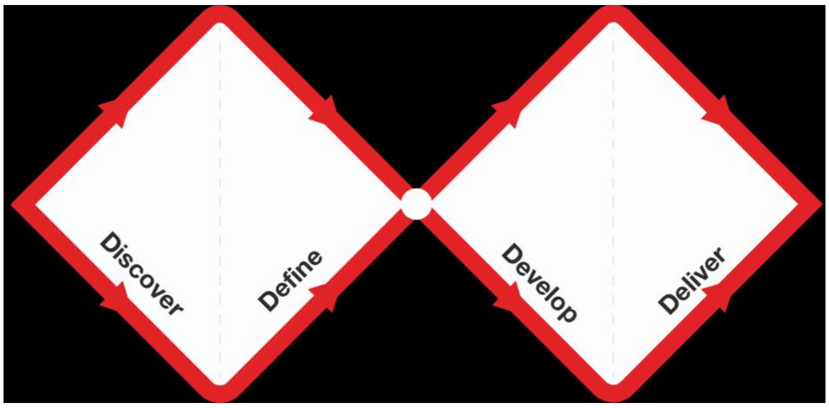


FIGURE 15 (DESIGN COUNCIL, 2024)

It is also important to emphasize that each RAVE model can be seen as an individual instrument with different behavior depending on the dataset it was learned on. This is because of the unsupervised learning network (as discussed in 2.1.4 Latent Space) with deeply entangled latent variables that makes it impossible to make universally valid conclusions that an interface could be built on. In contrast to, for example, a filter knob, which can be exactly tailored to the frequency range of the filter and, additionally, is always correlated to the frequency value. Whereas the exact meaning of RAVE latent variables is always changing from model to model.

This is why providing many interaction methods that all vary in the way they produce signal data is important because they can potentially complement each other depending on how a model reacts.

5.1 Deciding which interaction methods to use

As I described in my motivation, the interface should consist of multiple unique interaction methods with which the user can explore the capabilities of a RAVE model of their choice. Because I wanted to take advantage of RAVE's many directly controllable latent variables, I chose interaction methods that offer multidimensional output.

5.1.1 Multislider

Max's **Multislider (**Figure 16**)** is a good example of such a multidimensional output, because it combines multiple sliders into one interface, with each slider's value being bundled into one multidimensional ouput. The number of sliders and their range is adjustable making it ideal to control latent variables of RAVE models. In Figure 16, the multislider is used to directly control the eight latent variables of the decoder from a model called wheel. The number of latent variables can vary depending on the model and the settings you choose after exporting it after training. The models I trained all have 16 latent variables because I chose the maximum **fidelity**[8] when exporting my models.

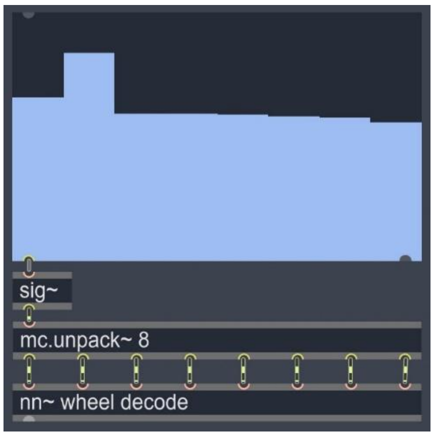


FIGURE 16 CONVERTING THE DATA FROM A MULTISLIDER TO CONTROL A RAVE DECODER

While the multislider makes it easier to set values for multiple dimensions using one control interface, it is not possible to control multiple dimensions simultaneously, because it is controlled with a mouse or a touchpad and therefore only one slider can be controlled at once. Furthermore, it introduces a new "problem" I need to explain:

If we examine the signal data the encoder produces when exciting it with sample data (Figure 17 bottom right), we can see that this data is very erratic[9]. Figure 17 shows a drum loop as input and using a less dynamic signal (i.e., an ambience) produces less erratic data. But still the encoder data is never completely static. Consequently, the decoder is not used to interpreting completely static data, which means the data the multislider outputs leads to what I call the sound of **looping grains** you might recognize from granular synthesis where the grain length is very small which creates short audible loops[10]. While this sound of looping grains is interesting, it can quickly become tiring.

[8] The fidelity value is an export setting that decides how many dimensions are exposed for interaction. The higher the fidelity the more latent variables a model has.

[9] Data that has high volatility or a high rate of change

[10] The observation of mine that static data creates this sound of looping grains is based on the output of my models. I could imagine that if a model is trained on less erratic audio data (like drones or

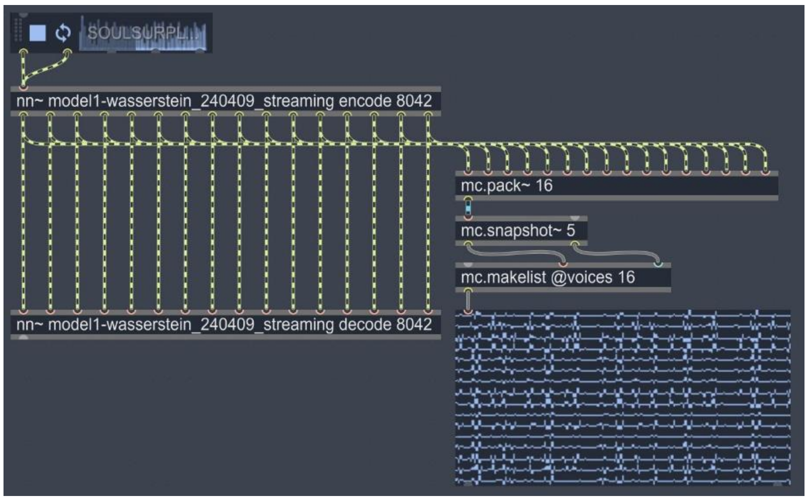

SOULSURPLU
nn~ model1-wasserstein_240409_streaming encode 8042
mc.pack~ 16
mc.snapshot~ 5
mc.makelist @voices 16
nn~ model1-wasserstein_240409_streaming decode 8042

FIGURE 17 ON THE BOTTOM YOU CAN SEE THE DATA THE ENCODER PRODUCES WHEN FED A DRUM LOOP

During my experimentations while designing the interface, I desired a more organically evolving sound that has variation to it and therefore I wanted to find a solution to generate more erratic data[11].

5.1.2 Motion Sensor

Because of the latter I started experimenting with the gyroscope and accelerometer of a motion tracker (MPU9150, Figure 18). The motion sensor's gyroscope and accelerometer each output separate X, Y and Z data streams. Consequently, it is possible to intuitively control multiple latent variables at once by using simple gestures.

Because of the previously mentioned static data, fast gestures, which result in more dynamic data, work a lot better to create sound textures containing more movement and interesting textures. But the problem of static data remains because as soon as the gesture stops, the sensor starts outputting static data again. But it is a little less of a problem because unless the controller is put on a stable surface, the controller is so sensitive to movement, that even slight hand movements create small variations in the data and therefore also to the sound of RAVE.

ambiences) the decoder is much better at interpreting data that does not have much movement in them. But this was not possible for me to evaluate and needs to be researched further.

[11] Data that resembles the volatility and rate of change that the encoder might produce.

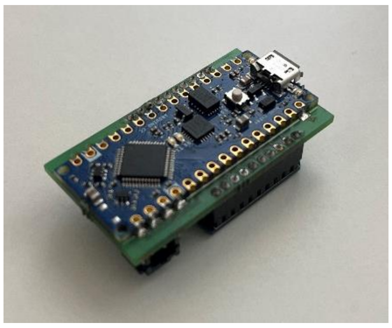


FIGURE 18 ARDUINO UNO BOARD CONNECTED A MPU9150 ON THE BOTTOM.

I tried to tackle this problem by experimenting with modulating the sensor signal with noise. For example, If I have a static signal from the motion sensor of 0.5, I then tried to modulate this signal with noise to make it flutter around its original value, in this case 0.5. While this mitigates the problem of static data it does produce rather uninspiring audio results and is not particularly easy to control as a user, because of the inherent randomness of noise. The only way to control the modulation is by adjusting the strength of the noise signal or by filtering the noise output. Hence, I abandoned this approach[12].

PREPARING AND MAPPING THE DATA

Starting from the top in Figure 19, there is a small module for each sensor axis to prepare the data. By pressing the plot button on the top left of each module, it is possible to show the data stream of each axis. Using the light blue slider on the right of each module, you can define a minimum and maximum value and therefore define a range for the signal. Additionally, pressing the center button takes the current value of the signal data as a center point, enabling to define a sensor position where the data is zero. After preparing the data, if necessary, the matrix below takes the six motion sensor axis and then enables the user to map them freely to each latent variable individually.

[12] It is possible that this approach works better with different models from mine.

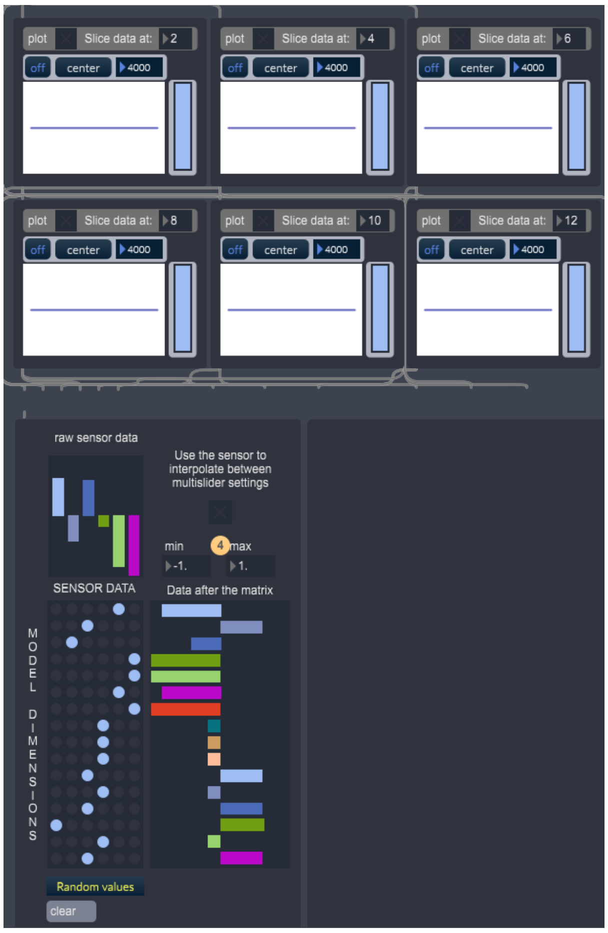
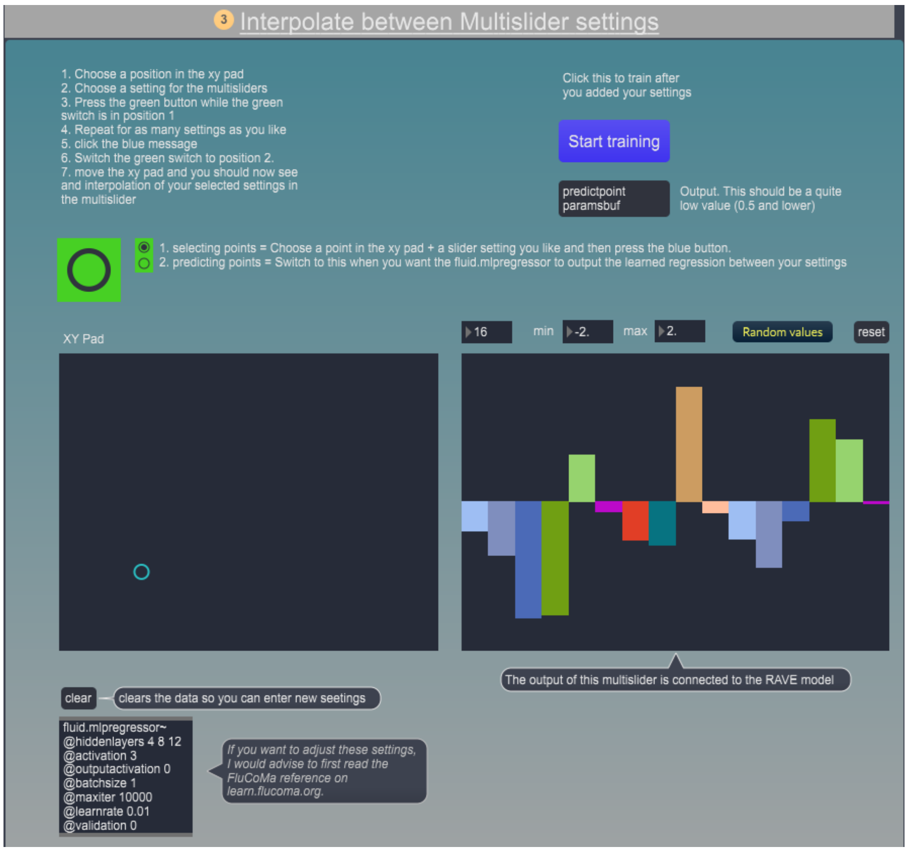


FIGURE 19 DEFINE THE RANGE AND CENTER OF THE DATA (TOP). MAP THE DATA TO THE LATENT VARIABLES (BOTTOM)

5.1.3 FluCoMa Regression

The third interaction method I introduced is the MLPRegressor from FluCoMa (Tremblay et al., 2022). The MLPRegressor is a neural network that uses supervised learning to map spaces of different dimensions together using regression. In other words, once the MLPRegressor was trained, it can smoothly interpolate between two connected spaces of different dimensionality. For example, a dial (which has one dimension) to the coordinates of a 3D space.



1. Choose a position in the xy pad
2. Choose a setting for the multisliders
3. Press the green button while the green switch is in position 1
4. Repeat for as many settings as you like
5. click the blue message
6. Switch the green switch to position 2.
7. move the xy pad and you should now see and interpolation of your selected settings in the multislider

Click this to train after you added your settings

```
predictpoint
paramsbuf
```

Output. This should be a quite low value (0.5 and lower)

[✓ 1. selecting points = Choose a point in the xy pad + a slider setting you like and then press the blue button.]
2. predicting points = Switch to this when you want the fluid.mlpregressor to output the learned regression between your settings

The output of this multislider is connected to the RAVE model

clears the data so you can enter new seetings

```
fluid.mlpregressor~
@hiddenlayers 4 8 12
@activation 3
@outputactivation 0
@batchsize 1
@maxiter 10000
@learnrate 0.01
@validation 0
```

If you want to adjust these settings, I would advise to first read the FluCoMa reference on learn.flucoma.org.

FIGURE 20 THE FLUCOMA REGRESSOR ADAPTED FOR MY PATCH

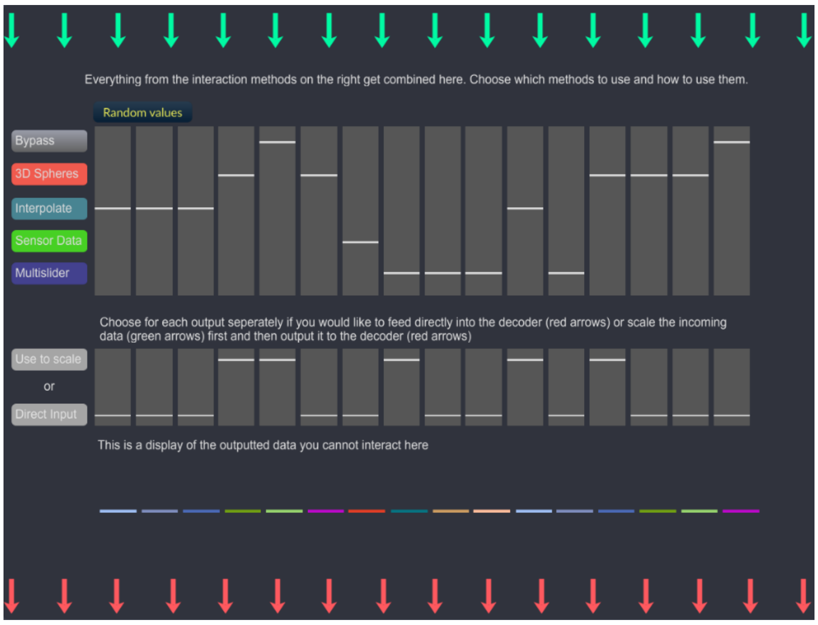
Figure 20Figure 18 shows the interface for the MLPRegressor I made for my Max patch. It works by choosing a multislider setting plus a position for the cursor of the XY pad. After pressing the green button, the cursor position and the multislider setting are now stored together in a buffer. This can be repeated for as many settings and cursor positions as desired, creating multiple linked multislider and XY-Pad settings. The MLPRegressor uses these linked settings to learn regressions between those settings. After choosing settings, users can start to train MLPRegressor using the blue button. Each time the button is pressed it starts one training run. This should be repeated until the black message box below shows a value near 0, which indicates that the model successfully learned regressions between all the chosen settings. Once training is done, the user can smoothly interpolate[13] between the settings the previously selected settings with the XY-Pad, creating a sort of preset XY-Pad that is filled with settings that are contextual to settings the user desired.

[13] While these regressions are not exactly like interpolations it is helpful to think of them like one.

5.2 Freedom of choice: Building a **junction** for the interaction methods

At this stage of the interface design, I had three interaction methods, but it was still a lot of effort to experiment with the combination of them or deciding to switch between latent variables that the interaction method should control, because I had to manually disconnect and connect Max's signal cables. For the amount of latent variables, it quickly became obvious that I needed to build something that allowed to quickly iterate through different interaction methods.

Therefore, I started to design a **junction**, as I call it (Figure 21). The junction funnels all interaction methods into one interface where users can decide which interaction method to use for every latent variable individually. Each grey bar represents one latent variable. With the white slider the user can select an interaction method or bypass the junction.


Everything from the interaction methods on the right get combined here. Choose which methods to use and how to use them.

Choose for each output seperately if you would like to feed directly into the decoder (red arrows) or scale the incoming data (green arrows) first and then output it to the decoder (red arrows)

This is a display of the outputted data you cannot interact here

FIGURE 21 THE JUNCTION

In addition, the interface features a second row of where you can decide between using the interaction methods as a scaling factor or as direct input for the latent variables. This resulted from the two distinct ways you can use RAVE, which I mentioned at the end of chapter 4:

1. Timbral transfer
2. Manual control of the latent variables

When using RAVE as a Timbral transfer tool, the junction allows the user to use any interaction method as a scaling factor of the encoder data or use a as direct input for the latent variables. Figure 22 shows a comparison between these two concepts with a dial that outputs signal data. The direct manipulation works without the encoder getting triggered by audio data, whereas the scaling only works when the encoder is already sending data.

Scaling the data is a great way to shape the sound characteristics of RAVE's timbral transfer. For example, we can use the multislider interface to influence the sound of a drum loop. Using the junction, we can then decide if all latent variables should be affected by it or just selected ones. The FluCoMa Regressor is also a great tool for this task because you can use it to store multiple favorite sound settings.

Because manipulating the latent variables directly does not require data from the encoder, the sound is directly tied to the setting or movement of the signal data from the interaction methods. For example, when using the motion sensor, the decoder directly responds to the motion. Shaping the sound from that resulting motion is then more tied to which axis data is input into which latent variable.

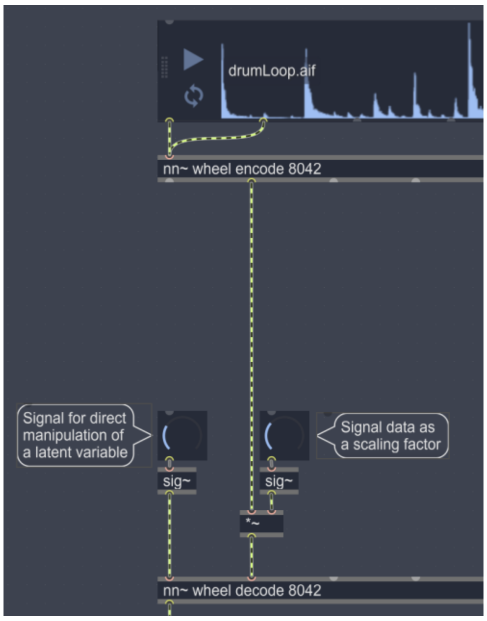

drumLoop.aif
nn~ wheel encode 8042
Signal for direct manipulation of a latent variable
Signal data as a scaling factor
sig~
sig~
*~
nn~ wheel decode 8042

FIGURE 22 COMPARISON BETWEEN DIRECT MANIPULATION AND SCALING

I would recommend to watch this video, to better understand the different methods that can be used to infer RAVE models.

6. Evaluating the interface

After designing the first prototype of my interface, I wanted it to be evaluated by experts to observe their first impressions and gather their opinion about preferences, expectations and use cases. Therefore, I conducted an expert review with selected participants that are all experienced sound designers and musicians. The group consisted of only four people, because my goal was not to test the interface systematically as if it were a final product and therefore four experts already gave me plenty of feedback to implement and evaluate.

6.1 Method

Each participant had approximately 30 minutes to use the interface during which I was always available to answer questions or suggest something. I divided the review into levels of complexity, starting with interaction methods that are easier to understand, such as the multislider, and then moving on to the more complex ones, so the participants had a smoother learning curve. Because the concept of RAVE and neural synthesis is so novel, I gave a short input beforehand about RAVE's architecture.

6.2 Results

In general, the participants described the learning curve of the interface as quite steep, and it took regular explanations from my side during the test until they were familiar with the functioning principles of RAVE. It was not the interaction methods that were hard to understand, but rather RAVE itself and how the interaction methods are controlling the sound generation. Despite this, all the experts started to quickly experiment and play with the interface to explore what impact their interactions have on sound generation. They almost used it as a learning tool to comprehend this novel sound generation method. It emphasized that the context of neural networks is of great importance for users because it evokes an interest in them to understand and explore it. "*I found it very exciting having neural networks in mind. You try to understand it somehow. You must accept that is this huge black box from which you cannot deduce a lot from, but it is definitely fun. It is not something where you think: "I heard that before". It is a surprise box." "…it is next level sound creation."*

Through their interactions, they slowly understood that the model's latent variables are deeply entangled with each other but also that it is hard to deduce what each latent variable means but they enjoyed this unpredictability. *"I liked finding out new things"*. Recent research shows that this balance between precise control and unpredictability is something sound designers want and enjoy because it allows them to explore and experiment (Kamath et al., 2024).

One expert also expressed his desire to explore the models systematically by comparing how the model would react to direct manipulation of the latent versus scaling of the encoder. *"How does it sound when I am on this side of the model versus the other [meaning scaling versus direct manipulation]. So, I can better understand where [their] difference[s] come from."*. However, when they started to use the **junction** to combine them, it introduced more complexity, which made it harder for the participants to keep track of the impact multiple interaction methods had on RAVE and therefore creating frustration. *"…You cannot quite comprehend [what is happening] if I want to go into a certain direction [it is not that easy]." "As soon as I tried to combine the interaction methods. I was like: I have a feeling what happened but how am I impacting the system?" "Understanding the relationship between the different modules in this large interface is sometimes difficult."*

The **multislider** also created confusion because some experts initially treated it like an equalizer where the sliders on the left effect low frequencies and vice versa. *"How should I understand that? The lower frequencies are on the left and the higher on the right?" "You immediately start thinking in equalizer mode…"* They quickly realized that the latent variables have no correlation to frequencies like an equalizer, but I wanted to mention it because it highlights the complexity of managing expectations experts might have and how those can impact the perception the interface and RAVE.

After this initial confusion, the multislider was understood very well: *"I understand this very good"*. And it was liked for its reproducibility of settings. *"The cool thing is that you can reach reproducibility."* The experts all used it to slowly move each slider to make a precise setting they liked. Both for the direct manipulation of the latent variables and scaling the encoder data. But it was especially useful for the experts when using it in combination with scaling the encoder data because then the multislider could be used to shape the sound of the generation with precise control reproducible settings. *"It is a mix of control that I wish for when designing sound."* But again, they realized that you can never quite conclude the meaning of each slider. *"I don't quite have an idea what I am doing but at the same time I know that I can make fallback and build something new." "[…] in the end, I cannot tell you exactly what I have to do to reach a certain goal."* Highlighting the inherent problem with an interface for a model where latent variables cannot be labeled (as described in 2.1.4 Latent Space).

The interaction of the experts became much more playful after switching to the **movement sensor**. It was considered the most fun by all experts. *"The sensor was the most fun."* The experts enjoyed exploring different gestures and movements, which they then mapped into the latent variables. Once they found a gesture, they tended to experiment more with the mapping of the sensor matrix and less with the gesture itself: using the gesture as a fixed variable and then adjusting the matrix while listening to the output. *"It is fascinating. The visual interface gives me a lot of hints [...] essentially, I am designing an input [for the latent variables] and listening what the [audio] output is doing with it."* They also remarked that the movement sensor is very expressive and felt a strong connection to their gestures. *"Gesture to sound. I understood that link quite fast" "It created associations like brushing your teeth or table tennis." "Lightsaber!" "It is so beautiful that it is so sensitive to small movements."* Furthermore, when using the movement sensor, the other hand was now free to adjust settings in the interface, which enabled the experts to multitask between gesturing and making adjustments.

When scaling the encoder data with the movement sensor, I observed that instinctively the participants tried to imitate the rhythm of the drum loop they heard with the movement of the sensor. In other words, they tried to time the data created by the sensor with the signal from the encoder to enhance individual drum hits. This is difficult to do since my models have about 600-800ms of latency, which is huge when trying to align movements with a sound. Even if the latency was negligible, the interface lacked visual feedback about the temporal relationship between movement and the impact of it on the generation, which would make it possible to time movement better with the audio content. For the review, I chose a short drum loop which was easy to remember and therefore the participants could negate the latency of the output by memorizing the loop and adjusting their movements accordingly. But with sounds that are harder to memorize (less predictable or no rhythmic structure) and therefore predict, this created a problem, because then the signal data from the interaction methods was not aligned anymore and, in effect, it was unsatisfying. This is a problem for all interaction methods, but it became more evident with the movement sensor because the experts desired to create movements that are timed with the audio input.

Like the multislider the last interaction method, the **FluCoMa Regression**, was interesting to the experts as a sound shaping tool when scaling the encoder data. With the key difference being that the experts could now make changes to all multislider dimensions at once using the XY-Pad. Additionally, some experts expressed the desire to connect the sensor with the regression, demonstrating once more how much they enjoyed the sensor as an interaction method.

As described earlier, the learning curve of the interface was high, but once the basic principles were understood, the overall experience with the interface was well perceived. *"There is not a moment of frustration and that can happen very quickly with interactions like this." "I found it pleasant. Straight forward." "In this setting, I found it pleasant."*

As the last quote suggests, it must be said that the perception of the interface might be skewed into the positive because I was always there to answer questions and guide the experts. If this was a consumer product, the complexity would likely be overwhelming. Although I was very surprised about the positive reception of the interface, I do not fully agree with the statement that there was no moment of frustration because as I earlier described the complexity of the system when combining interaction methods was and still is a problem and, from my observations, demotivated experts even though they did not explicitly say so.

"[...] it is not yet at the stage of a consumer product."

6.3. IMPLEMENTING THE FEEDBACK

After the expert review, I started to implement the feedback. In the following, I will fail these implementations.

6.3.1 DIDACTIC DESIGN OF THE INTERFACE

Didactic design is one of the most crucial points that would need to be addressed if an interface like this would be converted into a consumer product. The term didactic design is mostly used in the context of designing learning content, but I am using it to describe a didactic structure within an interface that guides first-time users smoothly from easier difficulty levels to more complex ones. Just as I described FabFilter's ProQ3 earlier. This is especially important for an interface like mine because of the complexity and the novelty of RAVE. I do not have all the answers to achieve this didactic design, but based on the expert's feedback; it starts with the reduction of complexity where necessary and maybe also a reduction in the freedom of connectivity of the interaction methods. But as my interface is merely scratching the surface of interface design for real-time-generative-audio-models, it is not something I can answer in this thesis and is therefore a topic for future research.

However, what I did do were smaller changes like improving the description texts of the interface. In addition, I reordered and numbered the interaction methods according to their difficulty level, so first-time users will have a more comfortable introduction to the interface.

6.3.2 3D Model

Because the participants enjoyed the interaction with the movement sensor a lot, I wanted to add a second interaction method which makes use of the sensor and focuses on a more playful approach while also addressing the problem of static data. I started to experiment with 3D environments and physical objects and connected the sensor with the 3D environment to control it. This seemed like an interesting approach because not only is there a lot of data you can extract from physical objects, but also because using a 3D environment could be visually appealing for users.

Within Max, we can create 3D environments where we can place physical objects like spheres and cubes. We can control these physical objects, change their physical properties like bounciness or mess with the gravity of the environment that these objects live in. This can be achieved by using Jit.gl, an interface to OpenGL that "renders images from a scene containing objects, textures and lighting" (Max Documentation, n.d.).

I placed 16 spheres, that are representative of RAVE's dimensions, inside a simple 3D cube (Figure 23). That all behave like physical objects with the previously mentioned properties.

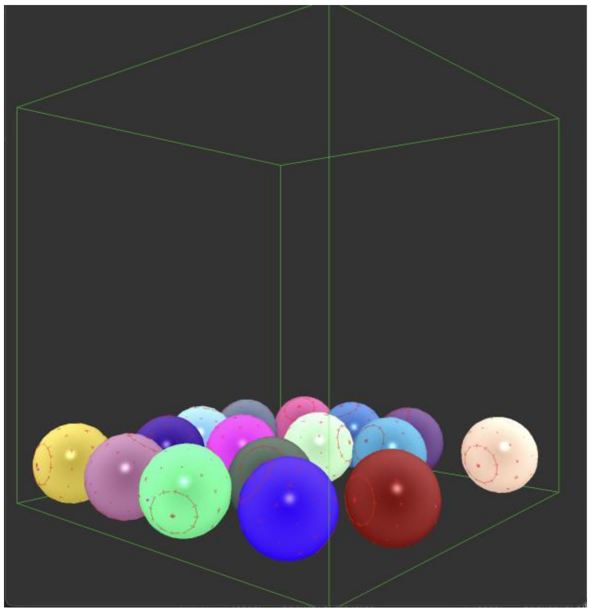


FIGURE 23 A GIT.GL.WORLD WITH 16 SPHERES.

Because a 3D world and physical objects allow for many adjustments, there is a lot that the user can specify:

1. With the sensor, you can control the direction of gravity of the 3D environment.
2. You can give the balls an impulse with a direction that is also controlled by the sensor.
3. The user can decide between the control of gravity or the impulse.
4. The force of the impulse is adjustable.
5. The size of the world is adjustable.
6. The restitution, or in other words the bounciness of the balls, can also be adjusted.

I like to compare this to the Sound Particles software which, very similar to my idea, offers a 3D environment to create sounds using a physics engine that controls the behavior of particles. In contrast to my 3D environment, Sound Particles attaches audio emitters to the particles that are captured using virtual microphones. Thousands of particles can then be programmed to follow a motion and combined with the built in doppler effect, then create complex sounds with a lot of motion in them. These motions can also be controlled using physics parameters like gravity or acceleration, enabling sound design by changing the physical properties of the 3D environment.

Extracting data from the 3D world

In Max, the position data of each individual ball is output and then scaled so it fits the signal data the RAVE model expects. By further processing the position data with differentiation, it is then possible to extract even more data streams.

1. Using the first derivative of the position, we get the **velocity**.
2. Using the second derivative of the position or the first derivative of the velocity, we get the **acceleration**.

Now there are three types of data to choose from for each individual ball: **position**, **velocity,** and **acceleration**. And changing bounciness, world size, gravity strengths, gravity direction and impulse force, gives even more possibilities to modify our 3D world and therefore the behavior of RAVE. For example, setting the world gravity to zero, the impulse response very low and the bounciness low the sensor the spheres react very sluggishly, therefore creating low position, velocity and acceleration data, which means RAVE's output will also be somewhat apathetic sounding. I hope this illustrates how the physical properties of the 3D-space can completely shape the behavior of RAVE while providing a playful visual metaphor.

To provide more variation in the data extracted from the spheres, there is also the possibility of applying randomization to the force vector that is applied to the spheres when moving the movement sensor. Depending on how strong the randomization is set, the spheres either move with slightly different speeds and direction or fly in completely different directions with differences in speed. This randomization can be set for each sphere individually. I implemented this randomization because I can only generate one singular force impulse from the movement sensor and therefore the spheres are moving on a similar path without the randomization. Consequently, creating uniform signals that then create a static sound.

I want to conclude by clarifying that this interaction method is still in a very early stage. From my experience, it still lacks the playfulness and joyful interaction of the other methods. One reason is that using the motion sensor to control the spheres is still difficult to do intuitively. Ideally, a performed motion would translate directly to the motion of the spheres, resulting in a stronger connection between the visual interface and user gestures.

6.3.3 Smaller fixes

After addressing the previously mentioned, there were a lot of smaller issues I gathered from the expert review that needed fixing or implementation regarding user experience. Firstly, I added randomization (Figure 24) to all multisliders and the movement sensor matrix. This was requested multiple times during the review and is an especially useful feature because it lets the user quickly iterate through settings and expand on ones they like. Combined with the randomization, I added reset buttons (Figure 24) that set the multisliders and the matrix back into a default state. This is especially useful for multisliders because it allows to reset all sliders to zero while examining each latent variable separately.

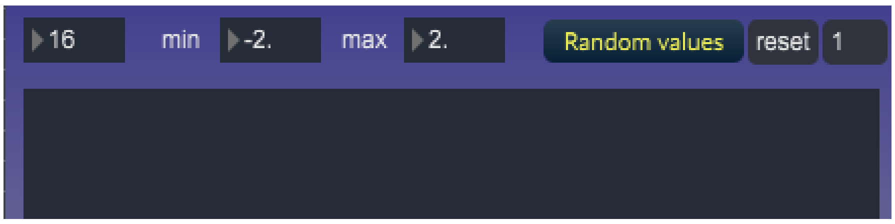


FIGURE 24 RANDOM VALUES AND RESET BUTTON ON THE TOP RIGHT

When pressing the random values button, the multislider will set a random setting within the range specified with the “min” and “max” values in the middle of Figure 24. The reset button will set all multislider values to zero.

Next up, I implemented a recording feature, which was also requested. This does not need much explaining since it only records RAVE’s output, which can then be saved into a file for further use.

The plotting and scaling device for the movement sensor data was also something that needed overhaul, after noticing it was barely used by users because of its complexity. When pressing the *center* button (Figure 25) the current value, which the sensor outputs, will be the new center point and the range of the output data will revolve around this new center point with a selected range (In Figure 25 it is a range of 4000). This enables the user to hold the sensor in a position they want and then define this position as their zero point where only zeros will be output. Consequently, this position will silence the model's output.

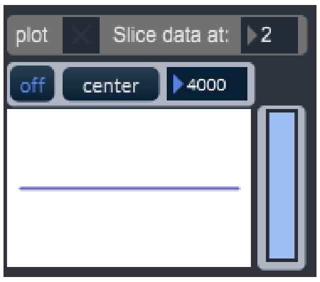


FIGURE 25 PLOT AND SCALE SENSOR DATA

Following the change to the plotting and scaling device, I combined the two junctions into one because as mentioned in *6.2* , this created confusion for the experts. I have already shown the finished junction in Figure 21 but before that it was divided into two modules: one for the direct manipulation of the latent variables and another for the scaling of the encoder data[14]. Although I could not verify this with a test, I think that this change reduces confusion and brings more ease of use.

During the review, the desire to use the movement sensor in combination with the FluCoMa Regression was expressed multiple times. Therefore, I attached a separate FluCoMa Regression section to the movement sensor module (Figure 26). Now the user can hold the sensor in a position they like, train it together with a multislider setting and consequently can interpolate between multislider settings using gestures. The regression is turned off by default and can be turned on if needed. Consequently, the movement sensor now has two operation modes, which enhance its possibilities.

[14] Unfortunately, I do not have a picture of the divided junction.

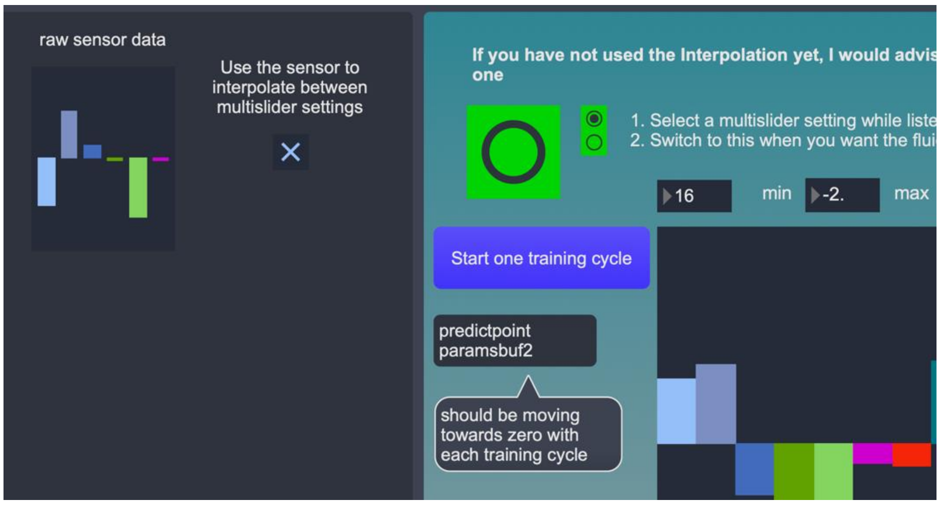

raw sensor data
Use the sensor to interpolate between multislider settings
If you have not used the Interpolation yet, I would advis
one
1. Select a multislider setting while liste
2. Switch to this when you want the flui
16
min
-2.
max
Start one training cycle
predictpoint paramsbuf2
should be moving towards zero with each training cycle

FIGURE 26 CLICKING THE BUTTON TO THE RIGHT OF THE RAW SENSOR DATA DISPLAY WILL OPEN THE FLUCOMA REGRESSION SECTION

After implementing the feedback, the interface consists of four interaction methods: A Multislider, a movement sensor input that can be combined with FluCoMa Regression, FluCoMa Regression with an XY-Pad and the 3D physics world. All interaction methods are channeled into the junction, where the user can decide which interaction method to use for each dimension individually. Additionally, it is possible to switch between direct manipulation of the decoder and scaling of the encoder data for each dimension individually. Hence, the user has various possibilities to customize the interaction with RAVE and, therefore, also the sound.

7. Discussion and Conclusion

This thesis explored the design of an interface for experimental exploration of RAVE models and contributed possible approaches to interfaces for real-time-generative-audio-models. The interface should not be viewed as a finished product but rather as scratching the surface of the topic. It can be viewed as a study for the interaction with real-time-generative-audio-models in general, which, in my opinion, will become increasingly relevant for future products that use real-time-generative-audio-models as its core technology.

One of the main design decisions was to provide multiple fun and versatile interaction methods that encourage a playful exploration of RAVE models. These methods can be freely combined with one another to manipulate latent variables directly or scale generated encoder data while it is being inferred by audio. The expert review showed that the experts enjoyed having a variety of interaction methods to explore and learn about the sound characteristics and behavior of the models. In addition, it also illustrated how the interaction methods complemented each other depending on the task the experts wanted to achieve. On the one hand, the motion sensor was enjoyed by all participants because of the excitement it gave them to control the model's sound generation with motion. And the other hand, the experts also enjoyed to precisely control/explore latent variables or shape the timbral transfer with interaction methods such as the regression or the multislider. Although this could not be validated in expert review, the 3D-world demonstrates that RAVE also offers potential for interfaces that offer less control over the latent variables itself but rather offers more playful and abstract interaction experiences where the interaction method is a playground itself.

It is as important to emphasize that I designed the interface with users in mind that have never heard of RAVE before and are not familiar with generative audio tools but are sound-designers or musicians just like me with an interest in adding another tool to their sound and music vocabulary. Based to the expert review, an interface that allows for experimental exploration of RAVE models was well received, and it is encouraging to observe that the experts enjoyed exploring and learning about RAVE models. The many customization options proved important for the experts to evaluate which methods work best and pick their favorite ones. Of course, this variety also has its downsides. Especially, when combining different interactions methods, it became clear that this can be overwhelming, but since the review was only 30 minutes per person, it is not possible to say if this would still be an issue after longer usage of the interface. But ideally, in my opinion, the number of interaction options would need to be reduced to a more sensible number, but of course that necessitates further user tests.

A related problem is the steep learning curve regarding the variety of interaction methods and the novel context of real-time-generative-audio-models that. Therefore, without a basic introduction into the functioning principles of RAVE, it might be difficult for inexperienced users to navigate and understand it. This is not something I could validate or refute in my thesis and therefore I am cautious to phrase a conclusion here, especially since longer usage of the interface was not evaluated. But personally, I think it is very challenging for future products to convey the strengths and weaknesses of real-time-generative-audio-models to users through interface design without frustration for the user. A reduction in possible interaction and combination methods could be a good start, in my opinion.

As discussed in 6.2 , timing sensor movements with temporal events of the input samples is a challenging task because of the model's latency. An additional visual interface that establishes a connection between the audio input and the interaction data would be helpful, so the user better understands which part of the audio sample is being manipulated. How this could be achieved is a question for future research.

As I mentioned, RAVE is an unsupervised network. While RAVE models sound astonishing and the interaction with the latent variables is great for an interface that takes an experimental approach like mine, they are not designed for tasks that have very particular goals in mind, because the latent variables are not labeled with sound characteristics sound designers and musicians can use for precise control. For example, a very common task for sound designer is to create ambiences for movies or games. If we use a RAVE model that was trained on ambiences, we could use it to create a fitting ambience for a particular scene only by experimentally exploring the RAVE latent variables until we find something useful. But since the latent variables are not labeled, this can be very tedious and might not work at all. Furthermore, the most realistic sounding outputs of RAVE are generated when inferring it with audio samples but for a sound design process this is counterintuitive because it does not make a lot of sense to search for an ambience in a library, use it to infer RAVE and adapt the output of RAVE by controlling the latent variables. It would be much more useful if usable ambiences would be generated by only inferring the decoder directly without the need for encoder data.

This is where prompt-based audio generation models are still superior because the inference only needs text and no preexisting audio content, but they do not offer the adjustability and real-time generation. If future real-time-generative-audio models develop a supervised training process that generates labeled latent variables that are useful for sound designers and musicians, it would open the door for interface approaches like Krotos Audios' Krotos Studio (Figure 27).

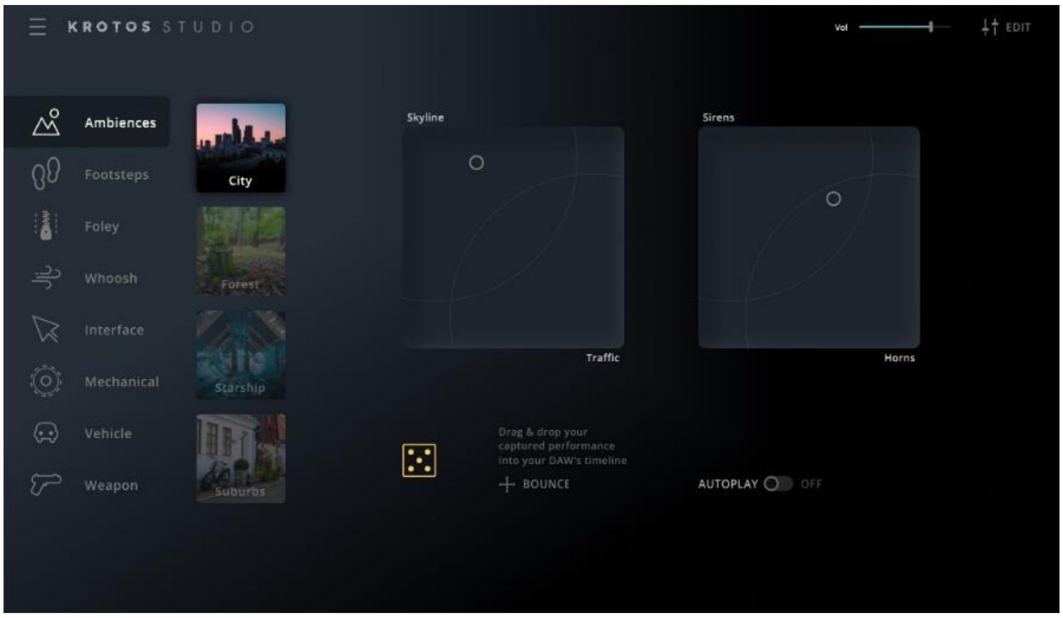


FIGURE 27 KROTOS STUDIO BY KROTOS AUDIO

Krotos Studio is an interactive sound design tool where different sound categories can be selected and then performed with XY-Pads. Figure 27 shows the Krotos Studio's interface with the city ambience category selected. With the two XY-Pads the sound characteristic of the city ambience can be adjusted between closer traffic sounds and skyline and sirens or horns. To my knowledge, Krotos works by fading between audio recordings. If we now imagine a RAVE model with labeled latent variables in the background, it already illustrates how powerful of a tool this could become. Especially if we have sixteen controllable latent variables that all correspond to meaningful labels (i.e., traffic intensity, daytime, location, vegetation, height over ground, birds, planes, etc.) and are entangled with each other. Interaction approaches like these would combine playful exploration with functionality that professionals desire into one cohesive and easy-to-understand user interface/experience.

This brings me to my last point: DAW Integration. I can only speak for myself but most digital tools, instruments, effects, etc. I use are integrated into my DAW workflow and I suspect that most professionals do not enjoy having multiple tools outside of their DAW. Therefore, DAW integration is an important if not a necessary step to make a successful audio AI tool geared towards audio professionals. This can range from providing VST support[15] to integration of AI tools within the DAW itself. While there is no audio software on the market yet to feature fully integrated AI, an example is

[15] While writing this last chapter IRCAM released a VST integration for RAVE.

Adobe’s video editing software Premier, where multiple specialized AI models are integrated within the software environment that clean audio, supply subtitles or expand video clips with generative models, to name a few. I could imagine that DAW manufacturers such as AVID, Steinberg or Ableton could develop their own AI tools that perfectly integrate within the workflow of their DAWs. Which in my opinion makes most sense for user experience and unique selling points for these manufacturers.

To conclude, my thesis shows that the interface design for real-time-generative-audio-models is still a vastly unexplored research topic. My thesis presents one of many approaches to this topic and as described not only does the interface design have much potential but rather it is an integral part of further development of architectures for real-time-audio-models that will path the way for playful and functional AI audio tools for audio professionals. In my opinion, future products that combine these two disciplines have a high chance to create something that will enhance professional audio workflows in terms of productivity and creative expressions.

BIBLIOGRAPHY

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing. https://doi.org/10.1007/978-3-319-94463-0

Briot, J.-P., Hadjeres, G., & Pachet, F.-D. (2020). *Deep Learning Techniques for Music Generation*. Springer International Publishing. https://doi.org/10.1007/978-3-319-70163-9

Caillon, A., & Esling, P. (2021). *RAVE: A variational autoencoder for fast and high-quality neural audio synthesis* (arXiv:2111.05011). arXiv. http://arxiv.org/abs/2111.05011

codecademy. (2023). *Binary Step Activation Function*. https://www.codecademy.com/resources/docs/ai/neural-networks/binary-step-activation-function

Design Council. (2024). *The Double Diamond*.

Despois, J. (2017). *Latent space visualization—Deep Learning bits #2*. https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df

Engel, J., Hantrakul, L., Gu, C., & Roberts, A. (2020). *DDSP: Differentiable Digital Signal Processing*. https://doi.org/10.48550/ARXIV.2001.04643

Google. (2024, May 13). *Model inference overview*. https://cloud.google.com/bigquery/docs/inference-overview

Hunt, A., Wanderley, M. M., & Paradis, M. (2003). The Importance of Parameter Mapping in Electronic Instrument Design. *Journal of New Music Research*, *32*(4), 429–440. https://doi.org/10.1076/jnmr.32.4.429.18853

Ideami, J. (n.d.). *Edge Horizon Approach*. https://losslandscape.com/gallery/

Jordan, J. (2019). *Variational Autoencoders*. https://www.jeremyjordan.me/variational-autoencoders/

Kamath, P., Morreale, F., Bagaskara, P. L., Wei, Y., & Nanayakkara, S. (2024). Sound Designer-Generative AI Interactions: Towards Designing Creative Support Tools for Professional Sound Designers. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–17. https://doi.org/10.1145/3613904.3642040

Liu, Y., Jun, E., Li, Q., & Heer, J. (2019). Latent Space Cartography: Visual Analysis of Vector Space Embeddings. *Computer Graphics Forum*, *38*(3), 67–78. https://doi.org/10.1111/cgf.13672

Max Documentation. (n.d.). *Working with OpenGL*. https://docs.cycling74.com/max8/vignettes/working_with_opengl_topic

Muriuki, G. (2018). *Loss function illustration*. https://towardsdatascience.com/deep-learning-personal-notes-part-1-lesson-2-8946fe970b95

Rocca, J. (2019). *Understanding Variational Autoencoders (VAEs)*. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Tremblay, P. A., Green, O., Roma, G., Bradbury, J., Moore, T., Hart, J., & Harker, A. (2022). *Fluid Corpus Manipulation Toolbox*. https://doi.org/10.5281/ZENODO.6834643

USE OF AI FOR WRITING THIS THESIS

Since I wanted this thesis to be as accessible as possible, I wrote it in English. English is not my native language and therefore I used a variety of AI tools to help with sentence structure, grammar and spell checking. The AI I used, are listed below.

QuillBot Paraphrasing Tool: Because I am not a native English speaker, I used QuillBot whenever I needed to refine sentence structures to get my point across. I did not use QuillBot as a tool to generate text ideas.

Built in Word Editor: I used the built in Word Editor for grammar and spell checking.

Deepl Translate: To translate single words from German to English.

ProWritingAid: For correcting punctuation errors.

ChatGPT: Help for coding and general questions about machine learning and neural networks. ChatGPT was never a direct source that I used as evidence for my argumentation. I verified everything ChatGPT claimed with actual research.

Declaration of Autonomy Master-Thesis

I hereby confirm that I have written this Master's thesis entitled "A Bottleneck of Opportunity - Designing an interface for the experimental exploration of RAVE models" independently and without outside help.

Any content taken from other sources such as texts, images, audio, graphics, software, etc., whether verbatim or analogous, is correctly cited with full attribution of authorship and source. In addition, all passages created with the help of AI-supported programs are clearly marked and provided with the exact name of the program used and the prompt applied. It is declared how AI tools were used for the translation of my own text, idea generation, brainstorming or similar.

Furthermore, I confirm that the thesis has not yet been published and has not been submitted in an identical or similar form as an examination or final project at another university, educational institution or in another degree program.

I acknowledge that a violation of these requirements may have legal and disciplinary consequences in accordance with § 26 Regulatory Framework for Bachelor's and Master's Degree Programmes at Zurich University of the Arts in conjunction with §§ 8 ff. Ordinance of the Universities of Applied Sciences Act.

With my signature I confirm the accuracy of this information:

First name: Floris　　　　Last name: Demandt

Swiss matriculation number: 21-192-182

Date: 21.05.2024

Signature: [illegible]