3YP Report

17 May 2023

Frankenstein - An Intelligent Modular Synthesiser

Name	Page Range	Page Total
Siheng Feng	6, 12-33, 96-97, 111-113, 115	30
Haedam Oh	34-56, 98-104, 105, 116	30
Yuanjun Wan	57-79, 106-110, 113, 117-118	30
Zhaoqiu cheng	3-6, 7-12, 80-96, 114, 118-120	30

Contents

1 Introduction - Zhaoqiu Cheng, Siheng Feng **3**

1.1 Project Overview - Zhaoqiu Cheng . . . 3

1.2 Market Research - Zhaoqiu Cheng . . . 5

1.3 Performance Objectives - Siheng Feng . . . 6

2 Introduction to Synthesiser Basics - Zhaoqiu Cheng **7**

2.1 Basic Concepts of Modular Synthesisers . . . 7

2.2 Introduction to Basic Synthesis Techniques . . . 11

3 Module 1: Parameter Matching - Siheng Feng **12**

3.1 Introduction . . . 12

3.2 Backgrounds and Related Works . . . 15

3.3 Parameter Matching Prototyping . . . 17

3.4 Parameter Matching Summary . . . 33

4 Module 2: Instrument synthesis and Timbre transfer - Haedam Oh **34**

4.1 INSTRUMENT module overview . . . 34

4.2 Instruments synthesis: algorithms and methods . . . 36

4.3 Training and Evaluation of instruments . . . 39

4.4 Timbre transfer . . . 55

5 Module 3: Reverberation Module - Yuanjun Wan **57**

5.1 Overview . . . 57

5.2 Introduction to Reverberation . . . 58

5.3 Room Impulse Response(RIR) Structure . . . 59

5.4 Artificial Reverberation Model . . . 60

5.5 Feedback Delay Network . . . 64

5.6 Parameter Matching of Reverberation Model . . . 67

5.7 Implementation and Evaluation . . . 72

5.8 Software Interface Design . . . 76

5.9 Conclusion . . . 79

6 Hardware and Overlay Design - Zhaoqiu Cheng **80**

6.1 Why do we Want to Make a Hardware Interface 80

6.2 The Objectives of Hardware Design 80

6.3 Specifications to Eurorack Hardware Design 81

6.4 Frankenstein VCO-VCF-EG Module 82

6.5 Frankenstein Instrument Overlay Design 94

6.6 Frankenstein Verb Overlay Design 95

6.7 Interactions Among the Hardware Modules 96

7 Software Back-End - Siheng Feng, Haedam Oh **96**

7.1 Introduction - Siheng Feng 96

7.2 Programmable Hardware - Siheng Feng 97

7.3 VST Plug-in Design: JUCE - Haedam Oh 99

7.4 Server - Haedam Oh 101

7.5 Application Programming Interface(API) - Haedam Oh 102

7.6 Database - Haedam Oh 103

8 Business - Haedam Oh, Yuanjun Wan, Siheng Feng **105**

8.1 Project Management - Haedam Oh 105

8.2 Project Finance - Yuanjun Wan 106

8.3 Risk Analysis - Siheng Feng 111

8.4 Privacy for Data Collection - Siheng Feng 112

9 Summary - Siheng Feng, Yuanjun Wan **113**

1 Introduction - Zhaoqiu Cheng, Siheng Feng

1.1 Project Overview - Zhaoqiu Cheng

1.1.1 Motivations and Project Objective

The primary objective of this project is to lower the entry threshold for beginners interested in using a conventional modular synthesiser. Today's beginners often prefer virtual studio technology (VST)(as shown in figure 1) and traditional analogue synthesisers, mainly due to their simplicity and ease of use. However, modular synthesisers have the advantage that the signal path in can be modified manually, allowing users to create unique sounds. This comes at the cost of having to learn the complex interactions between the modules (as shown in figure 2), which can be daunting for beginners. We hereby introduce Frankenstein, a modular synthesiser equipped with machine learning and intelligent features. These features guide the user through the process of making music without them having to learn the intricacies of traditional modular synthesisers.

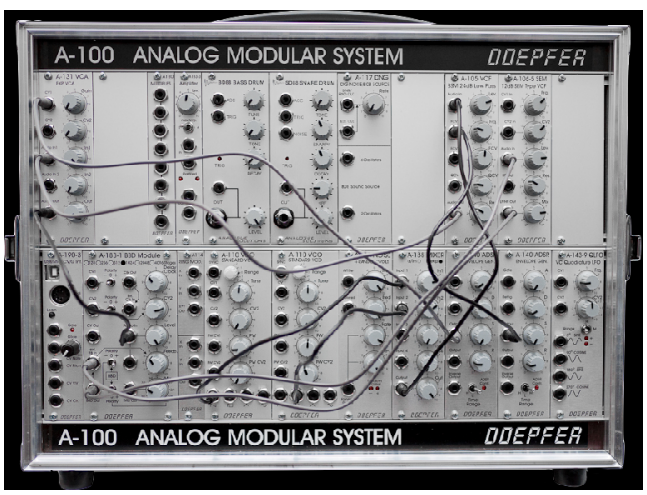


Figure 1: An example of various modules mounted into a box and interconnected with patch cables[1]

Figure 2: Arturia DX7[2], a virtual software synthesiser that has a much simpler interface

1.1.2 Frankenstein's solution

Frankenstein is an AI-powered modular synthesiser that consists of three intelligent features: Parameter Matching, DDSP Instrument, and Reverberation which are supported by Frankenstein's hardware, to provide guidance to beginner users. Each of the three features has its unique hardware design, as shown in figure 3. The unique features of Frankenstein are listed below with an overview in figure 4:

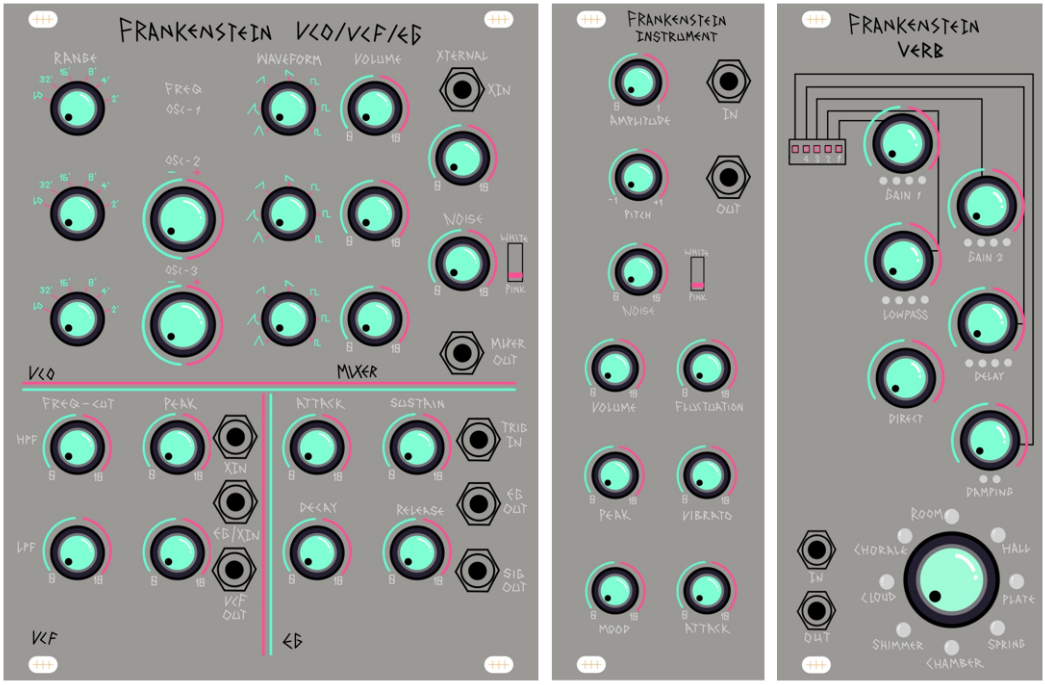


Figure 3: The designed overlay to support the three features, with names: VCO/VCF/EG - Parameter Matching; Instrument - Instrument; Reverberation - VERB

1. Parameter Matching(section 3) The algorithm integrated into the **VCO/VCF/EG overlay**(6.4) of Frankenstein is designed to solve one of the most asked questions by the beginner: "How can I recreate a sound I heard with a synthesiser?". This is a skill that usually requires years to master. We hope that bringing this function to Frankenstein will help beginners overcome the challenges of learning to use a synthesizer and motivate them to learn to recreate the sounds of their favourite singer or composer.

2. Instrument(section 4) Together with **Instrument overlay**(6.5), it is responsible for generating musical sounds like those of an instrument. This module is a unique and efficient synthesiser because Instrument integrates different instruments using a single baseline model, DDSP. Its three main features are 1. Synthesising many different instruments (e.g., Violin, Piano, Flute, Guitar, Drum, Danso). 2. Timbre transfer from one instrument to another. 3. Three independent controls on pitch, loudness and reverberation and noise with 6 different high-level controls.

3. Reverberation(section 5) The reverberation module sits on the **VERB overlay**(6.6) and provides users with complete control over various parameters to create reverberation effects representative of different environments. This module is suitable for use in both studio production and live performances. To assist beginners in achieving the desired reverberation effects for specific environments, an additional

software algorithm has been integrated. This algorithm automatically calculates the necessary parameters, enabling users to produce the desired reverberation effects with ease.

4. Intelligent user guidance and interaction(section 6) The parameters obtained by the Parameter Matching algorithm are reflected on the hardware interface automatically with our motorised knob design. For the reverberation module, the parameters are mirrored by the LED guidance ring that highlights the position to which the knobs need to be turned to in order to get the desired sound effect. Users can enjoy patching the modules together in different scenarios, whether it is for composing or for live performances.

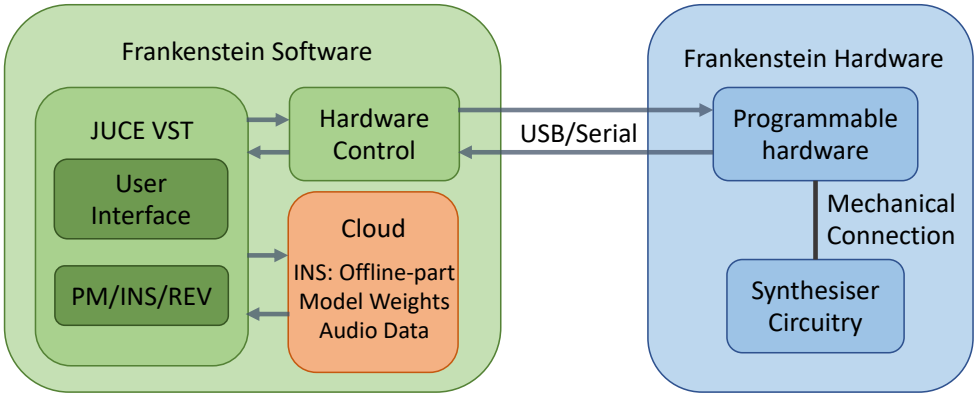


Figure 4: The high-level overview of Frankenstein. The local(calculation on the user's device) components of Parameter Match/Instrument/Reverberation (PM/INS/REV) are hosted by JUCE VST. The Cloud part contains the Instrument component that requires higher computation power. Intelligent user guidance is achieved by programmable hardware in section 7.2.

1.2 Market Research - Zhaoqiu Cheng

In this section, we present our observations of the current synthesiser market and define our target market and customers, thus proving the need for bringing an introductory, beginner-friendly product to the market.

1.2.1 Background Research – Current Synthesiser Market

According to market research conducted in 2022 [3], the synthesiser market size is predicted to have a growth of $65.16 million between 2022 and 2026. Music festivals, which have been very successful in recent years, are key drivers of the synthesiser market as they require DJs to invest in equipment and inspire fans to get into music production. The report has also shown a rise in demand for substitute products, slowing down traditional synthesisers' market growth. The major substitute product suggested by the report is music synthesiser software, creating competition for traditional hardware synthesisers.

In a market analysis done in 2019 by the same company [4], it is suggested that artificial intelligence (AI) has opened new possibilities with its composing abilities. **Therefore, the development of AI-based synthesisers is expected to be a significant market trend**. This report also observed that due to the growth of social media, do-it-yourself (DIY) music projects have become more commonplace, and these require musicians and engineers to build their own instruments.

1.2.2 Research on Inexperienced Users of Modular Synthesisers

We interviewed electronic music lovers around us, 'as people completely new to modular synthesisers, what is stopping you from buying a modular synthesiser and creating music with it?' Firstly, we do not have a competent level of music literacy, which makes the production process difficult. Secondly, modular synthesisers are not as easy to use as acoustic instruments such as piano and guitar in terms of how their sounds are made and altered. Although there are books and videos that introduce how modular synthesisers are used, it is time-consuming for someone who is new to them to pick them up and start composing.

1.2.3 Target Customers

Based on the evaluation on ourselves, we defined our target customers as people who are interested in electronic music, have a competent knowledge of music theory, master at least one musical instrument, and are beginners to synthesisers. We have quite a specific definition for our target customers. Therefore, our product falls into a niche area where AI and hardware interfaces are combined, which has rarely been explored, and thus makes our product stand out for its novelty.

1.3 Performance Objectives - Siheng Feng

This section outlines performance objectives related to user experience. We categorise our performance objectives into two categories:

1. Requirements for Local and real-time execution on user devices for interactive functions.
2. Server-side and offline execution on servers for the user-submitted jobs that require extensive computations.

The real-time objectives apply to the entire Parameter Matching module in section 3, the basic synthesis part of the Instrument module 4.2.3, and the entire Reverberation modules 5.7.5. The 'Near-Real-Time'

objectives apply to advanced features of the Instrument module 4.2.3. Based on the user hardware survey conducted by Steam[5], more than 98% of users have personal computers (PCs) with more than 8GB of RAM. This survey might skew towards higher-performance PCs. Considering Dell's product lineup[6], only 6 out of 222 laptop models have less than 8GB of RAM. Ideally, real-time computations should be completed within 10 seconds[7], which is the usual wait time before users shift their focus. For offline server tasks, we align our time objective with conventional audio synthesis algorithms in section 4.2.3, where the time taken for audio synthesis ranges from several hours to a week.

Here are our performance objectives summarised in a list:

1. The RAM requirement for user-side applications **should be less than 8GB** (User experience).
2. Real-time tasks should be executed within **10 seconds** (User experience).
3. The offline task should be executed at a time matched to conventional algorithms (User experience).
4. The model we use must demonstrate its **effectiveness and feasibility** as a prototype (Project Development).

2 Introduction to Synthesiser Basics - Zhaoqiu Cheng

2.1 Basic Concepts of Modular Synthesisers

To better understand and develop our design of modular synths, we should first clarify a few basic concepts. The potential of modular synths lies in their interconnectivity. A great analogy of modular synths is the human body. Different synthesiser modules can be considered as different body organs, each performing its specific roles. Patch cables, or blood vessels in human bodies, carry the information or "nutrients'' for the whole system to work. While chemicals are essential for the communication of body organs, control voltages are the key to module communications in synths. Thus, to allow the users to stitch their 'Frankenstein', we have to use patch cables to communicate the electronic signals among each component.

2.1.1 Voltage Controls

1. Pitch voltages: Voltage is usually used to modify pitch. In voltage-controlled oscillators, the pitch control protocol is known as volts per octave or V/Oct, meaning a 1-Volt change leads to a 1-octave

change, where the pitch shifts up as the voltage shifts up.

2. Control voltages (CV): Modules like voltage-controlled filters (VCF) and voltage-controlled amplifiers (VCA) take control voltage as the input. For instance, control voltages are used as a signal for the cut-off frequency of a VCF and a signal to attenuate or boost a sound via VCA.

3. Audio signals and patch cables: Like CV, audio signals are electronic signals, but these always fluctuate within the audible range. Thus, it is possible to patch an audio output to a CV input to create interesting sound effects. Monitoring the CV signals by patching them into the outputs is also possible, but the result may be disappointing and could damage the speakers (e.g. if the CV pitch exceeds the speaker's operating frequency range). Therefore, beginners can patch cables without causing damage to the physical interface.

   Different modular synthesizer formats feature various patch cable formats to transmit signals across modules. We chose the Eurorack format for this project as it is the smallest, most common, and beginner-friendly form of modular synths. The patch cables used for this format are the 3.5 mm tip-sleeve cables. Unlike tip-ring-sleeve cables that are used for both balanced and unbalanced signals, tip-sleeve wires are only used for unbalanced signals. [8][9]

2.1.2 The Material for Sound Synthesising - Sound Sources

The most common sound generator is the oscillator or a voltage-controlled oscillator, precisely tuned with the previously mentioned 1V/Oct protocol. There can be other audio sources, such as a microphone input, where the input is a real-time recorded sound. Another sound source can be a sampler, which records and plays back a portion of music.

Most basic waveforms are created by adding sinusoidal waves at different frequencies, as expressed with Fourier Series.5 Oscillators can generate one or more basic waveforms, including sawtooth, square, triangle, and sine waves. These waves differ in strength, which is defined in terms of how many harmonics the wave contains and their intensity.

The Sawtooth wave is ideal for imitating synth bass and lead sounds due to its powerful sound, for it contains both odd and even harmonics, which decrease in volume as the frequency increases. The square wave is weaker and hollower than the sawtooth wave since it only contains odd frequencies, making it suitable for mimicking wind instruments like the clarinet. Next, we have the triangular wave, which also includes only the odd frequencies but is even weaker in volume. The triangular wave is a

typical starting point to make the bass sound as it has the standout lower frequency harmonics. Finally, the sine wave is the basic building block of all the other basic waveforms, which is often used to recreate pipe sounds, such as flute or organ, and is the weakest waveform. [8][9]

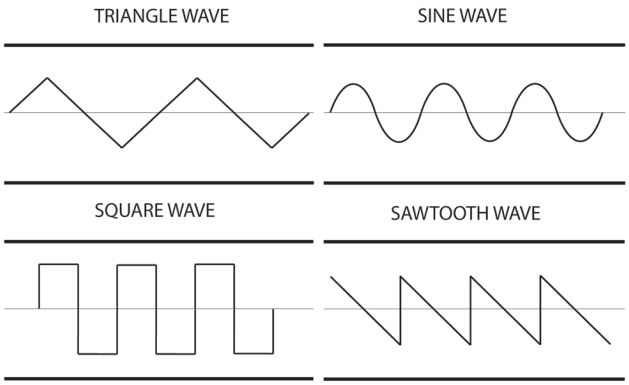


Figure 5: Common sound waveforms.

Figure 6: Left: Four common sound waveforms. Right: Minimoog Model D that uses subtractive synthesis.

2.1.3 To Alter the Characteristics of the Sound - Audio Modifiers

After the first step of “making some noise” comes the second step of modifying the sound generated. A common way of modification is to trim the sound with a filter, essentially cutting off some harmonics. Some common filters are high-pass, low-pass, band-pass, and notch filters. While we are familiar with the first three types of filters, the last type – notch filters, reduces the intensity of the harmonics near the cut-off frequencies, leaving the harmonics further away to pass through, which works the opposite way to band-pass filters.

One might think that all filters produce the same sound if set to the same parameters, but this is not always the case. Different filters may have different circuitries despite having the same specifications, leading to a tonal shift in the sound output.

Another category of audio modifiers is wave shapers. They allow you to edit a harmonically simple wave and add harmonics to give the sound a more dynamic texture. Therefore, starting with a simple sine wave is more common than a sawtooth or a square wave. Some common wave-shaping tools are saturation circuits, full-wave and half-wave rectifiers, and wave folders. [8][9]

2.1.4 To Modify the Fundamental Frequency of Sound – Audio Modulators

Modulators bring life to a sound by adjusting when and how a sound comes in and fades out. They achieve this by changing the loudness, harmonic mix, and other properties of the sounds. General modulators include envelope generators, random sources, and low-frequency oscillators.

1. Envelope generators (EG): A voltage profile is created to describe the loudness of sounds, which is separated into four stages, namely attack (A), decay (D), sustain (S), and release (R), as shown in the figure below7. 'Attack' is the stage where the voltage level rises to its maximum from 0 volts. Then the voltage level decreases to a chosen level ('Decay') and is held relatively constant, for it is the major stage for sound presentation ('Sustain'). The voltage level then decays to zero from the sustain level as the key is released ('Release'). [10]

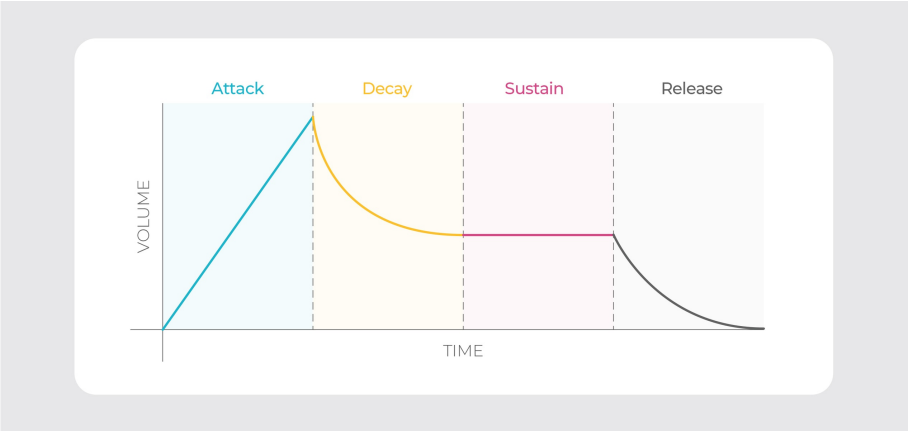


Figure 7: A diagram that indicates the ADSR of the sound.[11]

2. Random sources: Components like random gates, triggers, and voltages are all forms of random sources. They are all control voltage signals that indicate the start and finish of notes. The voltage rises to 5V when the note starts, holds for the length of the note, and drops down to 0V for the gate signal. The trigger signal is more like a pulse function that indicates when the note has been struck. The change in voltage in a random voltage source is not limited to a specific amount. [9]

In this section, we introduced the basic building blocks of modular synthesisers, namely control voltages and patching cables, and basic modules for sound synthesis, such as VCO, VCF and EG, for which the VCO design will be further explored in section6.4.

2.2 Introduction to Basic Synthesis Techniques

Various synthesis techniques have emerged since the invention of sound synthesis. Here we will introduce the two basic synthesis techniques we adopted in our models: additive synthesis, subtractive synthesis, and frequency modulation synthesis.

2.2.1 Additive Synthesis

The idea of additive synthesis is to create a broad spectrum of sounds by adding simple sounds originally inspired by the design of church organs. Relatively simple sounds from individual pipes are combined to form a rich harmony. The theory of additive synthesis is rooted in Fourier analysis. Although Fourier analysis usually deals with sinusoid-made-up periodic sounds, additive synthesis can also recreate non-periodic sounds with certain functions other than sinusoids.

Additive synthesis is one of the most powerful techniques for its generality. However, it is also computationally demanding as a large amount of data is processed for every note. To make a note sound livelier, non-harmonics and noise sources need to be added along with the harmonics, which involves many oscillators and envelop generators. The process of tuning a sound can then be very time-consuming and complicated for synthesiser beginners, which is a problem we aim to solve with our project. [12, 13]

2.2.2 Subtractive Synthesis

Unlike additive synthesis, which combines various sources to produce the desired sound, subtractive synthesis combines a harmonically complicated sound and trims down unwanted frequencies using filters, envelope generators, and low-frequency oscillators. In the mid-1960s, engineer Robert A. Moog introduced a game-changing electronic instrument – the Moog synthesiser, which used subtractive synthesis as its basic technique. A part of our project is based on his design of the popular Moog Minimoog Model-D70.

Nonetheless, subtractive synthesis has its limitations. The main drawback is the lack of feedback interactions between the sound modifiers, modulators, and sound source, which introduces difficulties in controlling sound characteristics and recreating the timbre of acoustic instruments. This is why the most popular sound processing technique shifted from subtractive to frequency modulation or FM synthesis. [12, 13]

3 Module 1: Parameter Matching - Siheng Feng

3.1 Introduction

3.1.1 Motivation for Parameter Matching

Revisiting the Analogue Synthesiser An analogue synthesiser is more than just a musical instrument. It's a complex tool that requires a foundational understanding of its numerous components to manipulate sound effectively. For example, to use the oscillator, users must discern the acoustic distinctions between a 'square-shaped wave' and a 'saw-shaped wave' or interpolate the sound when these two are combined [14]. This learning curve can be steeper compared to traditional instruments such as the piano, as the synthesiser has many adjustable parameters, each with its own non-intuitive implications.

Beginners' Challenges A common question from beginners is "How should I start using a synthesiser" [15]. Advice usually suggests getting hands-on experience, either through buying a physical synthesiser(figure 8) or using software like a Virtual Studio Technology (VST) or learning from tutorial software(figure 9). Inevitably, mastering a synthesiser involves much trial and error. Amongst these beginner questions, one topic that often comes up is how to replicate a specific sound they've heard. It has hundreds of posts on the Reddit forum[16]. It might at first seem simple, but it actually requires the knowledge to decompose the audio to match the corresponding parameter set, a skill that's challenging to acquire.

Figure 8: Arturia Mini V[17], one of the recommend synthesisers for beginner from the community

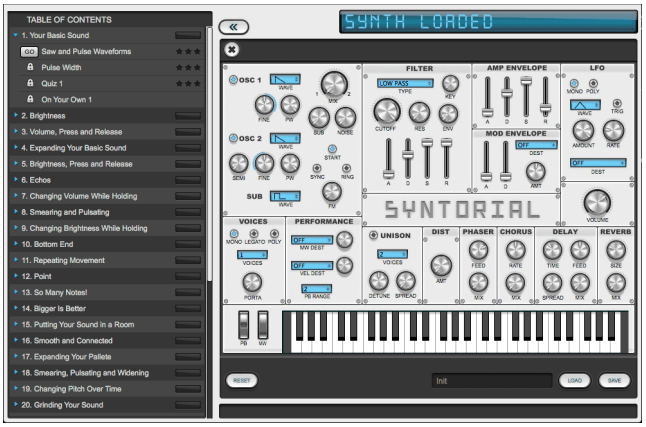


Figure 9: Syntorial[18], A synthesizer tutorial software

Introducing the Frankenstein's Parameter Matching Module The paradox between beginners' initial desire to recreate certain sounds and the experience required to achieve this often discourages begin-

ners. To simplify the process for beginners, we decided to incorporate the Parameter Matching module[1] into Frankenstein. Using machine learning, this module matches an input sound, from song snippets to locomotive roars, to synthesiser parameter presets that best recreate this sound. This gives a starting point for users to construct their own sound and keep their enthusiasm during the learning process.

3.1.2 Parameter Matching System Overview

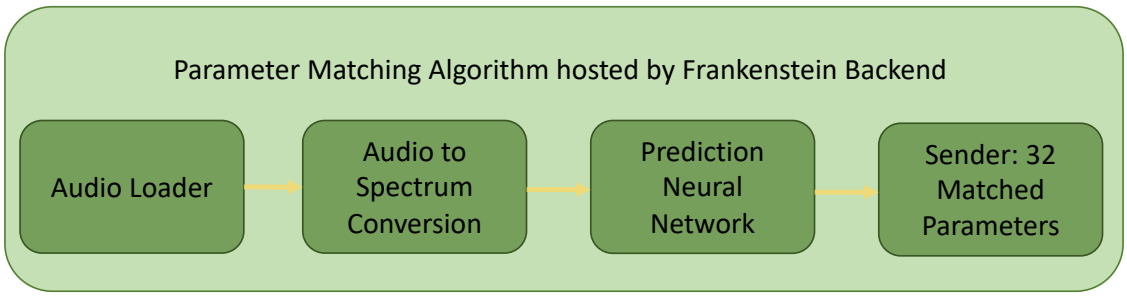


Figure 10: System Overview

At a higher level, Parameter Matching is an algorithm (figure 10) hosted by the computation back-end in section 3.3.2. It controls the VCO-VCF-EG hardware modules in section 6.4. Inside the Parameter Matching algorithm, a four-second audio effect is received from Frankenstein's VST and is then pre-processed by the audio-to-spectrum conversion into an intermediate format that the neural network can recognise. Lastly, 32 parameters[2] are predicted and sent back to Frankenstein's VST in section 3.3.2.

A prototype was built based on the available works in this area as a proof of concept and a minimum viable product (MVP). Although the prototype was built with limited resources and on a smaller scale, it delivers outstanding results and also meets our design objectives, as introduced in the section 1.3 and 3.1.4. The implemented code for parameter matching can be accessed through the link of Siheng's shared folder (`https://drive.google.com/drive/folders/181wBma-MXyyFgpl6MOZew1NWl-8Zv_iK?usp=sharing`).

3.1.3 Similar Products

Parameter Matching has emerged as a trending research topic in recent years. According to market research, there is neither a synthesiser VST nor hardware that successfully integrates parameter matching

[1]Remark: The parameter matching module is for matching the parameters of an analogue synthesiser, which is different from the "parameter matching" in later reverberation-module that finds the reverberation's configuration.

[2]The actual number of parameters that users can interact with is based on the hardware modules they have purchased (Frankenstein's base VCO-VCF-EG module has 21 parameters), but the model still predicts 32 parameters to adapt to the user's future upgrades

into its tutorial function. Table 1 lists some representative similar software, along with their pros and cons.

Software Name	**Pros**	**Cons**	**Price**
Syntorial[18]	• Interactive Learning • Comprehensive courses with instructions • Built-in synth for user projects	• Steep learning curve • Expensive • No Parameter Matching	£115.99
Ableton Learn Synths[19]	• Interactive learning • Free to use • Web-based, no need for installation	• Limited function • No built-in synth • No Parameter Matching	Free
Audulus[20]	• Ability to customise patching like a modular synth • Mobile platform, vivid user interface	• Complex interface • Not designed for learning • No Parameter Matching	Free with in-app purchase: up to £20.99
Flowsynth[21]	• Provides sound matching for parameters • Free to use	• Only a research demo • No built-in synth • Requires manual database download and installation • Requires knowledge of Python package management and local server	Free

Table 1: Related Softwares

Syntorial requires users to guess parameters from a given sound and marks how accurate the guess is. Ableton Learn Synth is a set of interactive lessons explaining the meaning of each parameter; users can't develop their own sound and integrate it into their works. Audulus, which focuses on patching modular synthesisers, contains even fewer tutorial features. It's clear that none of these three tutorial software options includes the parameter matching feature often required by beginners. This is a common problem faced by the current synthesiser community. On the other hand, Flowsynth is more of a demo program in machine learning research and not a product, which presents difficulties for non-technical users.

Frankenstein is designed to address the limitations listed above. This hardware synthesiser integrates parameter matching into its existing framework of a modular synth. The backend API conceals all tech-

nical details so that users can operate it in a similar way to any other synthesiser. As it's designed to assist users in using the Frankenstein modular synth, there's no direct competition with the systematic courses provided by Syntorial. It might instead assist Syntorial's users, as it provides sound matching for arbitrary sounds, not limited to those within the Syntorial courses.

3.1.4 Parameter Matching accuracy objective

The quality of matching is rather a subjective matter due to the inherent limitations of the synthesiser described in section 3.2.2. In Eslings[22], the Normalised Mean-Square-Error (MSE_n) on parameter prediction can serve as a more straightforward, objective quantitative criterion for accuracy. Our objective is to achieve an MSE_n value close to 0.15 to 0.18. Ideally, the best criterion would be to compare the input sound with the recreated sound directly from the synthesizer, but this is challenging to accomplish (illustrated in section 3.2.2). These are the listed Accuracy Objectives as a summary.

1. The predicted parameter should capture the important features of the input audio.
2. The parameter prediction MSE_n value below 0.18.
3. Algorithm is capable of learning a general understanding of sound (explained in section 3.2).

3.2 Backgrounds and Related Works

This section briefly covers works on parameter matching and some general strategies in music information retrieval (MIR). These works helped us in some early decision-making processes and guided the development of our prototype.

3.2.1 Synthesizer tuning with intuitive words

There are various approaches to assist users in operating an analogue synthesiser, including the use of verbal or semantic labels such as ”brightness” and ”metallic”[23]. While such approaches reduce the complexity of interaction between the synthesiser and user, it fails to address the versatility of an analogue synthesiser and the vast range of sound effects it can produce[24]. The semantic labels can also be subjective, resulting in different interpretations from different individuals, such that using semantic labels for parameter control can be unreliable or imprecise. **As a result, this feature is not included.**

3.2.2 Limitation of parameter matching in general

There is also a clear limitation that a synthesiser cannot replicate any input sound. These out-of-domain signals might lead to inaccurate estimation as the synthesiser cannot produce them in the first place. Some research, such as Masuda's[24], also focuses on modelling these out-of-domain sounds with a differentiable DSP (Digital Signal Processing) synthesizer capable of capturing similar parts between in-domain and out-domain methods. **Frankenstein must be able to have a general understanding of the sound, not limited to mapping parameters.**

3.2.3 Parameter matching approaches

Common workflows to tackle this problem involve 2 steps, information retrieval from audio and performing estimation based on information. Previous attempts use linear regression(LR) [25], deep learning techniques(DL)[26], and iterative genetic algorithm(GA)[24]. **GA approaches** employ iteration to find the matched parameter set. The result can be great, but it can also be time-consuming. In[24], such matching took roughly 20-25 minutes to perform, **which is not acceptable** for our design objective in section 1.3. **DL approaches** include the use of sequential models, for instance, Long Short Term Memory[26] (LSTM) to learn from samples or the use of Convolutional neural network(CNN)[27] to learn the spatial relationship of audio spectrum in figure 13. But none of the two methods helps to resolve the limitation in section 3.2.2.

There is another DL method that helps to resolve the limitations by using Variational Auto Encoder(VAE) in Esling's[22]. It is believed that a vector of, for example, 64 numbers is enough to represent the entire input audio snippet. We can see its intuitive connection to a synthesiser, where users can use a dozen parameters that generate much more complex audio data. The Flowsynth[21] mentioned earlier employs this method in the software. But it comes with a cost of higher complexity to implement.

3.2.4 Related work section summary

From the listed method above, LR and GA have their own limitations, with either poor accuracy or slow execution. For those DL models, we believe that learning a more general representative space is important for beginners as they might not understand what sound can be produced by the synthesizer and what not. **As a result, VAE architecture is used in our prototype.**

3.3 Parameter Matching Prototyping

3.3.1 Introduction

We implemented a prototype to demonstrate the MVP of parameter matching. Decisions were made based on the performance and practicability under the scope of this project. In order to implement a prototype with machine learning in music information retrieval, there is a general to-do list that we summarised from previous works[27, 24, 22]:

1. **Software Platform**(3.3.2): A VST was used to simulate Frankenstein's hardware and make the development of parameter matching easier without a hardware synthesizer in data collection.
2. **Audio Feature Extraction**(3.3.3): A method to extract information from raw audio data, which made the representation of this information compatible with common neural network architectures.
3. **Audio Dataset Gathering**(3.3.4): These data were used to train this Machine Learning model. Models are data-driven. The quantity of the data and its relevance to our task are important.
4. **Synthesiser Parameter Selection**(3.3.6): The main objective is to help beginners get used to a smaller but effective set of parameters to control the synthesiser.
5. **Parameter Matching VAE Model Building**(3.3.7): This is the core of the parameter matching. The data extracted is fed into the network, and we obtain the predicted synthesiser's parameter. A few different sub-architectures were explored for this task.
6. **Model Training**(3.3.8): Model training is another important aspect of a machine learning project. The weights of the model were trained with an appropriate setup and strategy to optimise the result.
7. **Performance Evaluations**(3.3.9): The performance at the training stage and at run-time on a native machine was compared. This took both accuracy and efficiency into consideration, where we made our final decision on how Parameter Matching Model was constructed.

We decided to follow this to-do list in our prototype to avoid falling into a no-man region which contains a higher risk of delay and failure. The following sections are structured based on this to-do list.

3.3.2 Parameter Matching - Selecting VST Platform

Parameter matching using real hardware synthesisers poses challenges in data collection due to labour intensity. The quality was also reliant on the precision of the person tuning the synthesiser as well as

the noise produced during audio recording. **Using a simulated virtual synthesiser is a much more reasonable idea both for data collection and model testing.** But we need to decide which one to use.

Figure 11: Diva VST by u-he[28]

On the other hand, **authenticity or correctness were less important** for developers developing or musicians using a virtual synthesiser[29]. They care more about whether the sound effects of the synthesiser are able to convey their emotions and ideas in their musical works. In particular, different synthesisers have different signal processing algorithms resulting in the same parameter set, thus producing different sounds. For instance, DX7[2] is designed based on frequency modulation(FM), which is a different type of synthesis technique from the hardware system of Frankenstein in section 6.4.

Diva VST[28], as shown in figure 11, is a subtractive synthesiser. The waveform originates from its oscillator and is then shaped by other components like filters, which mimic the signal flow on an actual analogue synthesiser. It's compatible with the dataset generation tool: Renderman[30], which is helpful for dataset generation. **Given these advantages, Diva is the chosen platform to work on.**

3.3.3 Parameter Matching - Selecting Method for Audio Features Extraction

"Signals carry overwhelming amounts of data in which relevant information is often harder to find than a needle in a haystack" Quoted from "A Wavelet Tour of Signal Processing"[31]. In this section, we compare choices of audio representations and make decisions based on our performance objective in section 1.3.

There are two common categories of representation, or we call them features defined by Humphrey et al's[32] and [33], Deep Learning Features and Hand-crafted features. Their typical pros and cons are

summarised in table 2. While Deep Learning Features demonstrate impressive performance in recent Natural Language Processing tasks, for instance, Chat GPT[34], their huge complexity (170 trillion parameters used[35]) and computation power required makes it unsuitable for our project.

Feature type	Deep learning Features	Hand-crafted features
Pros	• Learn complex features. • Applicable to recent transformer[36] architecture.	• Relatively simpler computation using traditional algorithms. • Research has a long history, reliable.
Cons	• Difficult for the human to interpret the feature extracted by machine. • Requires high computational power.	• Limitations in capturing complex patterns and variations. • Potential loss of information that we require.

Table 2: The comparison between deep learning feature and more traditional handcrafted features

We choose to build our model around Hand-crafted features, however, we still need to make the decision on which one to use. A typical feature extraction process is summarised in the figure.12 There

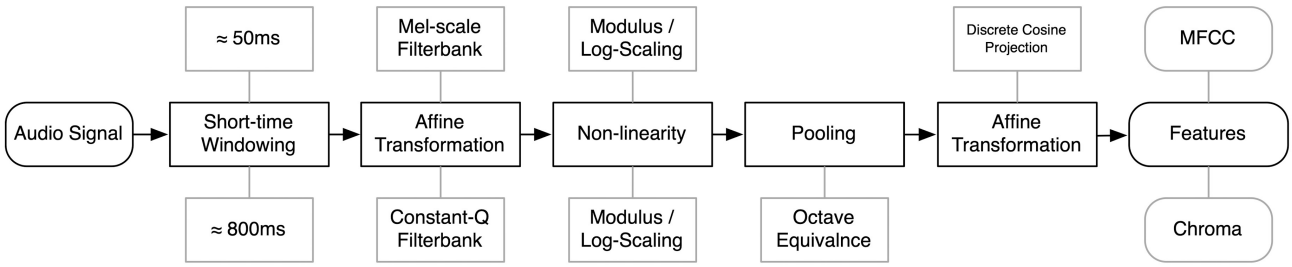
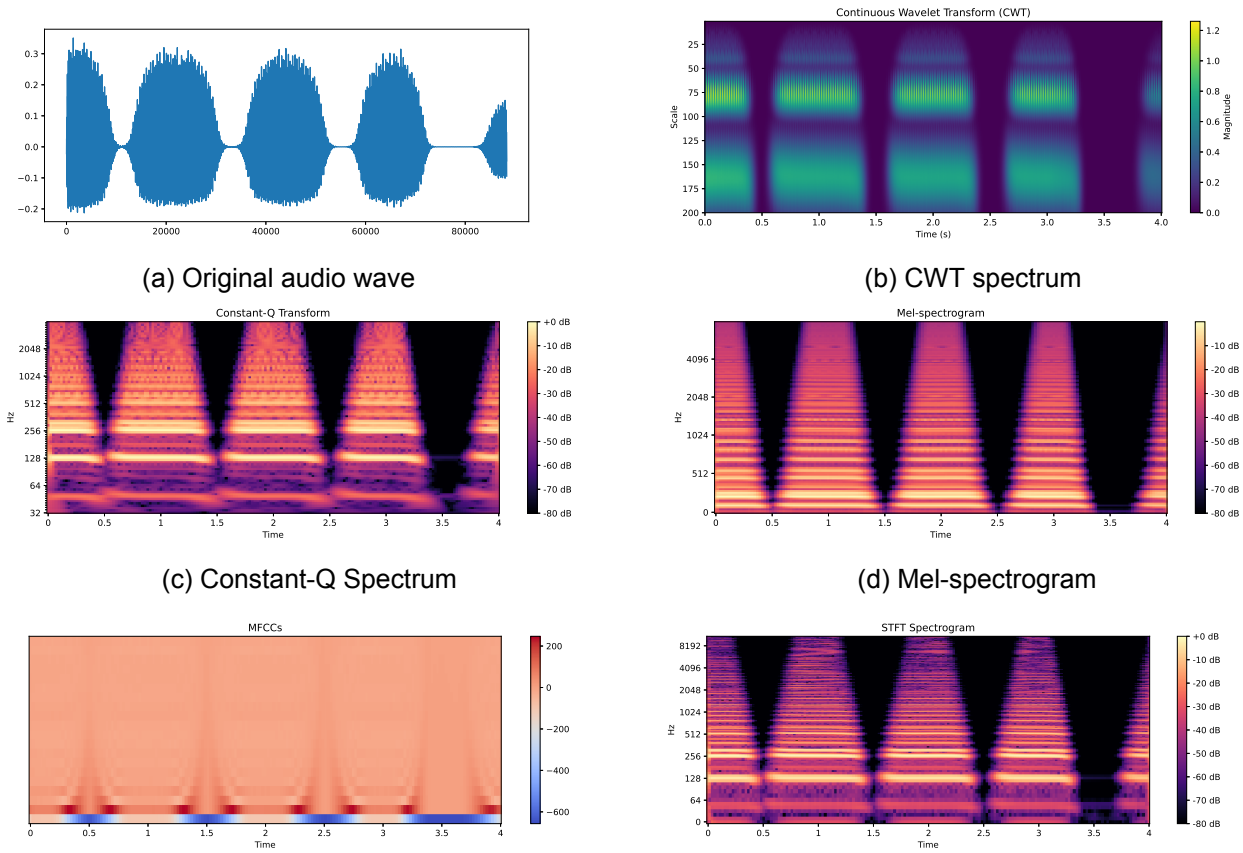


Figure 12: Traditional Audio Feature Extraction[33] process pipeline. The general process can be summarised as a time-frequency conversion from audio to spectrum with non-linear scaling

are a few common handcraft features, including STFT (short-time Fourier transform) with linear and Mel scale (Log Mel-spectrum), constant-Q transform (CQT), Mel-frequency Central coefficients (MFCC), and continuous Wavelet transform (CWT). We use Python Librosa[37] library to generate a list of spectrums as an illustration in figure 13 with the raw audio data from Diva-dataset[38].

It can be seen that Mel-spectrum contains relatively less noise and contains more spatial information than MFCC. Its amplitude is scaled to the human auditory perception, which makes it more favourable in music information retrieval tasks[22]. In Muhammad's [39], tests were conducted using different representations



(a) Original audio wave

(b) CWT spectrum

(c) Constant-Q Spectrum

(d) Mel-spectrogram

(e) MFCC spectrum

(f) STFT spectrum

Figure 13: We generated 6 different representations of the same 4-second audio effect for comparison. The specific code configuration can be found in the google drive folder in section 3.1.2.

of the classification of environmental sound. Mel-spectrum outperformed the other representations. **We chose to use Mel-Spectrum** from its strength of better human perception and information extraction for environmental data, which better suits our purpose in parameter matching as stated in section 3.1.1.

3.3.4 Parameter Matching - Dataset Gathering

As introduced in section 3.3.1, the quality of the data affects the performance of parameter matching. An ideal data set in our task should satisfy the following criteria: 1. It should be generated by a synthesiser VST or a real synthesiser; 2. It should be in enough quantity(the more, the better); 3. It should be easily expandable if further training is required.

There are a few commonly used datasets in MIR tasks, including the popular Nsynth data set from google magenta project[40], the Diva data set[38], the Arturia data set used in [23], and others like the synth1B1 from[41]. However, none of these fits our needs. Nsynth is a large artificially generated data set with

305,979 musical notes. However, the original purpose for Nsynth was made for the audio synthesis task in architecture like WaveNet, and it was not designed for parameter matching. The Arturia data set is too small that contains only 1,233 samples. Although the samples are from a real synthesiser/VST, no label was provided at the time visited by us, therefore, it's out of consideration. The synth1B1 generated by torchsynth[41] contains 1 billion sounds that take in the design of a modular synth, allowing arbitrary connection between modules, but it requires the manual generation of data-set which can be time-consuming under the scope of our project.

For the parameter matching tasks, the easier solution is to use automation tools like DawDreamer [42] and Renderman[30]. These tools were programmed to have a Python interface that controls Diva VST to generate a set of sounds and also records the corresponding set of parameters, which can be very handful in training. The Diva data set includes 11k 4s audio generated by Diva VST, each paired with sets of 281 parameters. **We chose the Diva dataset as it fulfils the criteria defined earlier.**

3.3.5 Parameter Matching - Audio Data Pre-Prossessing

We implemented an intermediate data storage format that involves using Pandas[43] to speed up training, such that the Mel-Spectrum required by our training program doesn't need to be recalculated during each training session. The original Diva data set is saved in the scattered Numpy[44] files, each one corresponding to one audio sample. It has a format of ["chars", "audio", "param"], which corresponds to semantic tags (not related to our task), raw audio signal and the paired parameter set that generates it.

For input audio, we first truncated it into 4-second audio clips with its magnitude normalised to between 1 and -1 to ensure the balance between Samples. We then used Librosa[37] to generate a set of 224x224 Mel-spectrum for all 11217 samples[3]. We then normalised its values to between [0, 1]. This is to satisfy the input dimension and magnitude for backbone pre-trained neural networks like VGG[45] and ResNet[46], which later set benchmarks for our proposed network. Now we have computed the required input feature that we chose in section 3.3.3 and with the Mel-spectrum, **which from now on are images of size 224x224**[4], and parameters are stored in Pandas' data frame used in the later sections.

[3]Configuration: [sample rate = 22025, hop length = 396, fft size = 2048, n_mels = 224]

[4]The Mel-Spectrums in later sections are now all considered as 224x224 images, we no longer consider them as a spectral matter. Each image corresponds to one audio sample.

	chars	audio	param
0	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[-0.001064, 0.01423, 0.002949, -0.001461, 0.00...	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...
1	[[0.0, 0.0, 1.0], [1.0, 0.0, 0.0], [1.0, 0.0, ...	[-1.764e-05, 0.0002363, 4.95e-05, -2.456e-05, ...	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...
2	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[0.0002646, -0.003479, -0.0007234, 0.0003562, ...	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...
3	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 1.0, ...	[-0.0002673, 0.003696, 0.0008445, 3.01e-05, 0....	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...
4	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[-0.000429, 0.005707, 0.001185, -0.0005865, 0....	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...
...	...	...	...
11212	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[0.000525, -0.007233, -0.001715, 0.000626, -0....	{'Chrs1: Wet': 0.7140000000000001, 'HPF: FreqM...
11213	[[1.0, 0.0, 0.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[5.823e-05, -0.001474, -0.0003765, 0.0002148, ...	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...
11214	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[0.001548, -0.000559, 0.0002031, -0.0001493, 8...	{'Chrs1: Wet': 0.8427, 'HPF: FreqModSrc': 0.0,...
11215	[[0.0, 0.0, 1.0], [0.0, 0.0, 1.0], [0.0, 0.0, ...	[0.001167, -0.015396, -0.00325, 0.0016, -0.001...	{'Chrs1: Wet': 0.685, 'HPF: FreqModSrc': 0.0, ...
11216	[[0.0, 0.0, 1.0], [1.0, 0.0, 0.0], [0.0, 1.0, ...	[0.0019245, -0.02522, -0.005375, 0.002638, -0....	{'Chrs1: Wet': 1.0, 'HPF: FreqModSrc': 0.0, 'E...

11217 rows × 3 columns

Figure 14: Parameter Matching Data Frame

3.3.6 Parameter Matching - Parameter Reduction

We decide to match the input audio to, in total, 32 parameters. This is according to the number of parameters that usually appear on a beginner-friendly synthesiser[17]. The fact is that we don't necessarily need more parameters to have efficient control over the synthesiser. We achieve this by investigating the relationship between the parameter and the audio generated by the synthesiser, which is done in two steps[47]: 1. reduce the dimension of the spectrum of the input audio, 2. perform correlations test on parameters and the spectrum with reduced dimension.

We use Principle Component Analysis (PCA)[48] to perform a dimensional reduction of data sets. The reason for doing this is simple. **It is too computationally expansive to calculate the correlation of each parameter with each "pixel" of the 224x224 Mel-spectrums**. A Mel-spectrum is reduced to 256 most significant components, which is much fewer than 224*224 = 50176 using PCA in Sklearn [49]. Under this configuration, we achieved a cumulative explained variance of 96.32%, as shown in the figure 15, indicating 96.32% of information is retained, which is higher than that of Esling's 80%[22].

We carry out the second step using Pearson Correlation Coefficient[50] to evaluate the most influential parameters by their correlation to the audio-generated Mel spectrum. PCC is widely used for parameter selection between input and output. In this way, we can know the correlation of each pixel of the Mel spectrum with respect to each parameter. The result of our PCC analysis is shown in figure.15. We obtain a list of parameters, ranked by their influence, including attack, decay, wave shape and others. An additional manual screening is performed based on[22] to ensure the parameters selected by this process are intuitive for beginners. Based on this, **a set of 32 Parameters[5], is chosen for parameter**

[5]Including ADSR of Diva's ENV1 and ENV2; Delay, Phase, Rate of LFO1 and LFO2; the shapes, tune and volume for OSC

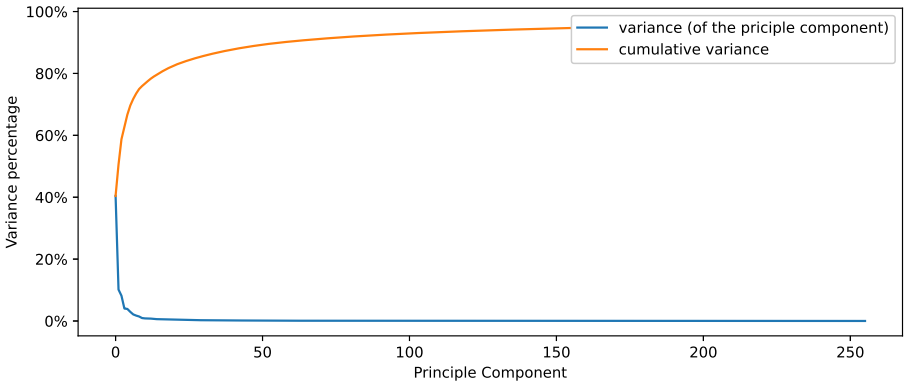


Figure 15: The explained variance of PCA on Mel spectrum: The higher the explained variance, the more information our principle components retains from the original Mel-Spectrum of the audio

matching. This ensures some not commonly used parameters for beginners. For instance, "Rotary" of Diva VST[28] is not included, despite it having a higher influence on the output audio.

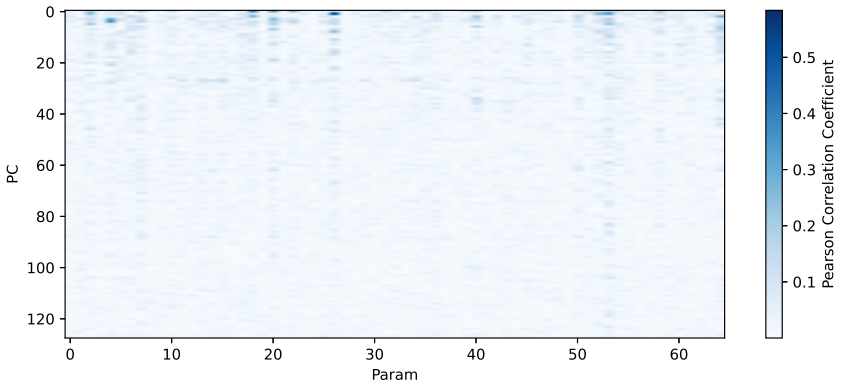


Figure 16: Pearson Correlation Coefficient for different parameters with different principal components. The colour denotes the relevance of a parameter to a principal component(PC). We sum the PCC of a parameter for all PC to obtain the overall influence of a parameter on the generated audio.

3.3.7 Parameter Matching - Neural Network Design

Introduction We present Frankenstein's network FRKN as the prototype and the MVP of Frankenstein's Parameter Matching Module, and We show that FRKN can achieve comparable performance in parameter matching than some deeper networks like VGG16[45] and ResNet50[46] while using much less memory and computation time. In this section, the detailed design of the FRKN is described, including a brief introduction to the VAE network that builds our model on.

and VCF's configurations. Terms like ADSR and ENV are common terminologies for Diva VST. These are the parameters that control the sound output. For more explanation, please refer to[28]

Brief introduction VAE Introduced by Kingma and Welling[51], the Variational Autoencoder (VAE) aims to represent complex high-dimensional data in a lower-dimensional latent space. It presumes that data distribution in a higher dimension (in our case the audio wave and Mel-spectrum) follows a distribution in a lower dimension (in our case the parameter set), thus simulating the synthesiser that generates audio from a set of variables. VAE is a probabilistic model with Bayesian inference, which accommodates the variation and flexibility in the data. Methods using VAE address the limitation in section.3.2.2 by attempting to model data on this audio space instead of on parameters directly, facilitating a more general understanding of sound.

A VAE network consists of three main blocks, including the encoder and the decoder blocks for the unsupervised learning on audio space[22], and the third supervised regressions network to the parameter set from the lower dimension latent variable. The Encoder reduces the dimension of the Mel-spectrum to a lower dimension, and the Decoder aims to recreate the input image. Combining the Encoder and Decoder, we are targeting to learn a meaningful lower dimensional space that can represent the input audio.

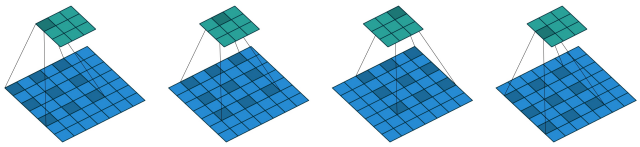


Figure 17: Dilated Convolution[52], a kernel(green) is dilated when convoluted with image(blue)

Encoder Prototyping For prototyping, we build four different encoders. Three that based on established architectures, CNN1 best resemble the CNN Encoder in [22], VGG16[45], ResNet50[46], and the last FRKN-Enc(as shown in figure 18) that we proposed in this project.

The design of the FRKN Encoder was based on the following considerations:

- **Dilated convolution**: Given the periodic nature and spatial dependency of the Mel-Spectrum in figure 13, dilated convolution layers in figure 17 might be better at learning this spatial features[52]. This is due to both the larger receptive field (kernel covering a larger area), which is suitable for harmonics information, and the preservation of spatial resolution.
- **Depth of the CNN network**: Based on Barkan's[27], we are aware that adding more convolution layers follows the law of diminishing returns. Therefore, we make FRKN have a depth of 5.
- **Latent dimensions**: We set the size of the latent variable to 64, thus simulating the audio generation process by the 64 non-constant parameters of the Diva-Data set. While this might not directly

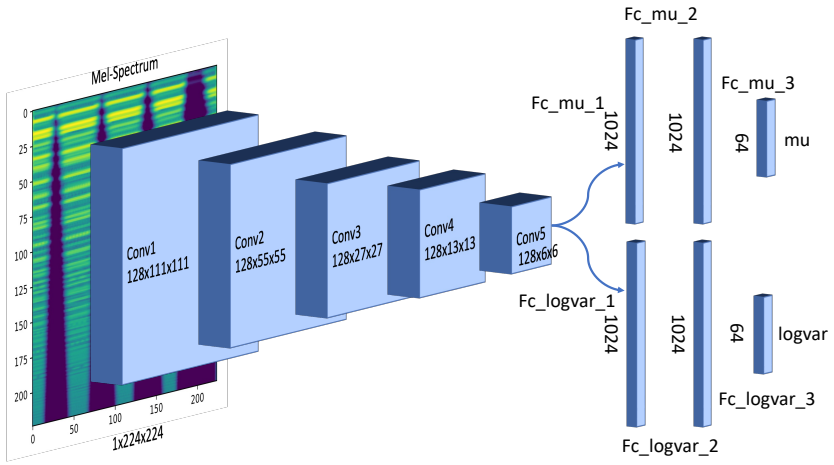


Figure 18: Frankenstein's Encoder Overview (FRKN-Enc)

model the distribution of the parameter, it models an audio space (or a shifted distributed parameter space) where the regression network attempts to match these two spaces. We validate this choice in section 3.3.9.

- **Activation**: We chose to use ELU[53] for the activation in the middle as it helps to avoid vanishing gradient. The last activation is set to Sigmoid for output of range 0-1.
- **Dilation**: We used dilation of 1 as the result of some of our earlier testings, where larger dilation (2, 3, 5 or 2^l) perform worse than 1.
- **Batch Norm, Dropout, Stride, Channels and Kernal size**[6]: We adapted these settings from Esling's[22] and a slightly smaller kernel that we later showed to perform better.

The comparison of CNN1 and FRKN-Enc is summarised in table 3.

Enc block	Kernel	Stride	dilation	Padding	C_in	C_out	Activation	Batch Norm
CNN1	7	2	2^l	$3 * (dil + 1)$	1 or 128	128	Elu	Yes
FRKN-Enc	5	2	1	1	1 or 128	128	Elu	Yes

Table 3: Comparison of the two encoders CNN1 from[22] and Frankenstein's FRKN. The configuration of CNN1 is best recovered from Esling's original code that includes this large dilation at later layers. The Padding size is used to accommodate the large Dilated Kernel at later layers

[6]These are common terminologies in machine learning, some detailed explanation can be found here: `https://goodboychan.github.io/python/datacamp/pytorch/deep_learning/2020/07/29/02-Using-Convolutional-Neural-Networks-in-PyTorch.html`

The classification layers of ResNet50 and VGG16 were removed, so the convolution output is directly connected to the FC layers described above. Note by convention that the output of variance is log var; to use it later, it needs to be converted to std by $std = exp(0.5logvar)$ in the sampling process.

Decoder Prototyping By common practice[51], the decoder usually behaves as a mirror of the encoder in a reversed way to best reproduce the input. Upsamping and ConvTransposed were used to replace Pooling and Convolution[54]. The out-padding argument was configured properly so that the input dimension matched the reconstructed image$P_{\text{out}} = (size_{\text{in}} - 1) \times stride - 2 \times P_{\text{in}} + (size_{\text{Ker}} - 1) \times dil + 1$, where P_in and P_out are the padding and out-padding, respectively.

Regression Network Prototyping We perform parameter estimation with a three-layer Multi-layer-Perceptron(MLP) as it is the easiest and most straightforward to implement. This is adapted from Esling's[22] and with indicated future improvements noted in section 3.3.10. The latent variable is first sampled from the mean and the variance that is estimated by the FRNK's Encoder. This Latent variable is sent to the MLP network to estimate the parameter.

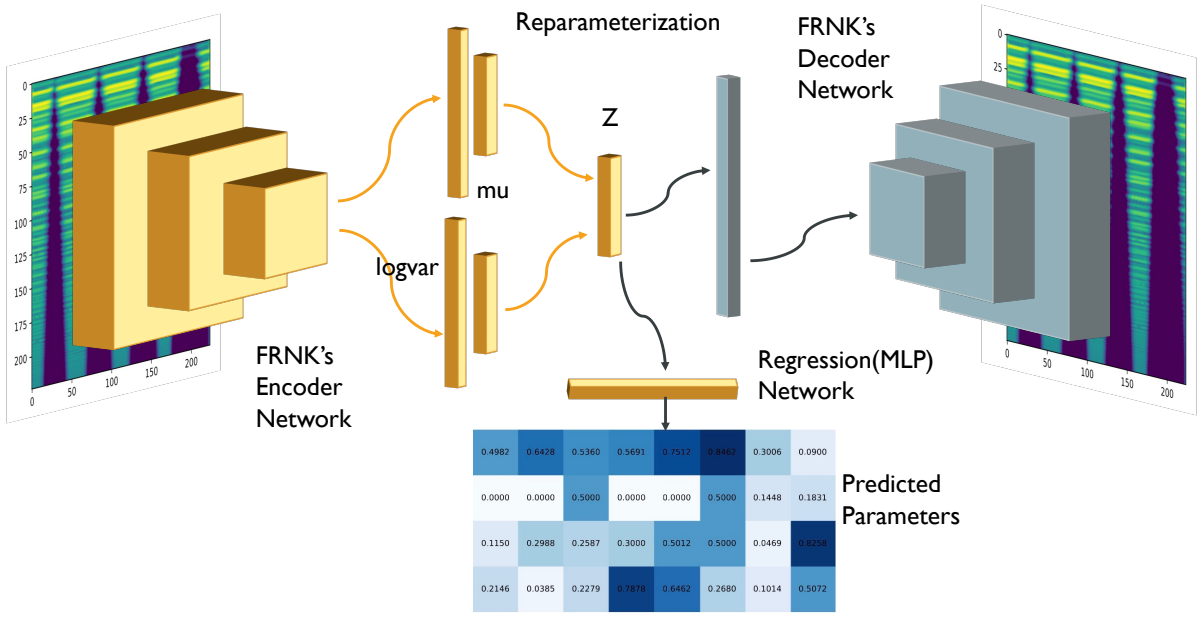


Figure 19: Frankenstein's Parameter Matching Neural Network(FRKN) Overview. Reparameterisation means we sampled the latent variable Z from the mean and variance from the encoder's output.

FRKN Summary As shown in Fig.19, the Mel-spectrum first passes through the encoder of our choice, and then the μ and $log\sigma$ for the distribution of a latent variable z (intuitively a "compressed image") $p(z|x)$

is determined. A sample of z is drawn from this distribution and follows two paths, one to the decoder for the reconstruction of the input (unsupervised) and another one goes to the MLP network for parameter prediction (supervised).

3.3.8 Parameter Matching - Neural Network Training

We performed training on the said architectures in the previous section. We show that they converge and also demonstrate some training strategies to improve the training results.

Encoder-Decoder training criterion In[51], the evidence lower bound (ELBO) is used to assess the performance of the Encoder and Decoder by considering both the quality of reconstruction at the output of the decoder and the distribution of the latent variable. The latter also acts as a regularisation on the network's weight learnt during training to prevent over-fit and exploding gradient. ELBO loss with equation 1, is used in our training process, where β is a hyperparameter we can arbitrarily select.

$$\mathcal{L} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z)) \tag{1}$$

Regression MLP network training criterion We use L1 normalised MeanSquareError (MSE) (reason explained in section 3.3.9): $\frac{1}{n}\sum_{i=1}^{n}[(y_i - \hat{y}_i)^2/\sqrt{E}]$ where the normalisation constant E is the mean energy of one parameter across 11k samples. We compare the prediction $\hat{y}_i$ and the target y_i.

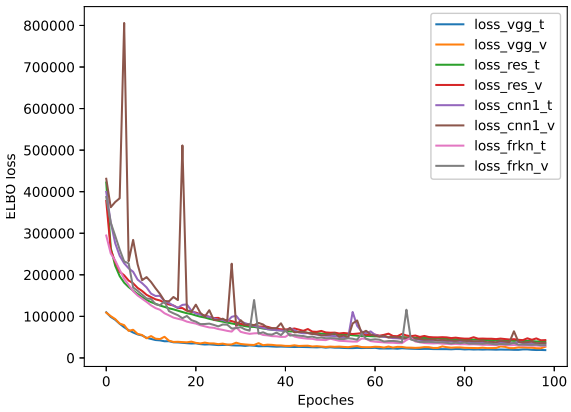
Neural Network Training Configurations For all models, there are **two main training stages**. We schedule[55] the learning rate for different parts of the network at different stages. We also manually reduce the learning rate for both parts of the network when the loss plateaus.

During the **first stage**, the Encoder and Decoder(Enc-Dec) pair were trained in an unsupervised way without parameters. ELBO loss is used with a learning rate of 0.0001. The proportion β of KL divergence and reconstruction loss of ELBO can be tuned. The learning rate of the FC layers is set to 0 as they are not trained at this stage. This guarantees the Encoder and Decoder are trained properly without noise from parameter labels at the early stage. We obtaine convergence before the 100th epoch.

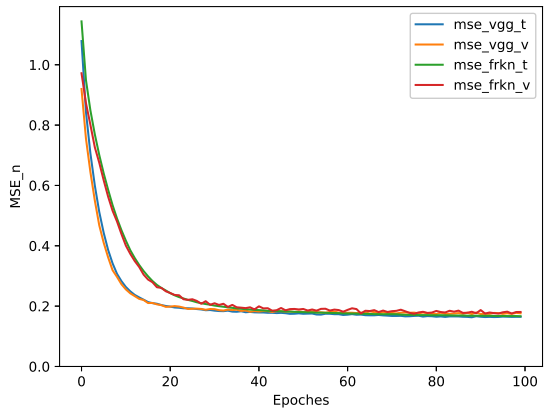
During the **second stage**, the loss is replaced by a combination of ELBO and normalized MSE loss. their proportion is adjustable as a part of the hyperparameter tuning process. Learning rates are also reconfigured for the Enc-Dec pair and MLP. A relatively lower learning rate is used for the Enc-Dec pair

due to the “fine-tuning purpose,” and we choose 0.0002 for the learning rate of the MLP network.

We plot The losses against epoch during the training process in figure 20. Adam Optimiser[56] is used to train the network with 200 epochs at batch size 400 on one Nvidia A100, taking roughly two hours for one training set(9000 pairs of Mel-spectrum and parameters).



(a) The plot of ELBO loss for 4 different architectures. This is the first stage.



(b) Normalised MSE for VGG and FRKN. This is the second stage

Figure 20: Loss vs Epoch for VAE and regression network. The loss ends with t is the training loss, and v is for validation loss.

Hyper-parameter Tuning During the training process, in order to obtain the best possible outcome, hyper-parameters like the loss ratio of ELBO and MSE_N, the ratio of reconstruction loss and KL divergence in ELBOβ, number of epochs, learning rate, batch size, choice of activation functions, number of convolution layers and others mentioned earlier need to go through a careful and rigorous testing process. For these hyperparameters, there is some general searching technique, including the widely used random search by Bergstra et al.[57], and Grid Search (exhaustive search of the entire hyperparameter space), which was claimed to perform worse by Bergstra. Some methods involve Bayesian Optimisation[58] is used to help determine the hyperparameters as described in [59] or tools like Ray-Tune[60]

K-Fold K-Fold is a model evaluation and selection technique that splits the training set and validation set multiple times and evaluates the model performance on the average of these.

Early Stopping Early stopping helps reduce the over-fitting problem of the model. The typical observation is that the training loss keeps reducing while the validation loss on the other hand start rising. Therefore we can choose a point to stop the training before over-fitting takes place.

Additional Regularisation Some additional regularisation terms might help improve performance. These include L1 and L2 regularisation and Dropout. L1 and L2 are penalty terms for large weights in the network. We set Dropout = 0.3 such that 30% of the weights are randomly set to 0 in each forward run.

3.3.9 Parameter Matching - Neural Network Performance Evaluation

In this section, we show that the proposed Frankenstein's FRKN is the better option to be the neural network in the final design and address some design choices we made in earlier sections. The performance of the Enc-Dec pair and the regression network are evaluated separately. We also include a final test with recordings from Diva VST with the parameter we predicted earlier, thus validating Parameter Matching.

Evaluation on Enc-Dec part of all 4 different VAE models The performance comparison is shown in table 4, with ELBO and L1 normalized MSE loss on reconstruction spectrum and parameter prediction.

Model	ReconLossTrain	ReconLossTest	MSE_nTrain	MSE_nTest	ELBOTrain	ELBOTest
VGG16VAE	0.024378	0.033898	0.15968	0.17559	20994	26256
Res50VAE	0.052039	0.059923	0.17895	0.18445	40171	44467
CNN1VAE	0.047279	0.052701	0.17580	0.18392	33628	36068
FRKN	0.036375	0.041136	0.16114	0.17813	28532	30669

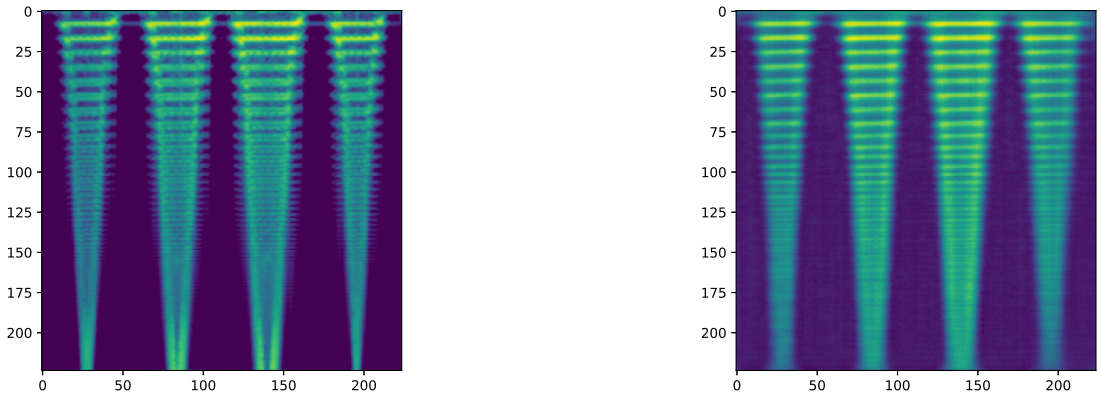
Table 4: Reconstruction, ELBO, MSE_n loss for 4 different VAE based model

In figure 20, the ELBO loss and default normalised MSE loss are illustrated (Note that all 4 models share the identical regression MLP network for a fair comparison). It can be seen that both of these two converge at the stage-switching(in section 3.3.8) point (before 100 epoches), and the validation loss remains close to the training loss, so there is no significant issue of over-fit. Although the VGG Encoder-Decoder pair yields the lowest ELBO and reconstruction loss, FRKN is not far off. This can also be seen from the MSE_n loss plot. Slower convergence might illustrate that the quality of latent variable learned from VGG contains less noise, but they converge at roughly the same point, such that the limitation is contributed more from the regression network instead of the Enc-Dec network.

Neither Res50VAE nor CNN1 of Esling's[22] performs well in our training. We acknowledge different data set pre-processing and training configurations can lead to different results. But under the scope of our experiments **VGG16 and FRKN perform better overall the architecture of VAE**

Encoder-Decoder Reconstruction Quality We can see that the reconstruction of FRKN's Enc-Dec pair is acceptable as it reconstructs the original input image, as shown by the comparison in the figure

21 One of the key characteristics of using AutoEncoder-like architecture is the ”blurring” or ”smoothing” effect, as the wiggling property of the original Mel-spectrum is lost.



(a) The original input Mel-spectrum image

(b) FRKN reconstructed Mel-spectrum image

Figure 21: Reconstruction quality. Remark that the reconstructed image(b) is not the recreated audio. It only tells the quality of the latent variables(the “compression” of the input image) and the effectiveness of the Enc-Dec pair. The input image has a dimension of 224x224. The closer the reconstructed is to the input image, the better the performance of Enc-Dec.

The reconstruction quality and robustness can be further improved by training with a larger data set with out-of-domain data. As previously argued, the objective of the encoder-decoder pair is to learn a representative and universal audio space that extracts characteristics of synthesised audio effects, which we have achieved.

Size of Latent dimension It is another hyperparameter we can optimise on. A test at a lower input dimension was carried out to decide the size of the latent dimension in this project. The objective of this is to address the compromise between the reconstruction quality and regression noise. An attempt is made to reduce the latent dimension to 32 to see whether it can help the regression network. The result indicated that the performance using 32 latent variables is worse than using 64 variables both for parameter estimation and Enc-Dec reconstruction. Their comparison is summarised in the table 5. It can be seen that a larger dimension yields better performance. 64 is the largest dimension used in Esling’s [22]. We can perform testing on higher dimensions in future to further improve this result.

Evaluation of Regression Network In figure 20 (b), we have shown convergence can be achieved. This indicates that the training is effective. While we also acknowledge that a more sophisticated design of the regression network can be made, we believe it is sufficient to validate Parameter Matching with

Model	ELBO Loss Train	ELBO Loss Test	MSE_n Train	MSE_n Test
VAE32	32558	22057	0.2103	0.2139
VAE64	24196	16251	0.2047	0.2082

Table 5: VAE Latent Comparison for CNN1VAE at for input Mel-Spectrum image at 128x87, VAE64 and VAE32 has 64 and 32 latent variables respectively.

FRKN. For the MSE loss normalisation, we can either use MSE loss directly without normalisation or apply normalisation (L1 or L2) across these 11 thousand samples. The result is shown from figure 22 to figure 25, test performed on FRKN. In theory, normalisation is performed such that each parameter is treated evenly without the influence of their individual variance. If one of the parameters tends to have larger energy(variance) most of the time, its MSE might dominate that of smaller ones. Tests are performed using different normalisations. Their difference can be seen from the parameter plot in some sense, but better to be heard from their audio files in the shared folder.

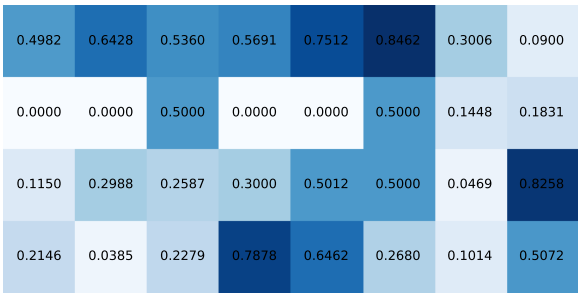


Figure 22: Target(A)

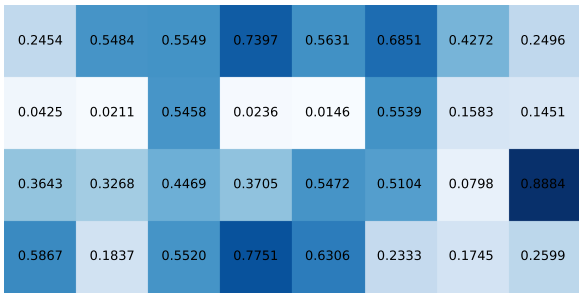


Figure 23: Prediction with no normalisation(B)

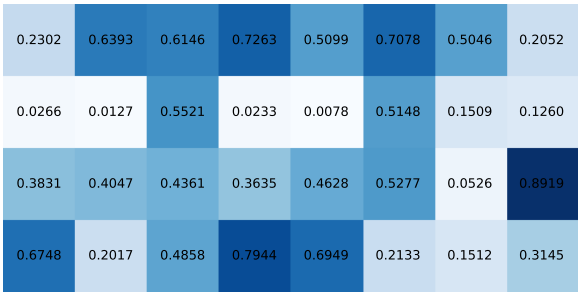


Figure 24: Prediction with L1 normalisation(C)

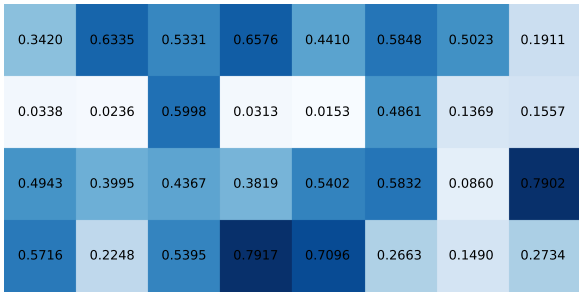


Figure 25: Prediction with L2 normalisation(D)

Each square corresponds to a synthesiser parameter, the total 32 parameters are formatted into a 4x8 image as shown

Evaluation on Audio of Matched Parameter with Diva VST We use Diva VST to recreate the audio based on the parameter predicted and convert the output audio to Mel-spectrum for comparison, as shown from figure 26 to 29 (The output audio from Diva-VST using predicted parameters). The difference might be best to be heard to be noticed. L1 normalisation balances MSE at different signal energy levels,

which also best recreates the Target spectrum. **L1 normalisation is therefore used in Frankenstein**.

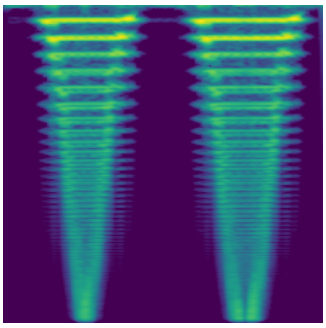


Figure 26: Target(A)

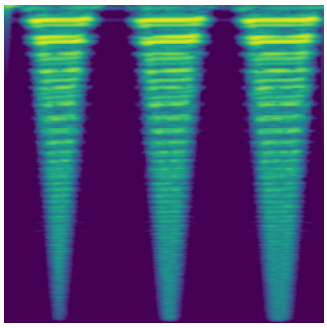


Figure 27: No norm(B)

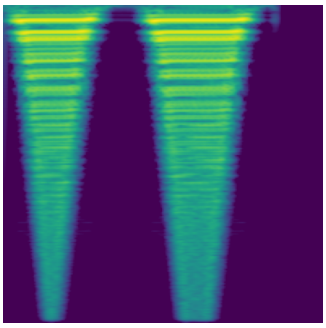


Figure 28: L1 norm(C)

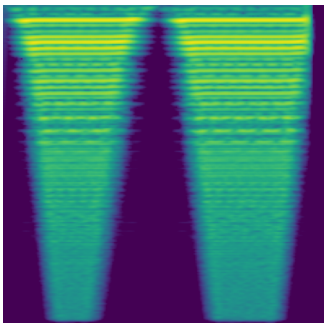


Figure 29: L2 norm(D)

These four Mel-Specturms images correspond to figures 22 to 25. The close to the target, the better the parameter matching. There is a horizontal shifting due to a slight difference in the starting time of recording. We care about their shapes only. Note that The No norm result has higher low-frequency oscillation(LFO) than the rest.

Memory and Model size From previous comparisons, VGG16 seems to outperform other Encoder-Decoder designs, but its heavier size might limit its applicability. The number of weights in a model, training time, execution time, and memory consumption(expected) are listed in the table 6. The high re-

Model	Number of weights	Training Time/s	Execution Time/s (M1-Pro CPU)	Est. Memory (MB)
VGG16VAE	85,336,609	8,398.71	0.53637	325.35
Res50VAE	58,592,801	3,464.56	0.26177	223.57
CNN1VAE	52,118,435	2,719.79	0.23246	198.85
FRKN	**14,084,387**	**1,877.13**	**0.11188**	**53.72**

Table 6: Model Resource Comparison. Remarks: The Audio input is a 4-second audio effect, with audio specs listed in the data set section. Execution time is measured on a personal laptop, which may be dependent on system load. Time used an average of 5 runs on Macbook Pro 2021 with an 8-core CPU. Memory is calculated based on the number of parameters, with 4 bytes per parameter(FP32). The actual size depends on the specific version of Pytorch

construction quality of VGG16 comes at the cost of using more memory and longer computation time. As expected, larger models require more memory and time to both train and run. Balancing the performance and resources required, as well as the design's practical concern, **We can conclude that the proposed FRKN performs the best overall and is used in Frankenstein.**

Practical Advantages The separation of the Enc-Dec's reconstruction and parameter prediction brings flexibility to Frankenstein's architecture. The Enc-Dec and be unsupervised trained with out-of-domain sounds that don't come from an analogue synthesiser, for instance, NSynth Data-set, which helps resolve

the limitation in section 3.2.2. Our universal input dimension also makes adaptation and training with different backbone architectures easier.

3.3.10 Parameter Matching - Limitations and Future improvements

In machine learning, it is particularly challenging to find a network structure that has the best performance. The selection of neural network architecture is usually based on the experience of the researchers and a large trial-and-error process. The usual technique is to perform Neural Architecture Search (NAS), such that the performance from different architectures is compared against each other. The methods include random search[57], Bayesian optimization[58]. Under the scope of our project, it is both time and resource-limited to try this method. The same applies to hyperparameter tuning. We believe our performance can further improve if we have more resources available. The experiment setup can be improved by running the same setting multiple times to achieve a more robust result.

It would be better if we could compare the input sound and output sound directly. However, in section 3.3.8, we see that training is done by comparing the model's predicted parameter with the true parameter. If we want to compare the input sound with the synthesiser's output sound from the predicted parameter, we need to incorporate a synthesiser's VST inside the training loop, which is very difficult to achieve. But this is possible if we provide an interface to access the VST during the training procedure, so the matching target will be directly the recreated audio, potentially further improving the performance.

3.4 Parameter Matching Summary

Parameter Matching using FRKN yields great performance with an overall normalised MSE_n of less than 0.18 (the parameter generates output audio that successfully captures the input characteristic) while maintaining a lower memory and computation requirement (about 0.1 seconds on a personal laptop). Despite the limitations mentioned earlier, it is sufficient to say that the parameter-matching prototype is working and can be used by Frankenstein's VCO-VCF-EG module. We also see room for improvement to be done as a future development plan when this product receives funding as planned in the later finance section. Wrapping everything up, we hope the parameter-matching module can make beginners' life easier. Together with other modules of Frankenstein, we embrace an even more inclusive synthesiser community.

4 Module 2: Instrument synthesis and Timbre transfer - Haedam Oh

4.1 INSTRUMENT module overview

4.1.1 System Description

The main goal of this module is to synthesise musical instrument sounds. Digital synthesizers are not able to generate instrument sounds very well due to the complex nature of timbre with the limited number of oscillators and computing powers. With the advance in machine learning algorithms, an INSTRUMENT Module will provide the users not only a variety of musical instrument sounds but also real-time timbre transfer to a target instrument. In this report(See Figure30), we are going to discuss (1) each instrument's sound synthesis and evaluation(Figure30 **Step1**) (2) timbre transfer and expression controls(Figure30 **Step2**) both real-time and offline(Figure30 **Step3**).

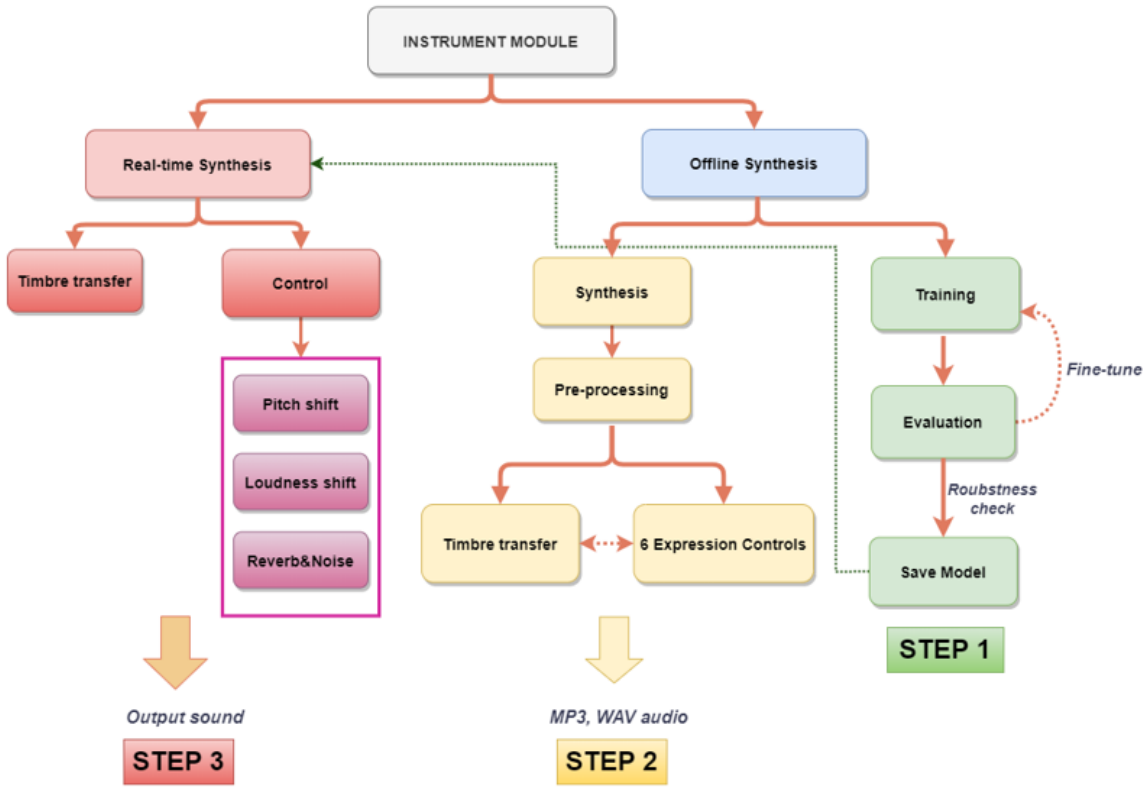


Figure 30: Work Breakdown Struture, **Step1**: musical instrument sound synthesis and evaluation of sound quality. **Step2**: controllability inside INSTRUMENT Module, mainly Timbre transfer and Expression Control. **Step3**: a real-time synthesis with hardware-software interaction.

4.1.2 Software Interface (VST Plugin)

- **Real-time and offline synthesis**(See Figure33 Left up corner): As mentioned in Figure30, our model performed both online synthesis and offline synthesis. Online real-time synthesis is done when user provides their own sound input through hardware or by their musical instruments. In offline synthesis, where we provide additional instrument models with the computing power of GPUs.
- **3 low-level(basic) controls**(See Figure33 UI): User can preprocess the input audio by using three controllers: Pitch shift, Loudness shift and Reverb&Noise. Pitch can be shifted by 1 octave from the highest to the lowest pitch of input audio. Loudness is scaled 0 to 1 so that user can design their own loudness per frame. Reverb and Noise module can add or remove the effect of Reverb and noise present.
- **Timbre transfer**(See Figure33 UI): This preprocessed input can be delivered to Timbre transfer where you can select one of the recommended instruments to change the timbre of input audio.
- **6 high-level controls**(See Figure33 Left box): In offline synthesis, with MIDI input, detailed controls on high-level features such as (vibrato, attack, peak, mood, volume, and fluctuation) can be controlled at note level.

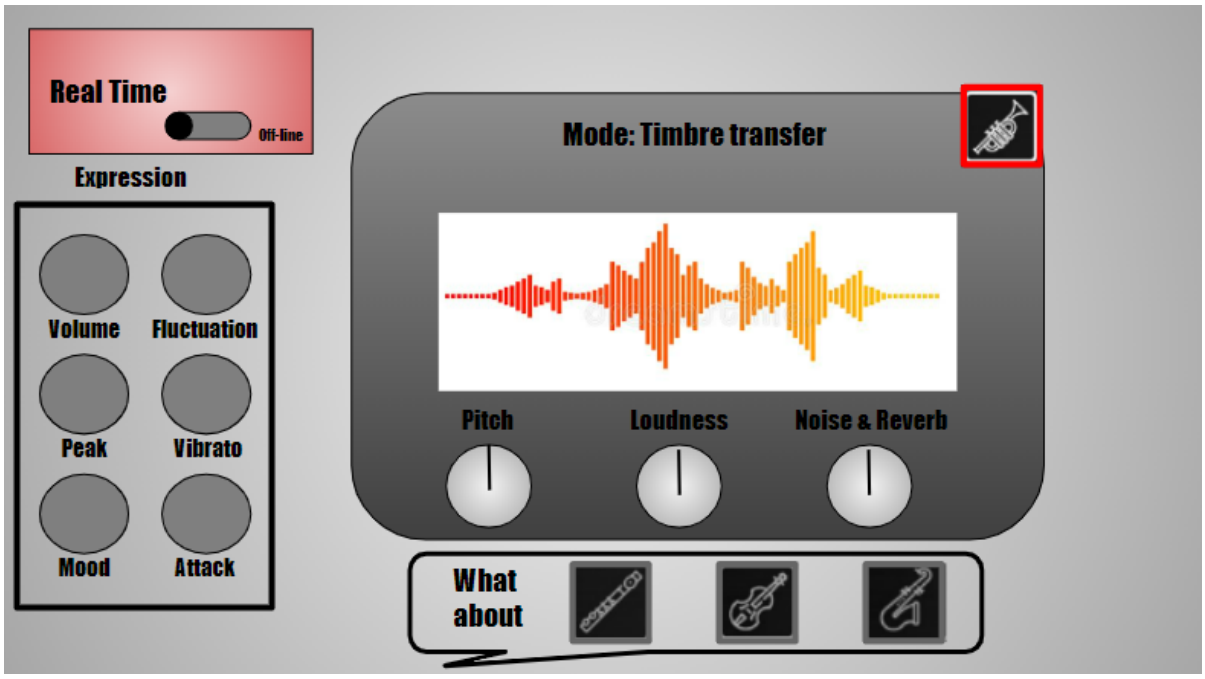


Figure 31: Software Interface of INSTRUMENT Module

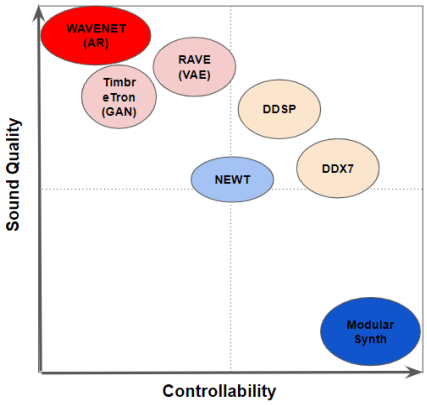
4.2 Instruments synthesis: algorithms and methods

4.2.1 Classical synthesis techniques

- Additive synthesis Section2.2.1: linear combinations of an integer multiple harmonics with varying amplitudes
- Subtractive synthesis Section2.2.2: filtering of complex sounds to shape harmonic spectrums changing envelopes or ADSR.
- Physical modeling[61]: mathematical equations of acoustic characteristics of an instrument
- Sampling[62]: Per-frame sound or seed waveforms are upsampled to high-quality sound(44.1kHz) with articulations and expressivity present.

Models	Synthesis	Sound Quality	Timbre transfer	Real-time Inference	Traninability e.g) How long for training GPUs resources	Memory e.g) how many parameters	# Instruments	Controllability e.g) ADSR, pitch, amplitudes..
DDSP	Additive	4	4	Yes	3h-12h	4M	4	Yes
DDX7	FM	4	3	Yes	a day	4M	3	Yes
NEWT	Waveshaping	4	3	Yes	1h-6h	300k	3	Yes
RAVE	Sampling	5	5	Yes	a week	17M	4	No
Timbretron	Sampling	5	4	No	a week	10M	4	No

(a) Models' performance summary



(b) Sound quality vs Controllability

Figure 32: Model selection process: important factors to considers: controllability, sound quality, trainability, and memory

4.2.2 Using machine learning in music synthesis

Many machine learning(ML) models in musical instrument sound synthesis are built upon traditional synthesis techniques[63, 64, 65, 66, 67]. Figure32 summarises different models and criteria for model selection. DDSP outperformed in controllability and trainability. Thus we choose DDSP[67] as a baseline model of the *INSTRUMENT* module.

DDSP Model Architecture

- **Synthesis techniques**: DDSP uses additive synthesis on harmonics: $\sum_1^P a[n]\sin(\varphi_p[n])$,and subtractive synthesis of Linear Time Invarient(LTI) filter response. These components are built on SMS

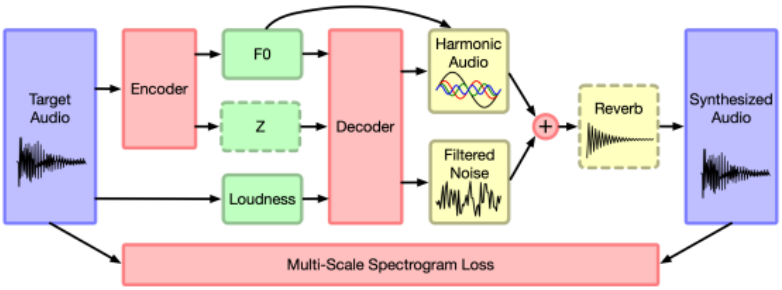


Figure 33: DDSP autoencoder

synthesis[68]

- **Encoder**: captures *fundermental frequency(F0)*,*loudness* and, *timbre(z-embeddings)*. It down-samples audio raw audio input to those three latent variables *F0,Z,Loudness*(Figure33 green). The objective is to learn features of input audio (mainly frequencies and timbre).
- **Decoder**: upsamples latent variables in order to learn the 1) harmonic distribution and 2) filter impulse response of per frame sound. Those two are summed up with learned Noise&Reverb to generate output sound at the end.
- **Latent variables**: F0,Loudness and Z-embeddings(timbre) are latent variables we can directly controlled. F0 is extracted by pretrained [69], Loudness[70] and z-encoder(MFCC30-batchnorm-Resnet-Dense)[67].

Instrument	Types	Genre	Available Dataset
Violin	Bowed-string	Western, Orchestral	URMP
Flute	Woodwinds	Western, Orchestral	URMP
Trumpet	Brass	Western, Orchestral	URMP
Drum(Bass,snare,kicks)	Percussion	Everywhere	DRUMSET
Piano	Keyboard	Western, Orchestral	Piano Triads
Guitar (acoustic)	Plucked-string	Pop, Jazz, Funk	GuitarSet
Danso	Woodwinds	Korean traditional	-

Figure 34: Chosen instruments: ranked by popularity according to survey[71].

Instruments selection

What musical instruments to provide for users are difficult decisions to make. Since our product Frankenstein target people who have a musical background but are not familiar with synthesis techniques, we have selected 6 different musical instruments: Violin, Flute, Trumpet, Guitar, Piano, and Danso, by types, popularity and synthesis performance (summarised in Figure34). We would like to show the integrative

Conditions/ Product Objectives	Synthesis-time	Memory	Output sampling rate	Overall Sound Quality (RMSE)	Input audio size & n_frames	Expressivities, articulations generation
1. Real-time synthesis ***(Personal device)***	Real time factor < 1 Synthesis time <500ms	Minimum 8GB RAM	16kHz	RMSE<5.5	1s~4s audio per input. (16khz) n=512 frames	Few present
2. Offline synthesis ***(Server-side computations)***	Real time factor<0.5 Synthesis time< 30s	Minimum 16GB RAM	44.1kHz	RMSE < 5	Maximum 5min .MP3 n=1024 frames	Most present
3. Training Instruments ***(Server-side computations)***	Training time: 12H~24H	Minimum 32GB RAM	44.1kHz or 16kHz	Training R MSE< 5.5	0.5-1h audio for training. n =1024 frames	All present
4. Timbre transfer ***(Personal device)***	Real time factor<0.8 Synthesis time <100ms	Minimum 8GB RAM	16kHz	RMSE<6	1s~4s audio per input. n=512 frames	Few present

Figure 35: Performance Objectives: 4 different scenarios, and different weights on different objectives to achieve. 1.Real-time synthesis: synthesis time, 2.Offline synthesis: sound quality, 3.Training: memory and training time. 4. Timbre transfer: synthesis time

synthesis ability of different musical instruments(not limited by types) in *INSTRUMENT* module.

4.2.3 Performance Objectives

In addition to general requirement in Section1.3, Figure35 summarises the initial target specification in *INSTRUMENT* model. In real-time synthesis and timbre transfer, we would emphasise more on synthesis-time. Real-time factor(= $T_{Synthesis}/T_{Duration}$) should be below one to convey audio without delay in real performance. 500ms is a maximum delay we set. In offline synthesis and training phase, we are more interested in generating good-quality sound than time it takes. Therefore, we target *Reconstruction loss*(RMSE) strictly less than 5 for evaluation.

4.2.4 Market research on competitors

Figure36 summarises current market products of instrument synthesisers. Although our model will generate lower-quality sounds compared to current synthesisers in the market, our model will provide better controllability of sound, a variety of different instruments (not limited by types) and easy mechanisms targetting beginners in synthesisers. Also, we provide *subscription plan* meaning that basic functions are *free*, but premium subscribers can have access to the server, where they can use more instruments in database, and have faster synthesis time with GPU available. This is more cost-efficient from user's perspective otherwise users may need to spend few hundres pounds just for the start.

Instruments	Violin	Flute	Piano	Guitar	Drum	Acoustic Instruments
Software/ VST Plugin	*Omnisphere2*	*Roland Aerophone AE-10*	*Pianoteq, Keyscape*	*Ample Guitar*	*Ableton Drum Rack*	*Garritan, Flowsynth*
Pros	- high-quality sample-based sounds, 20 options of types. - Control parameters: attack, release, timbre, and resonance	- a wide range of woodwind types - Portable - Controls: vibrato, bends, and glissandos using finger sensors	- Customization hammer hardness, resonance - lightweight and efficient in terms of CPU and disk space	- Realistic Sound - Articulations, strumming styles, fret noises - Wide Range of Guitar Models	- Arragement of drum hits, allowing for quick and efficient drum programming. - Drum Rack allows for extensive sound design capabilities	- Good Sound Quality - User-friendly interface
Cons	-Hard to learn for beginners -Mainly CPU Intensive	- Require prior knowledge in playing basic woodwind instruments. - High cost	-Subjective Sound Preference -Limited to non-Piano Sound	-No Physical Interaction -CPU and Disk Space Usage	- Sample-Dependent Sound Quality - No Built-in Step Sequencer	- Limited Customization - Limited Non-Orchestral Options
Costs	319 EUR	580 EUR	139.00 EUR for 2packages	120$ per package	Free	*Garritan* : $149.95 for 5 packages *Flowsynth*: Free
Synthesis techniques	Sampling based	Physical modelling + Sampling	Physical modelling algorithms	Sample-Based Synthesis, Round-Robin Sampling	Sample-based, wavetable	Sample-based Synthesis

Figure 36: Market research on current popular synthesis software: synthesisers by instrument type and a single synthesiser which integrates orchestral instruments, most of them used *Sampling-based techniques*. Advantages: good sound quality. Disadvantages: heavy computations, a lack of controllability of sound, not beginner-friendly.

4.3 Training and Evaluation of instruments

Training Overview

The objective of training is to obtain high-quality, but also robust audio waveform to real-world audio data such as YouTube. DDSP trained on URMP dataset[72] showed good performance in synthesising different types of instrument sounds [67]. However, the model suffers in real-world performance. This is because of the presence of noise and reverberation, uncertainty in unseen pitch and timbre, and the presence of high level features such as articulations and dynamics by professional musicians. Therefore, two-stage training procedures are used low-level learning for pre-training and high level for fine-tuning. We showed proof-of-concept [7] as a reference for our own evaluation experiments and improvement in model's performance. Furthermore, we trained and evaluated more various instruments such as Piano, Acoustic guitar, Drum and Danso to integrate different types of instruments in one platform.

1. **Pre-train** each instrument on note-level, monotonic dataset making sure each instrument learns own notes, chords, and timbre, but not high-level features such as articulations, expressions, and dynamics of music. Therefore, in general, dataset should cover the full range of pitch with a slow

[7] https://github.com/HaedamOh/3YP-Music-synthesis-.git

tempo and no articulations present.

2. **Fine-tune** the pre-trained model to improve robustness and regularisation according to [73]. The goal is to improve robustness against extremely high-to-low-pitched music in the presence of many articulations and dynamics. The pre-trained model would act like a pitch tracker and synthesized music would sound like simple note tracking performed by a beginner. We would like to synthesise much high quality of music performed by professional musicians. Therefore, fine-tuning existing sheet music performed by professional musicians improved the sound quality of music, with articulations and dynamics present in the resynthesised output.

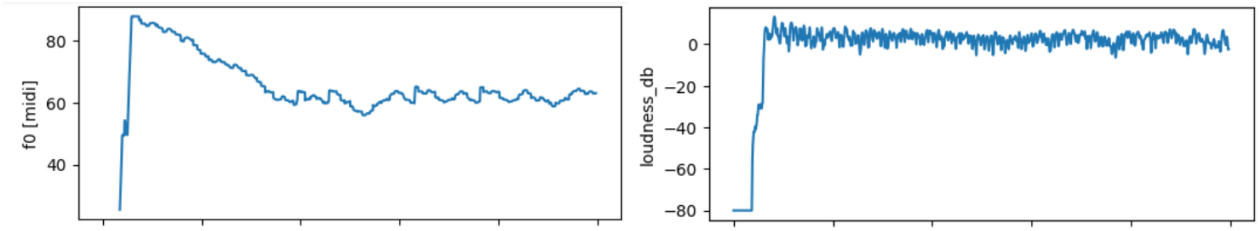
Evaluation Metrics

- **RMSE**[67](Reconstruction Mean Square Error): refering *Multi-Scale Spectral Loss*: $L_i = \left\|\left\|S_i - \hat{S}_i\right\|\right\|_2 + \alpha \left\|\left\|\log S_i - \log \hat{S}_i\right\|\right\|_2$
- **DTW**[74] (Dynamic time warping): Ground truth and synthesised output are two feature vectors, then find cost matrix C between them and use dynamic programming to identify an alignment path P that minimises the cumulative cost. Used for measuring alignments.
- **IS**[75] (Inception score): the Pitch Inception Score (PIS) and the Instrument Inception Score (IIS) [8]. These measures tell us if the generator has captured the true distribution of discrete MIDI pitch classes.
- **FAD**[76] (Fréchet distance): with a pre-trained VGGish model, both background and synthesised output embeddings are extracted. Then FAD is computed between multivariate Gaussians estimated on these embeddings. It reflects the human perception of sounds at a high-level.
- **Subjective evaluation**[77]: Mean Opinion Score(MOS), listeners scale between 1 and 5 on sound quality, timbre, pitch, etc.

4.3.1 Violin

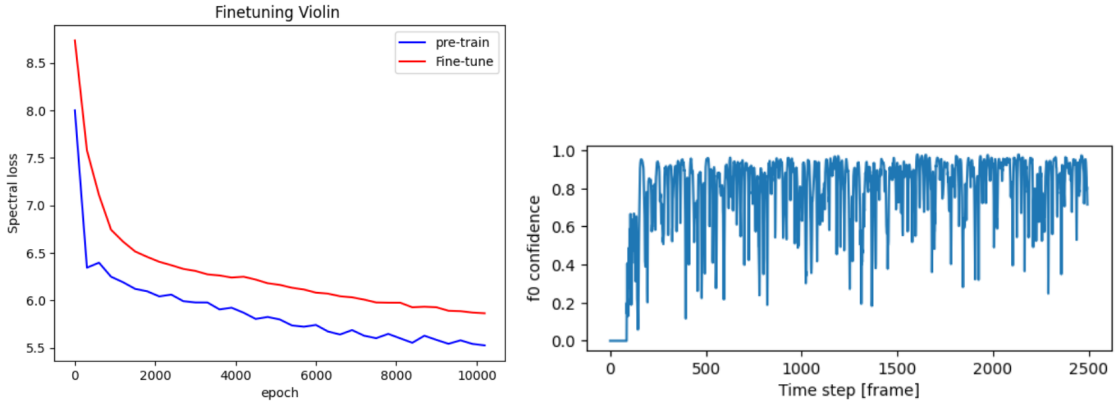
Training and Evaluation

The aim of training is to verify the robustness of pre-trained DDSP violin model[67] by fine-tuning on real-world data. This improves regularisation and robustness by *transfer learning*[76]. We collected total 6 minutes of audio in YOUTUBE, with small reverb and noise present. This data should cover a broad range of pitch, tempo and expressivity performed by professional violinists. The evaluation was done on



(a) Fundamental frequency(F0) distribution (b) Loudness

Figure 37: **Audio feature extraction**: *flight of bumblebee.mp3*, F0 extracted with *CREPE*[69] and Loudness with [70] to generate per frame. Frame size=2048, hop size= 512, 4ms per frame.



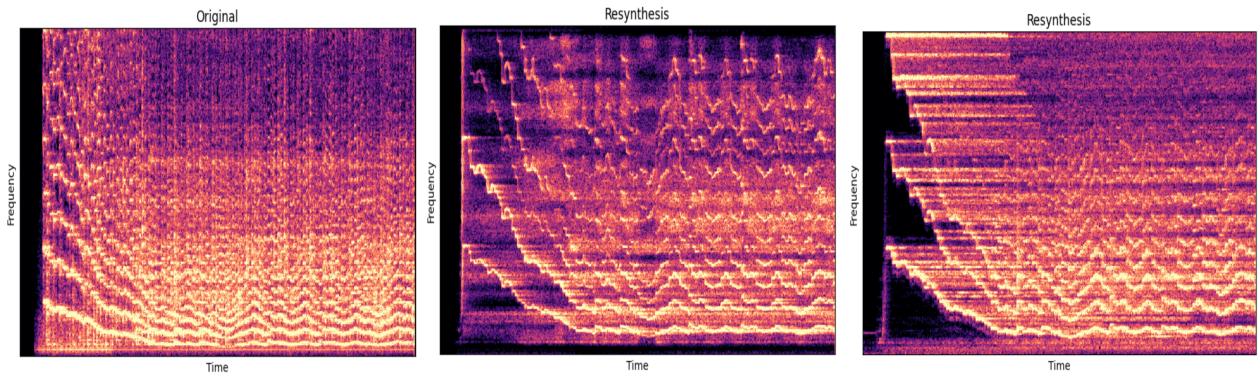
(a) Training loss curve: violin (b) F0 uncertainty from *CREPE*

Figure 38: Training scheme: *Batch size*=16, *Adam optimizer*, *Learning rate*=0.001, *decay*=0.01, *Sampling rate*=16kHz, fine-tune with single GPU *RTX A4000* for 6h. When fine-tuning, the encoder was freezed and only the decoder was fine-tuned. Fine-tune audio: *Csárdás, Devil's Trill Sonata*

*Flight of Bumblebee*Figure39. This is unseen data with extremely high pitch and fast dynamics present. This is the case when Figure38(b), CREPE(F0 extractor) cannot detect F0 confidently.

Results summary

Robustness test under uncertainty in F0: Figure38 shows DDSP suffers most when CREPE cannot clearly identify a fundamental frequency at each time step. However, we have shown the improvement in robustness by fine-tuning on the real-world dataset: In Figure39(a), there are 6 harmonics present in first 2s, our Figure39(c) have predicted 3 harmonics more clearly than Figure39(b) removing some the background noise present in Figure39(a). In addition, at the last 4s shows rapid change in pitch with many dynamics and articulations present. Figure39(c) preserved these high-level features whereas Figure39(b) fail to identify them. This shows significant improvement in both robustness and sound quality.[8]



(a) The original spectogram (b) Vanllila DDSP (c) Fine-tuned DDSP

Figure 39: **Evaluation**: resynthesis on 10s of *Flight of Bumblebee*. Notice clear peaks on low harmonics(1st,2nd,3rd) present in (c) than (b), and (c) better estimates noise distribution and high frequencies whereas (b) did not recognise high frequency noise and misinterpret them as low frequency noise.

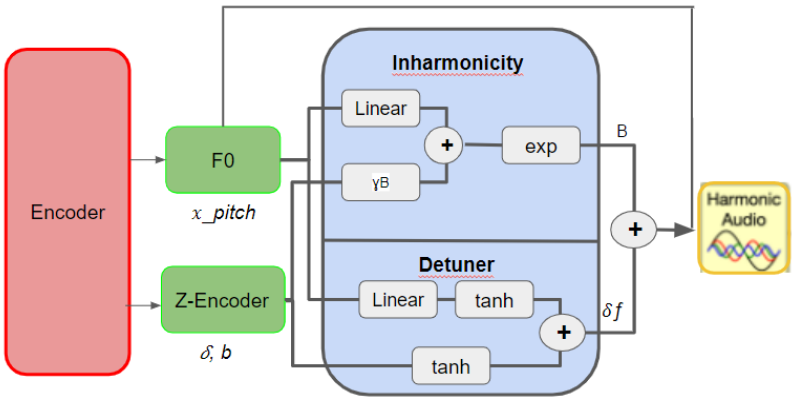


Figure 40: Redesign of Inharmonicity decoder: Now we re-design DDSP timbre encoder (MFCC(timbre)-Resnet14) to account for in-harmonicity embeddings with physical modellings. The Z-Encoder should compute an embedding vector z_i an inharmonicity modifier bi and an instrument specific detuning f_i for each recording step i should be computed.

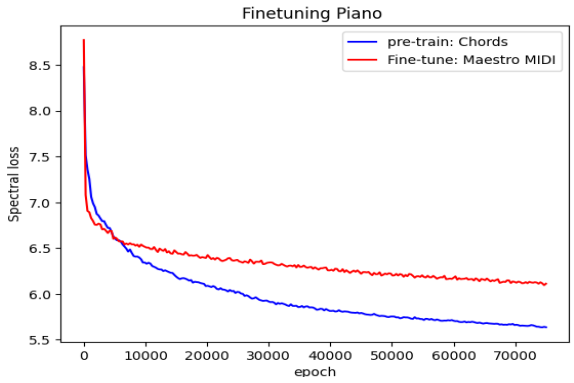
4.3.2 Piano

Piano synthesis techniques

Our DDSP based model on additive and subtractive synthesis is lacking of realism mainly because of the presence of inharmonic distributions. Piano-ddsp[79] showed modelling physical properties of the piano, tuned sounds and ADSR envelope varied with pedal effects. Inspired by Piano-ddsp[79], we have shown our re-designed decoder(Figure40 can synthesise piano(*Grand, Upright* types). We were able to show that our DDSP-based model can also perform equivalently well as other synthesisers in the current market under controlled audio input.

[8]synthesised flight of bumblebee available [78]

Experiment Methods



(a) Training loss

Models / Evaluation Metrics	Pre-train(Vanilia DDSP)	Re-designed + Fine-tuned
FAD	**4.763**	5.795
DTW	0.5248	**0.50996**
RMSE	6.701	**6.320**
MOS	3 / 5	**4 / 5**

(b) Evaluation: Redesigned DDSP(fine-tuned) VS Vanilla DDSP model(pre-trained), redesigned model outperformed except FAD

Figure 41: Two training phase: Our inharmonic decoder weights are frozen and set to initial values[80]: α_B = −0.0847, β_B =-5.82, α_T=0.00926 and β_T=-13.64 This is same as the normal DDSP model learning note envelopes, loudness and noise&reverb. Next, unfreeze the inharmonic decoder as before to learn inharmonics with using loss as $L1 = \sum |[\alpha_B, \beta_B, \alpha_T, \beta_T] - [\alpha_{B_0}, \beta_{B_0}, \alpha_{T_0}, \beta_{T_0}]|$.

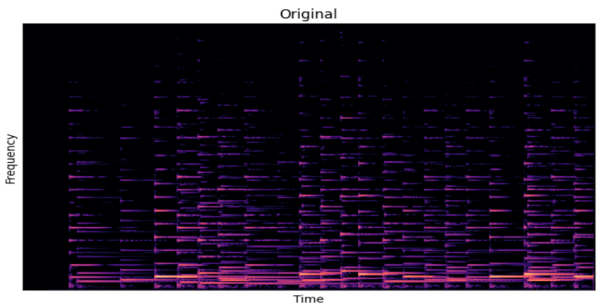
1. **Inharmonicity**[9]: Inharmonicy coefficient estimation at k-partial harmonics due to damping of string can be modelled by [80]. $B_p(t) =$ exp$(\alpha_T x_p(t) + \beta_T) +$ exp$(\alpha_B x_p(t) + \beta_B + \gamma_B b_i)$ where $x_p(t)$ is extended pitch by T seconds until damped to zero(envelope size). α_B, β_B and α_T,β_T are linear asymptotes (hyperparameters) in the treble range, and $\gamma_B b_i$ is the term that influences (modelling parameters) the bass.

2. **Partials and detuning**[10]: As the duplicated strings are slightly detuned from one another, partials beating can happen during performance. For n-strings sub-strings per note, a detuning factor δf of the fundamental frequency is computed by the detuner sub-model, by summing the detuning f_i at each interval with estimated detunings of the n sub-strings. $\delta f_(t) = tanh(g(x(t) + tanh(\delta f_i))$ and $f_k(t) = kf_0(t)\sqrt{1 + B_p(t)k^2}$, where $f_0(t) = 440 \times 2^{1/12(x(t)+\delta f(t)-69)}$ is for MIDI conversion of de-tuned pitch.

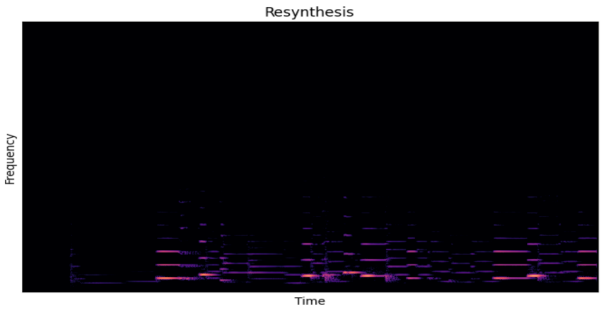
During a performance, the pedals activity lets the string still vibrates for a certain amount of time after the note offset and the interaction between simultaneous notes can change the timbre of an individual note. By carefully controlling input, we intentionally turn off pedal effects to simplify modelling but kept Inharmonicity factor $B_p(t)$ and detuned sound $\delta f_(t)$

Evaluation: Piano

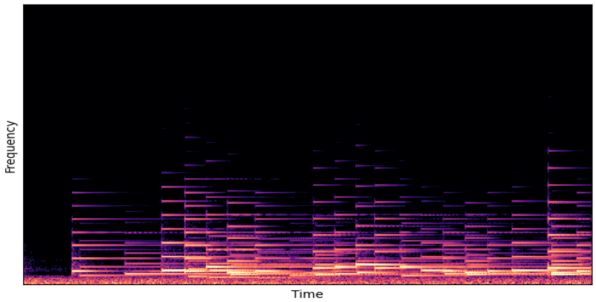
[9]Inharmonicity refers to a slight deviation from the integer multiple harmonics, e.g 2nd harmonics -> 2.2 harmonics
[10]detuning refers to slight out of tune from orginal pitch



(a) Dynamic piano audio song 10 seconds



(b) Pre-trained model: synthesised piano audio



(c) Re-designed(fine-tuned) model: synthesised piano audio

Figure 42: Synthesis done with 10s MastroDataset[cite]:

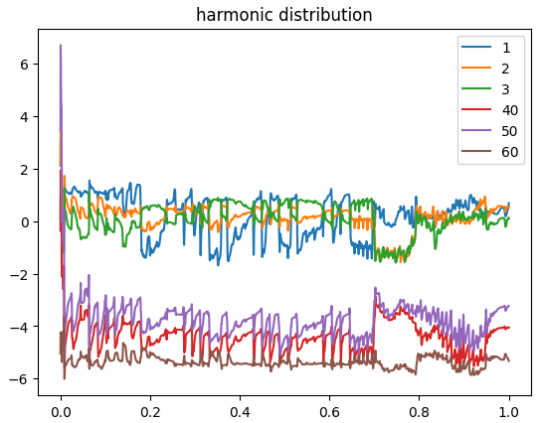
1. **Pre-train: capture low harmonics**

   It is clear that Figure42(b) lost high-frequency information or cannot model them at all. In fact, this is what we aim for pre-training, only integer multiple harmonics should be well modelled by k=0-6 oscillator. Even if we lost high-frequency content, we can still hear similar sounds by human perception. Figure43(a) shows harmonic distributions on low-frequency oscillators(k=1,2,3) dominate output sound, while high-frequency oscillators(k=40,50,60) have negligible amplitudes below -2dB most of the time. Figure43(b) proved our F0 is uncorrupted by noise. Even though reconstruction loss has given quite high as 5.5 on average, the model correctly predicts harmonicity correctly.

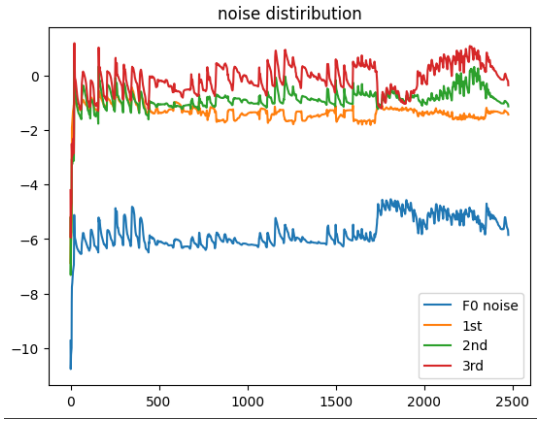
2. **Fine-tune: capture inharmonics and high-frequency distribution**

   Inharmonics: F0 integer-multiple harmonics are still dominant components in the frequency domain, but there are inharmonic components(non-integer multiples) also present (Figure44(b)). These components mainly model 'beating sounds' and they are distributed in the high-frequency spectrum rather than low-frequency. These results show that the model was able to capture inharmonics present at high frequencies.

   Detuner: the effect of detuning(δf) did not significantly contribute to F0 envelopes (44(a)). An estimated detuning factor was δf=-0.2880, which is not a very significant factor in the training stage.

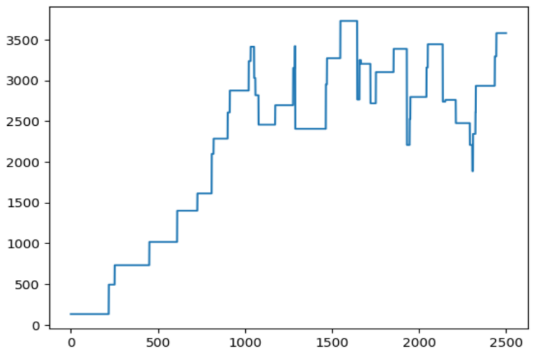


(a) k(low) =1,2,3 harmonics and k(high) = 40,50,60 harmonic distribution

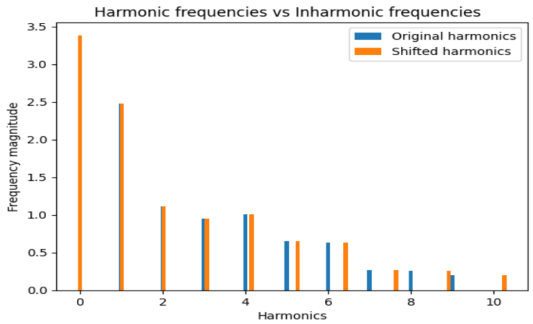


(b) Noise distribution for F0 harmonics and rest harmonics

Figure 43: Harmonics distributions and noise distributions. Note that low-frequency harmonics are more dominant than high-frequency harmonics. Noise present in high frequency not affecting low-frequency contents.



(a) F0 duration(sustain) envelopes



(b) Inharmonics of k=1-10 oscillators at first 1s.

Figure 44: F0 envelopes(a) and inharmonic contents present(b). Detuning slight deviated the F0 sustain envelope (some sharp peaks). Inharmonic contents are mainly present at higher frequencies.

But potentially useful in synthesising a greater range of notes to be played.

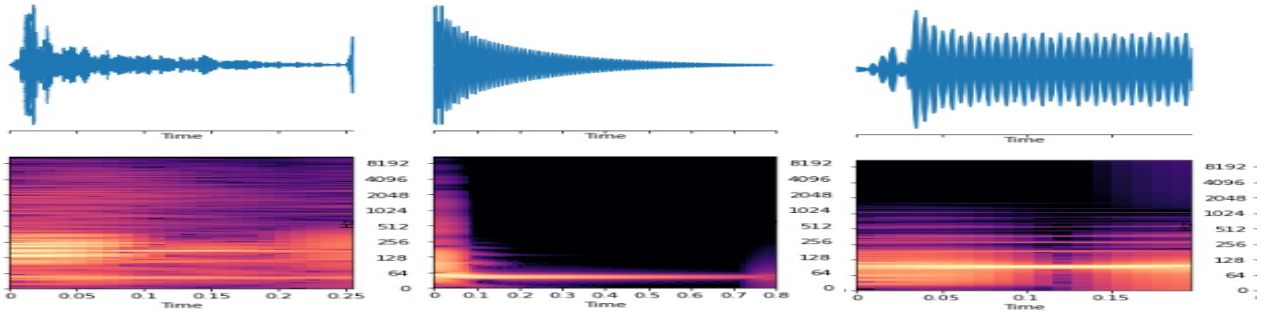
Results summary: Our model successfully synthesised piano in the absence of pedal effects. Cover most of notes present including any inharmonic sounds. Sound generated was also realistic by human perception.

4.3.3 Drum

Drum synthesis techniques

Drum is one of unpitched percussion instruments which is melodically important in both Western and non-Western music. It determines the key aspects of music such as tempo, rhythm, genre and mood. Therefore, we decided to include drum as one of instruments in our INSTRUMENT Module. Additive

synthesis alone will not be able to create drum-like sound because pitch itself is not a representative feature as shown in fuzzy spectrograms. Also, due to a limited number of oscillators and harmonics, we needed to carefully control our drum input audio under conditions.



(a) Snare: 100-500Hz (b) Kicks: below 150Hz (c) Toms: 50-200Hz

Figure 45: Drum Spectrogram: Although each has different frequencies range, the spectrograms do not show clear harmonic distributions, but rather fuzzy distributions.

DDSP Drum Model

Modifications on DDSP model have been made and tested to synthesise Snare,Kicks and Toms.

- **Transient sound in SMS[81]**: Transient sounds have high-amplitude, short-duration(less than 4s),and non-periodic components. They can be modelled as short impulses. [81] have shown that they comprise sinusoid components in the frequency domain. The proposed method uses additive synthesis in the discrete cosine transform(DCT) domain,then reconstruct the signal in the time domain(IDCT). $x[n] = d[n] + s[n]$, where $d[n]$ is a deterministic and $s[n]$ is a stochastic component. $d[n] = X + \sum_1^P a[n]$ sin$(\varphi_p[n])$ as harmonic models before but with presence of $d2$ as transient sound $d[n] = d1[n] +$ IDCT$(d2)[n]$

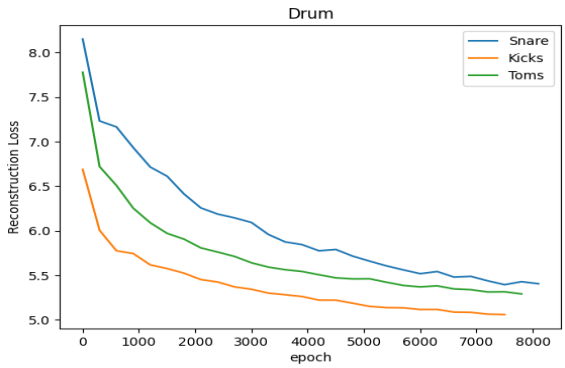
- **Filter: Impulse response** : Additive synthesis technique can also be applied to both harmonics and impulse response in frequency domain. Impulse response determines drum's amplitude, attack, decay, and duration. Former approaches on *Multi-scale Loss* penalise reconstructed sound output with ground truth at note-level. However, in drum synthesis, constructed impulse response should be compared with ground truth at frame-level amplitudes. Loss: $MSE_{DCT} = \frac{1}{n}\sum_i \sum_j (DCT(IR)(i,j) - DCT(Groundtruth)(i,j))^2$

Pre-processing and Data augmentation

- **Audio Preprocessing**: By experiment, it has been noted that 2s of transient drum sounds are not long enough to capture ADSR. The possible reasons are 1) uncertainty in onset detection by

CREPE[69](attack). 2) low accuracy due to the limited number of frames in embeddings(sustains& decays). Thus we have concatenated more than two transient sounds as a beat sequence to make sure the model captures the onset(attack) and the impulse response(decays) repeatedly.

- **Data augmentation**: *Drum Kit*[82] data is not long enough(total 2 minutes each) for training. This can cause overfitting[11] and not robust to unseen input. Therefore, we tried data augmentation[83] such as pitch shifting (by 10% of F0), delaying decays, amplitude reduction(by 10% of total) and MUSAN[84] noise (SNR = 5dB), while room reverberation and filtering high frequencies were not used to keep impulse response unchanged and timbre of drum respectively.



(a) Training loss

Evaluation	RMSE	FAD	DTW	IS	MOS
Snare	5.602	**5.7639**	0.4312	1.0	**4/5**
Kick	**5.060**	15.473	0.3600	1.0	3/5
Tom	5.427	14.93	**0.3202**	1.0	2/5

(b) Evaluation

Figure 46: Evaluation on Drum: Snares,Kicks and Toms. Pre-train data: *Drum kit Sound Samples*[82] contains 2s of Snare, Kicks and Toms .WAV 40 files individually, sampled at 44kHz.

Training& Evaluation: Drum

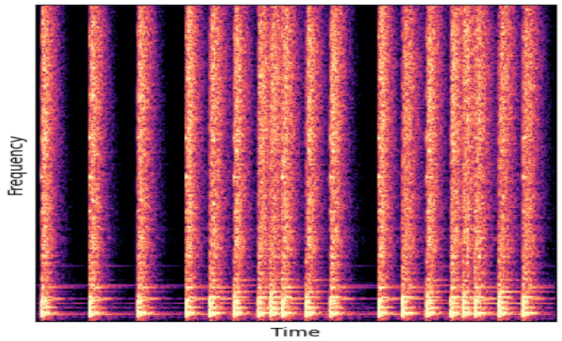
The aim of this training is to make sure the model learns the snares, kicks and toms by 1) impulse response 2) harmonics 3) timbre. At the end of the training, the model should be able to synthesise the drum beat sequence. Additive synthesis has its own limitations.

Synthesis results summary

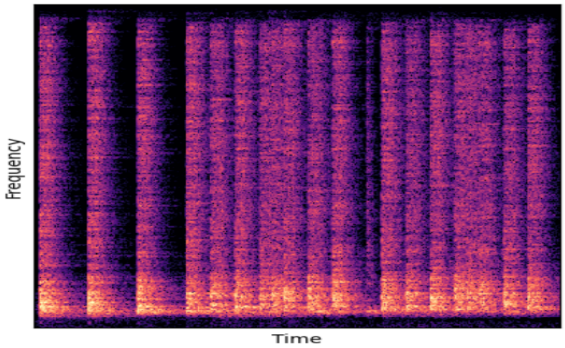
- **Spectrograms[Figure45]**: All 3 models correctly predict consecutive transients, but the difference in performance was due to corrupted noise components at low frequencies, which created much sparser harmonic distributions. For example, Toms showed high RMSE in Figure46(b) and not clear high- frequency harmonics in Figure45(f).

- **F0 and amplitude envelope[Figure48]**: Ideally, F0 envelopes should decay much quicker than amplitude envelopes, and the peak of F0 envelop should exactly be at the same as amplitude en-

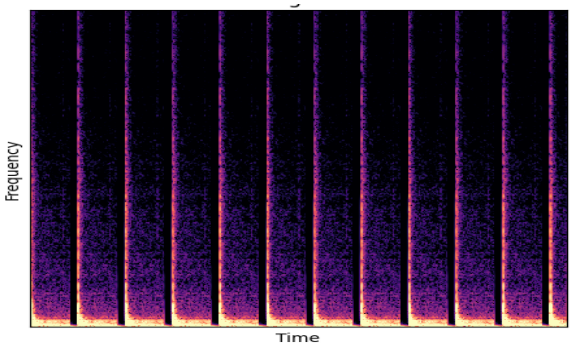
[11] overfitting refers to memorizing it instead of learning general patterns,leading to a lack of generalization



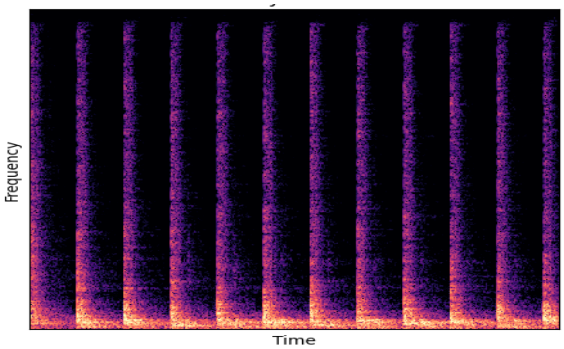
(a) Snare sequence beats



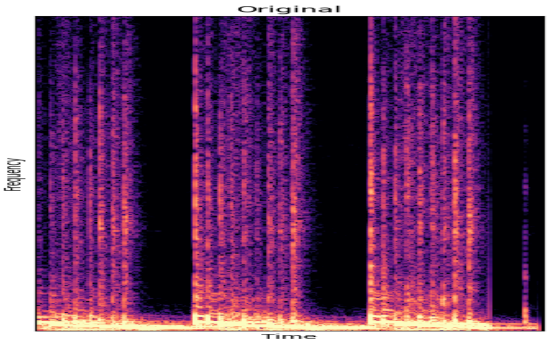
(b) Synthesised snare



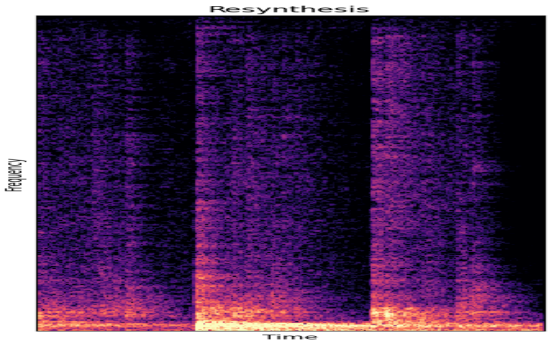
(c) 100Bpm Bass input



(d) Synthesised bass



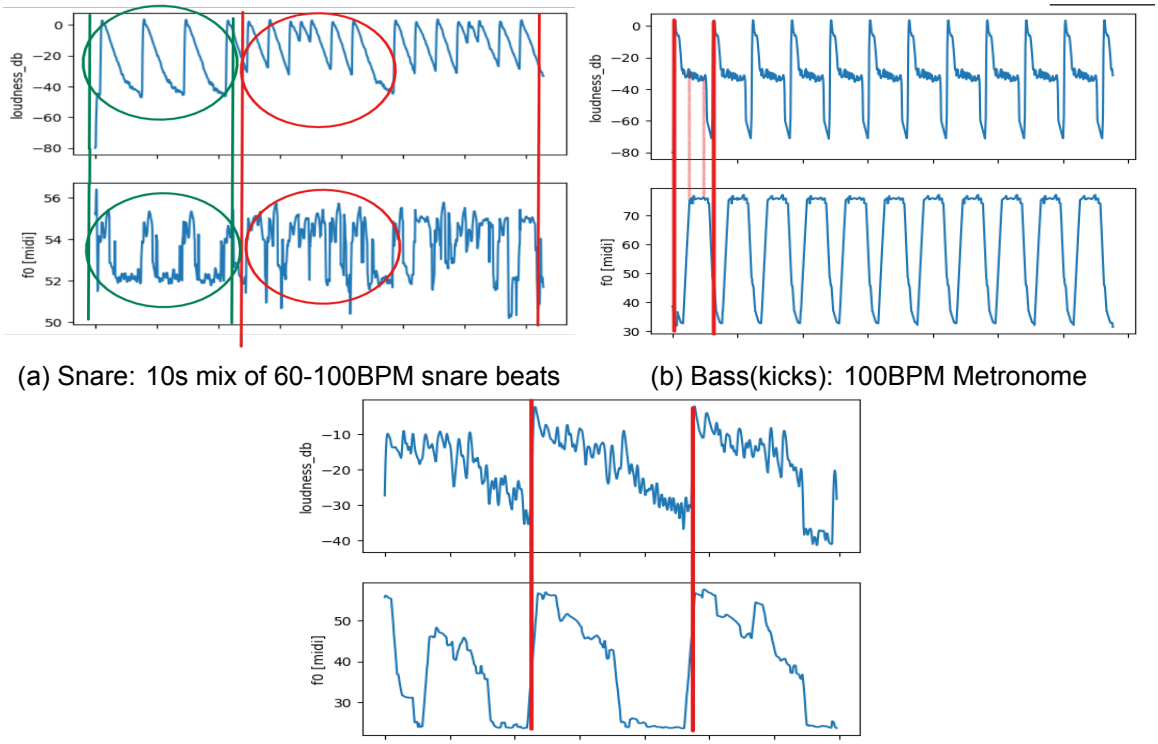
(e) Toms sequence input



(f) Synthesised toms

Figure 47: Snare, Bass(kicks), Toms synthesis results

velopes. Figure48(a) shows how F0 was detected at the instance of snares. The error was the onset note envelope shifted by a few ms in time. Also, in fast-tempo(100bpm), next sound envelope appeared before the previous sound completely decayed (which is equivalent to polyphonic sounds). This causes unreliable F0 over this period. Therefore, our model is only reliable for slow-tempo beat sequences (below 70bpm). Note that perfect match of F0 and amplitude envelope for toms as highlighted in both 48(b) and 46(b): DTW=0.3202. Alignment is well matched with F0 just detected at the onset. However, a pitch envelope is very slowly decaying this cause unpleasant sounds by perception as highlighted with high FAD score in 46(b): FAD=14.93. For Bass(Figure48(b)), the same problem of shifted F0 appeared, but it shows an accurate ADSR synthesis.



(a) Snare: 10s mix of 60-100BPM snare beats (b) Bass(kicks): 100BPM Metronome

(c) Toms: 3s of 4toms sequences

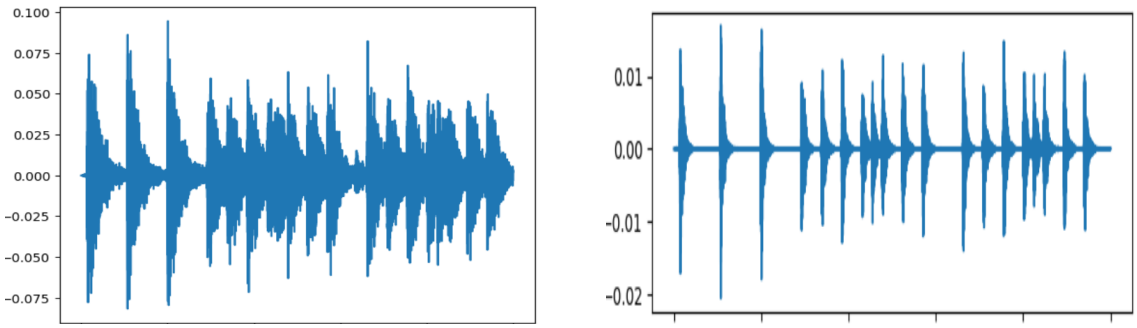
Figure 48: Synthesised loudness envelope and F0 envelope of snares, kicks and Toms. Note how the loudness ADSR matched to pitch ADSR in peak(onset), period, sustain and decay

- **Listening test**: All 3 models were able to generate percussive-like sounds. However, all 3 models could not accurately generate the timbre of Snare, Bass and Toms. Snare sounds very thin whereas input was a much deeper sound. Both Bass and Tom did not capture sharp attacks of transients as evaluated in the FAD score in Figure 46(b). Especially, toms showed slowly-decaying ADSR and pitch envelopes in Figure46(c). This would create much weaker transients compared to the original input sound.

- **Filtered impulse response[Figure49]**: In Figure49 impulse response is estimated by linear time-invariant finite impulse. Our assumption on LTI filter worked well to capture transients and generate envelopes. But the ambiguity comes with the room reverberation effect. The model was not able to distinguish between room reverberation and the reverberation of the drum itself. Therefore, many quiet amplitudes on drum reverberation were detected especially for the Bass drum.

Conclusion: Drum synthesis

Our DDSP model contains 3 basic pre-trained model: Snare, Bass and Tom. This model cannot synthesise polyphonic beats sequence in real time. Users can synthesise one envelope at a time. The bass



(a) Output amplitude (b) Filtered impulse response

Figure 49: Snare Impulse response estimation. From decoder output vector H_k, h_k = IDFT(H_k) in the time domain, with *Hann window* $W_k = 0.5(1-cos(2kN))$. The synthesised output is Y_k= $H_k X_k$, where X_k is additive harmonics 256 vectors.

was quite robust to high-tempo(>100bpm), whereas Snare and Tom have limitations on synthesising high-tempo. Therefore, this tempo(bpm) should be carefully controlled by users. Also, dereverberation before processing each drum transient will improve the sound quality. In terms of pitch, the model can synthesise pitch within or even outside of the learnt pitch range. This gives users the flexibility of synthesising the same type of drum but with different sizes or materials.

Generating drum beats: Using Hardware & Pram Module

Even though it cannot decompose drum beats due to the limitation of the monotonic DDSP model, we can combine different drum sounds to generate a complete drum beat sequence. In Section3.1.2, *Parameter Matching* module shows ability to model sound directly using modular synthesizer hardware. *Frankenstein* can produce a much more reliable beat sequence using both *Paramaeter matching, INSTRUMENT*, then using *timbre transfer* we can modify each envelope.

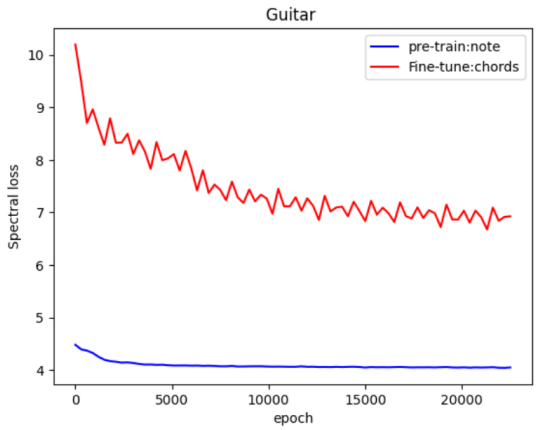
4.3.4 Acoustic Guitar

Introduction: Guitar

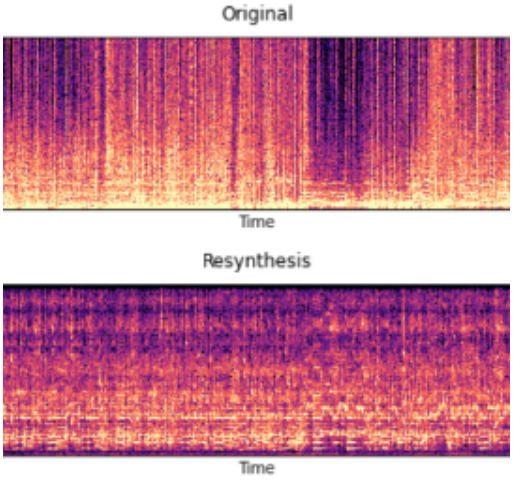
The guitar is a plucked string instrument. It is typically hard to be synthesised with additive synthesis(DDSP) due to inconsistencies. For example, different guitarists have their own way of plucking strings, the position of hands, different stiffness of guitar string(material properties)used, how much it is tuned, etc. Therefore, the guitar has its own ADSR depending on the performer's hand position and reverberation present through the base. Despite of difficulties, we trained a guitar DDSP model on note-level data, chord-level data, and then per note song and evaluated separately

Experiment: Guitar

First train note-level data, then fine-tuned on chord-level or real performance data. We need to see what chords are synthesised well and which chords are not well done.



(a) Training loss



(b) Synthesised guitar: Chords sequence

Figure 50: Batch size = 16, Adam optimizer, Learning rate =0.001, decay = 0.01 , Sampling rate = 16khz. Pre-trained: guitar note Fine-tuned:*GuitarSet*

Figure50 shows a significant increase in loss when the model synthesises chord-level audio. We have highlighted evaluation results on each chord. Figure50(b) shows an unsuccessful chord-level synthesis. RMSE=7.4, noisy, thin electric sound generated.

Evaluation: Guitar

Chords level synthesis was not perfect to generate all chords, especially minor chords. Chords with significantly high RMSE(above 6.5) loss sound rather = electronic not acoustic at all. But, all note level sounds were successfully generated with RMSE below 5.5.

Chords	**C**	**D**	**E**	**F**	**G**	**A**	**B**$flat$
RMSE	5.564	5.311	4.90	6.92	5.82	5.51	6.70
Chords	**C minor**	**D minor**	**E minor**			**A minor**	**B minor**
RMSE	4.80	5.77	5.18	-	-	6.333	7.21
Chords	**C7**	**D7**	**E7**	**Fmaj7**	**G7**	**A7**	**B7**
RMSE	7.86	6.433	5.0	6.53	6.801	4.701	7.801

Table 7: Chords reconstruction loss: RMSE is proportional to number of strings taht were plucked

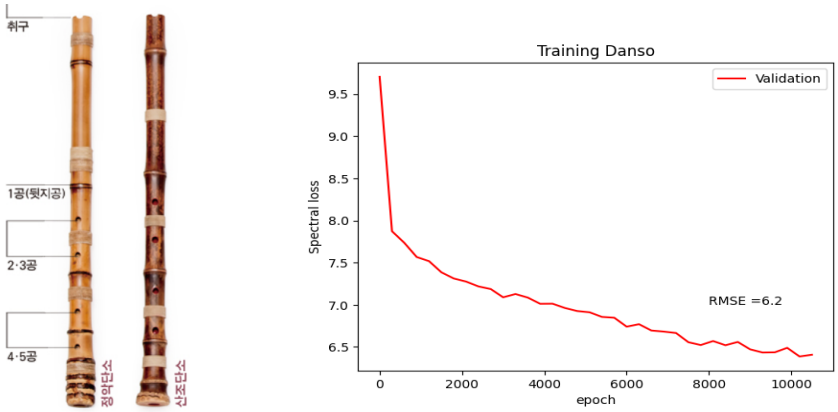
Results summary: Different from ideal scenarios, other factors such as the material properties of the guitar, the type of string, and the playing technique can influence the harmonic distributions present in a non-linear way. For example, different types of strings or pickups may emphasise certain harmonics over others, and playing techniques such as bending or vibrato can introduce additional non-integer harmonics

to the sound. We have noticed this effect amplified in *chords level* synthesis, where more than one string is plucked with fingers. The model failed to capture harmonics with unique ADSR by players and types. But, still be able to capture per string-based note-level sound.

4.3.5 Danso

Introduction: Danso

Danso is a Korean traditional woodwind type instrument (Figure51). It will be a very uncommon but unique instrument for most users. Most synthesis techniques were focused on Western musical instruments. Asian type music has become popular with a mix of conventional Western musical instruments. Here, we present Danso sound synthesis as an example of a unique, uncommon musical instrument for users' creativity.



(a) *Danso* (b) Trining *Danso* on pre-trained *Flute*

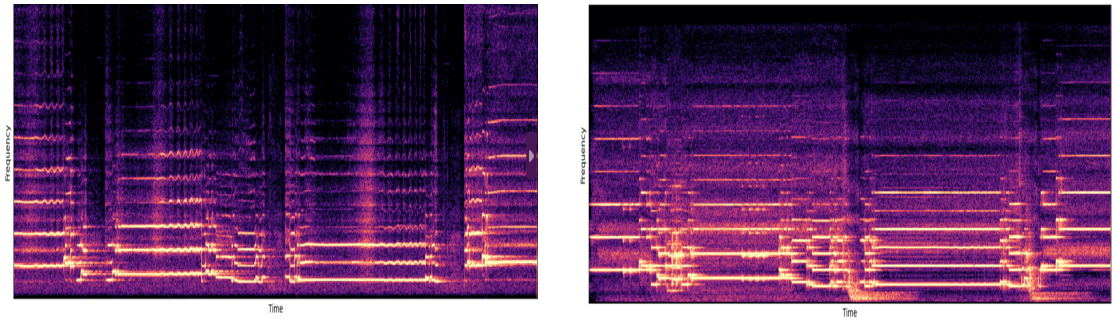
Figure 51: Traning Danso: **Pre-trained**: **DDSP pre-trained flute model**[67]. No available large, clean Danso audio data. **Fine-tuned**: **Gugak**[85]: total 5 miniutes Danso samples.

Training Danso

Motivated from the paper[86], the author used *transfer learning* on pre-trained Western drums to fine-tune on Indian tabla strokes of CNN-based classification. This achieved better generalisation with 3 times smaller dataset for fine-tuning. We used a similar approach: use pre-trained Flute to fine-tune Danso. This is because Flute has a frequency range of three octaves, from C4(262Hz) to C7(2096 Hz), and Danso has two octaves from G3(196Hz) to G5(784Hz). This covers the most pitch present in Danso.

Evaluation: Ambiguity in timbre

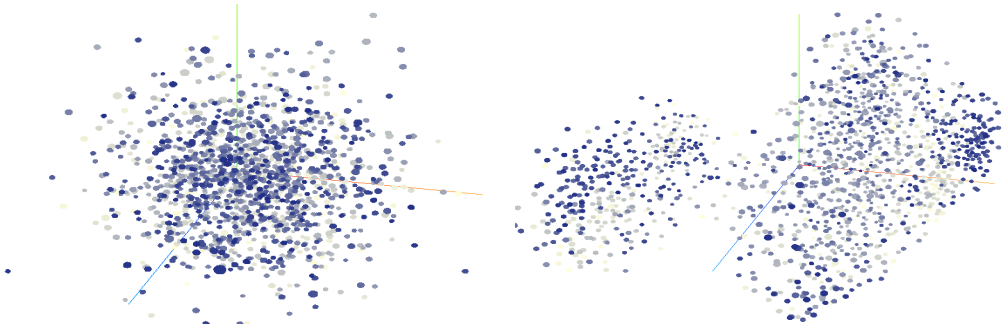
The sound quality is in the acceptable range with RMSE=6.2. However, the synthesised timbre was



(a) Danso Spectrogram (b) Flute Spectrogram

Figure 52: Spectrogram comparison between Danso and Flute: timbre is quite similar in the spectrogram(MFCC16) each other as a woodwind type instrument.

ambiguous, realising a more Flute-like sound rather than Danso or neither of them (Figure 52). This is a frequent problem when synthesising unseen Danso audio. One possible approach taken was to visually understand abstract timbre space using PCA[49] in previous Section3.3.6 and T-SNE[87][12] of timbre space representations. Figure53 shows the model failed in distinguishing Danso from Flute. This is the reason why it sometimes sounds like Danso and sometimes Flute depending on the input audio.



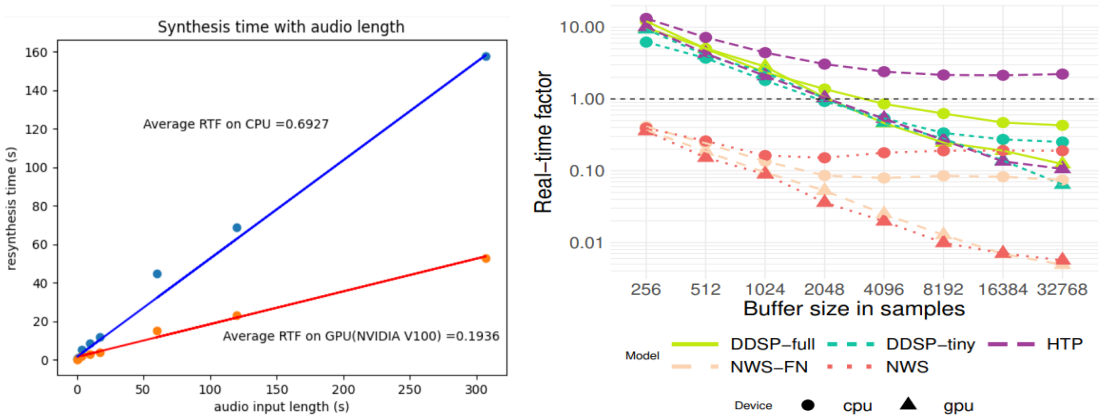
(a) PCA: Flute vs Danso (b) T-SNE: Flute vs Danso

Figure 53: PCA and T-NSE space(512 dimension embeddings scaled down to 3 dimensional feature vectors) for latent *z-embeddings*. Dark blue: Danso, Light blue: Flute. Failing at generating distinct timbre space of Danso. Timbre latent space is rather entangled, the latent variables of Danso and Flute are not separated. Thus, our DDSP decoder does not guarantee pure Danso-like sound, it lost the controllability of timbre in this case.

Results summary and Future work: Danso

Danso synthesis based on *transfer learning*[86] caused a major problem in generalising the timbre space of Danso and the timbre space is not guaranteed to represent the instrument well. Therefore, careful control and preprocessing of the input audio are recommended. For resolving this problem in future, we suggest some modifications on the model:

[12]Simmilar to PCA, it is a dimensionality reduction technique for visualizing high-dimensional data in a lower-dimensional space



(a) Total synthesis time of Our DDSP model with audio varying length

(b) *Hayes*[63] RTF vs buffer size in DDSP, and NEWT model.

Figure 54: Synthesis time on CPU and GPU, Real-time factor=0.69(CPU intel i7, 8 cores), 0.19(GPU, RTX A4000). Buffer size vs RTF shows we need to fix the buffer size for given input audio. The higher buffer size, the lower RTF can be achieved. But a buffer size greater than 1024 makes both GPU and CPU inefficient. Despite this, we select 4096 buffer size to strictly reduce RTF below 1.

1. Feature extraction: *z-encoder* uses MFCC-30 for extracting features. CQT or STFT may capture and distinguish the timbre of Danso and Flute better.

2. Public listening test: human perception could be the simplest and most powerful for modelling timbre latent space. *P. Esling*[88] have shown that regularizing VAE(variational autoencoder) timbre space with perceptual rating embeddings.

4.3.6 Practical considerations: *INSTRUMENT* module

INSTRUMENT DDSP module should meet the object specification (in Section4.2.3. Our INSTRUMENT DDSP module showed RTF=0.69(on average) on CPU (See Figure54) with most of the synthesis time less than 600ms. This is greater than 500ms we targeted, but since RTF=0.69 is strictly less than 1 shows a feasible synthesis time in practice. The memory requirement is the main problem since memory usage significantly grows with audio length. Alongside this, we noticed that buffer size should be controlled accordingly to prevent an explosion of memory[63]. On the personal device, the buffer size of 4096 should be fixed such that the maximum sampling rate=16kHZ to prevent overloading. With the use of GPU(RTX A4000), RTF=0.19 was achieved(Figure54). As mentioned in Section4.2.3, a maximum 5 minutes long audio synthesis (on the server) with 44KHZ sampling rate on GPU(RTX A4000) was feasible in practice.

4.3.7 Conclusion INSTRUMENT module

In *instrument* module, total of 6 models were provided and verfied their performance. Violin was robust to real-world data. Piano, Drum can synthesise for clean data under control input. Although Danso showed ambiguity in timbre, it could generate sound clearly. Acoustic guitar was only able to synthesise note-level audio, very susceptible to chords with multi-strings. Overall, the quality of sound was found to be lower than the existing market synthesisers. However, we showed controllability and a light-efficient model, which integrates different types of instruments only with additive and subtractive synthesis. This module can help beginners or non-instrument players generate and control musical instrument sounds. (All synthesised audios and model weights available [78])

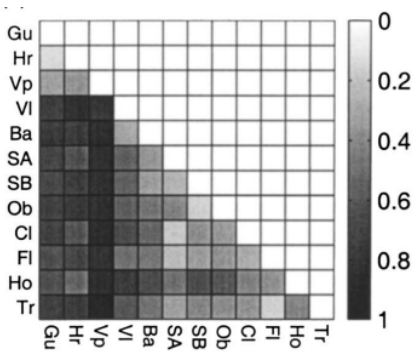
4.4 Timbre transfer

4.4.1 Introduction

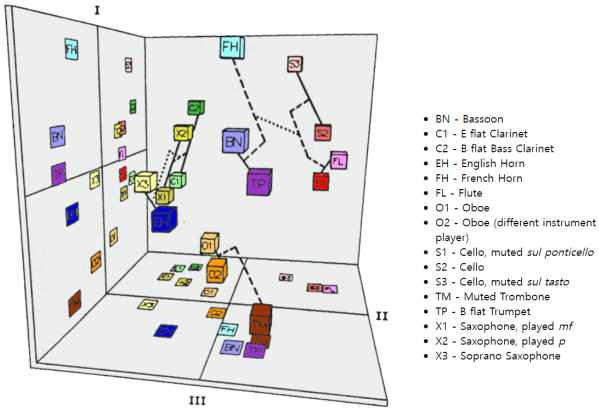
In our INSTRUMENT Module VST, we included a recommendation of instruments for timbre transfer in the previous Section4.1.2Figure 30. We showed timbre is abstract and hard to control in Section4.3.5. Thus timbre space between instruments should be disentangled, and provide users high-quality timbre transfered sound. While, state-of-art metric-learning-based recommendation[89] is the best, considering the size of computations, we will use conventional timbre space analysis of timbre descriptors[90] based on human perception.

4.4.2 Timbre space of INSTRUMENT module

When using timbre transfer, validations need to be made on (1) sound quality (2) preservation of F0 and loudness (3) preservation of own timbre as discussed in Section4.3.5. We are going to use 3 timbre representations: 1. short note level timbre similarity matrix Figure55(a), 2. 3-dimensional MDS of instruments Figure55(b) 3. long term dependent timbre matrix Figure55(c). Continuous feedback from the users on timbre transfer ratings will be used to modify the 3 timbre representations. Depending on the audio input length, we will provide different emphasises on 3 different timbre representations so that 3 nearest-neighbour instruments are given for *recommended* instruments for timbre transfer.



(a) Dissimilarity matrix on note *Bb4 Marozeau*[cite]. Darker means greater dissimilarity. First four, Gu(*guitar*), Hr(*harp*), Vp(*violin pizzicato*), Vl(*bowed violin*) showed more dissimiarity.



(b) 3D MDS timbre space[cite Grey], Axis i: Spectral energy distribution, Axis ii: synchronicity(onset-offset) of ADSR envelope Axis iii: high-frequency amplitudes and inharmonicities

Instruments	Violin	Flute	Piano	Guitar	Danso	Kick(Drum)	Snare(Drum)	Tom(Drum)
Violin	-	5	4	2	4			
Flute	5	-	3	2	4			
Piano	4	3	-	2	3			
Guitar	2	2	2	-	3			
Danso	4	4	3	3	-			
Kick(Drum)						-	3	2
Snare(Drum)						3	-	3
Tom(Drum						2	3	-

(c) Similarity matrix between instruments of *INSTRUMENT* module, based on MOS(scale out of 5), we could not validate timbre between*drum*(transients) and other deterministic instruments.

Figure 55: 3 timbre space representations: Simplest timbre representation is Figure55(c) based on our own evaluations. For note level, real-time timbre transfer Figure55(a) can rank the similarity in timbre. Figure55(b) is a more detailed measure for timbre transfer of its own ADSR envelope at frame level.

4.4.3 Conclusion: Timbre transfer

We provided the recommendation of instruments for *timbre transfer* based on the similarity between timbres. This can help users to select suitable options for timbre transfer avoiding bad quality sound output in software interface Section4.1.2. Continuous user feedback can enhance this timbre space representation for instrument recommendations.

5 Module 3: Reverberation Module - Yuanjun Wan

5.1 Overview

5.1.1 Section Overview

Reverberation is an essential component of sound that plays a critical role in creating a sense of space and adds realism to music production. The ability to stimulate the sound of a specific environment can significantly enhance the overall quality of synthesised music. Due to practical constraints, applying all reverberation effects via natural surroundings in music production is impossible. This constraint highlights the significance of developing an artificial reverberation module in our product. Multiple approaches to achieving artificial reverberation have been developed in recent years. In this section, a feedback delay network is applied to design the reverberation module after analysing the advantages and drawbacks of current approaches.

Parameter matching [13] of the reverberation module is often required when users want to apply a specific environmental effect to dry sound. However, manually tuning the parameters to achieve a target reverberation is time-consuming and requires sufficient expert knowledge, which is challenging for most users who are not professional audio engineers. As a result, our product incorporates an automatic parameter matching function based on a genetic algorithm to make the reverberation module more beginner-friendly. The algorithm optimises the parameters of the feedback delay network to match a target reverb iteratively. Therefore, the reverberation module designed in this section should be able to be both tuned manually and match the parameters automatically.

5.1.2 Design Overview

This section introduces the general flow of design before diving into more detailed discussions. As mentioned in section 5.1.1, the module will mainly consist of two parts: a reverberation model and a reverberation parameter matching algorithm. The reverberation model is straightforward: input signal from an audio file or MIDI keyboard passes through a four-line feedback delay network to generate a reverberated output. For the parameter matching algorithm, target impulse response wav. file is fed into the genetic algorithm to find the optimal parameters that imitate the reverberant environment of the Impulse Response. The Optimal parameters are then set in the feedback delay network to apply the reverberation

[13] This parameter matching function is for the reverberation module, which is different from parameter matching for analogue synthesiser in section 3.

effect on the input audio. The flow diagrams of these two parts and their interaction are shown in Figure 56.

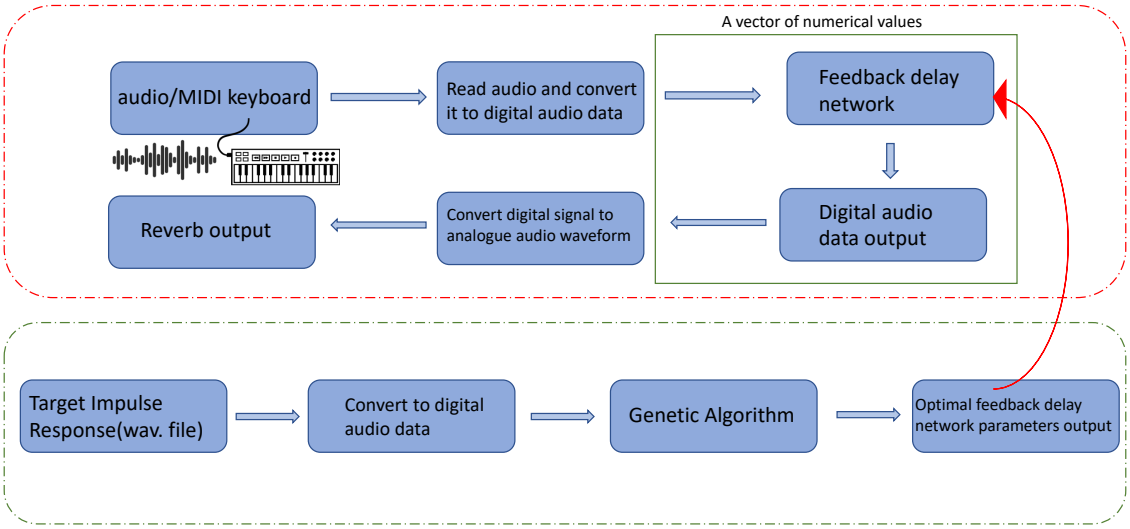


Figure 56: General flow of reverberation model and reverb parameter finding algorithm

5.2 Introduction to Reverberation

When sound waves emanate from a source such as an instrument in a room, they travel in all directions until they encounter surfaces, at which point they reflect and change directions. The process of reflection continues until sound waves eventually dissipate or are absorbed by surrounding materials in the room. As a result, we will hear sustained and echoey sounds due to the reflection of sound waves, and this effect is known as reverberation. Reverberation is influenced by various factors, including the size and shape of the room, the positioning of things inside the room, and the materials of their surfaces. It can thus be deduced that different reverberation effects can arise from different spaces (e.g. hall effect, room effect), which provides the original sound with various senses of depth, richness and complexity. Reverberation is widely used in music production. One example would be the famous song 'Bohemian Rhapsody', where different amounts of reverberation are added to piano and guitar to contribute to the grandeur and drama of the song by creating a sense of space around them.

5.3 Room Impulse Response(RIR) Structure

5.3.1 Introduction to RIR in Reverberaiton

A reverberant environment can be modelled as a Linear and Time-Invariant System (LTI). It can be described by its Room Impulse Response (RIR)[91]. A Room Impulse Response usually consists of direct sound, early reflection, and late reverberation. Figure 57 shows an Impulse Response plotted against time in dB.

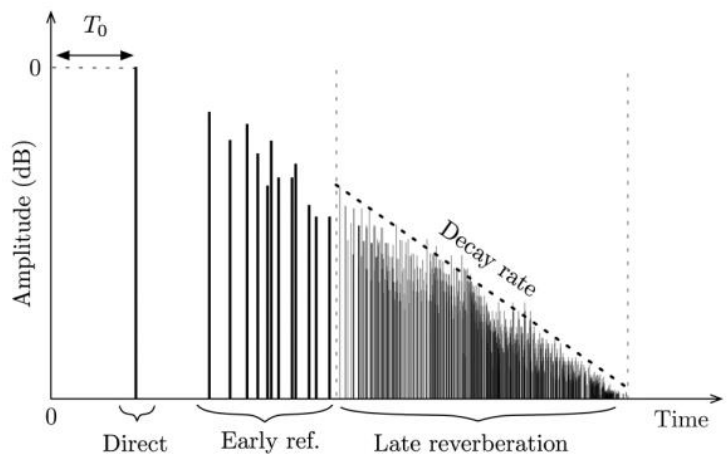


Figure 57: Example of a generic Room Impulse Response[92]

The first event of a Room Impulse Response is called the **direct sound**, which refers to the sound that travels straight from the sound sources to the listener without being affected by any reflections or echoes after a delay of T_0. It allows the listener to gain essential information about the original sound source, such as melody, lyrics and timbre of instruments.

Early Reflection occurs immediately after the direct sound. They are the first reflections of sound responsible for creating the sense of space, direction and distance in the sound. These elements are essential for a natural-sounding environment.

Late Reverberation is the last stage of the reverberation effect. At this stage, the sound waves that reach the listener have often been reflected against different objects many times. The late reverberation creates a period of dense echoes, generally **the longest audible portion of reverberation**[93]. Due to a high number of collisions, sound waves are attenuated by absorption until a point where the sound is no longer audible.

5.3.2 Impulse Response(IR) Collection

Impulse Responses are essential data in many reverberation algorithms since they contain information about the reverberant environment. In our design, Impulse responses are the reverb parameter matching algorithm's dataset, as we are finding parameters that match a particular impulse response instead of directly matching reverberated audio.

Both recordings and artificial responses can collect impulse responses. An impulse response is a recording of the reverberation of an acoustic space when an ideal impulse is played[93]. To record the impulse response, one needs to use a microphone to record a burst of sound (e.g. pop a balloon) until the room returns to silence. The wave file created can be used as the reverberation effect of that specific room/space. Impulse Response can also be generated by feeding an impulse to a Reverberation VST. The first method is usually preferred when audio engineers want to collect more natural reverberant effects, while the second method is preferred for more artificial effects.

5.4 Artificial Reverberation Model

5.4.1 Performance Criteria for Model Selection

Reverberation algorithm generally falls into one of three categories: convolutional, delay network and computational acoustics[92]. Ideally, all of them can produce proper reverberation effects from the continuous development of algorithms over the years. However, when a synthesiser module is developed, we should bring practical considerations in choosing the appropriate algorithm. We have identified **four main performance criteria** for this reverberation module.

Sound Quality – We need an algorithm to produce high-quality, realistic reverberation sounds. The reverberation should sound natural, smooth and balanced without harsh or artificial artifacts when we try to model natural reverberation.

Latency – The algorithm should have low latency, meaning there is only a little delay between the input and output signals. Low latency is essential for synthesisers as responsiveness is vital for music producers to improve efficiency, and this quality is also crucial for artists focusing on live performances.

Flexibility – The algorithm should be flexible enough to produce different types of reverberation sounds, which means it should provide the users with sufficient flexibility in control over the parameters of the reverberation model.

Efficiency – The algorithm should be efficient enough to run smoothly on a local device. In addition, it does not need significant computational resources, such as processing power, memory or time to run on the local device, which can result in slow performance or even system crashes.

5.4.2 Reverberation Algorithms and their performance

In the following sections, reverberation algorithms in the three main categories mentioned in 5.4.1 are introduced and examined on the four main performance criteria discussed in 5.4.1. Based on the examination of these criteria, the most suitable category is selected for the reverberation module.

Algorithm1: Convolution Reverb

Since the reverberant environment can be described by its Room Impulse Response, the convolution between Room Impulse Response and input audio applies the environment reverberation effect to the input[91]. The convolution of continuous time-varying input signal x(t) with room impulse h(t) gives an output y(t) defined as

$$y(t) = x(t) \cdot h(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t-\tau)\, d\tau$$

In digital signal processing, convolutional reverb is discretized by sampling the continuous signal at regular intervals. As a result, the convolution reverb is a discrete version of the equation introduced above.

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Every output is the combination of current and past x[n] values weighted by the sliding impulse response.

Performance Criteria Specific Examination of Convolution Reverb

Sound Quality – Convolution reverb is powerful in creating realistic acoustic spaces in audio production [94] since it convolved impulse responses of a real-world environment with audio. Due to its high level of realism, it is widely used in film and television sound design and music production[95].

Latency – Convolution reverb generally takes higher latency than other reverbs as it involves complex mathematical operations[96]. One of the most popular research areas in convolution reverb is lowering latency.

Flexibility – Convolution methods lack flexibility in controlling different parameters to alter the reverberation effect[92], thus less suitable for applications that require more parametric variations.

Efficiency – Convolution methods are more computationally expensive if users want to produce a sound convolved with long RIRs. In addition, the memory cost of storing each IR is another potential downside of convolution reverb.

Algorithm 2: Computational Acoustics

Computational acoustics methods in reverberation refer to the use of mathematical and computational techniques to stimulate the behaviour of sound waves in a reverberant environment, such as a room, a hall or an auditorium. The stimulation of acoustic energy involves several steps:

Geometric Modelling of the environment: A virtual reverberant environment model is first modelled using computer-aided design software. In the virtual model, the shape, size, and material properties of surfaces need to be specified.

Boundary Conditions: We also need to specify how the sound waves interact with the boundaries of the environment in terms of how they reflect, absorb, or transmit through the surfaces. The interaction of sound waves and boundaries usually depends on the material properties of the boundaries, such as absorption and scattering coefficients.

Wave Propagation: The propagation of sound waves is stimulated with an initial input signal. One method to solve the wave equation would be the Finite-difference time-domain (FDTD) algorithm, in which the wave field is computed at time steps and displayed at regular intervals[97]. The stimulation takes the geometry of the environment into account.

Reverberation: As the sound wave stimulation progresses, the reflections at the modelled environment's boundaries are calculated.

Performance Criteria Specific Examination of Computational Acoustics

Sound Quality – The sound quality of reverberation generated by computational acoustic depends on the accurate representation of the geometry of the environment and the realism of initial stimulation. However, we do not always have this information for every reverb effect, which makes the sound quality performance inconsistent.

Latency – Wave propagation method such as FDTD is computationally expensive, which results in higher

latency[98].

Flexibility – This method has good flexibility as we can control many geometric parameters and boundary conditions to influence wave propagation.

Efficiency – The complexity of the geometry of the environment, coupled with the wave propagation algorithm, can result in high computational costs.

Algorithm 3: Delay Networks

The basic principle of creating reverb effects with delay is to combine the original signal with delayed versions of itself to represent virtual reflections or echoes in the environment. The combination of input signals and delayed versions can be controlled by a set of feedback delay network. The delay network can further contain additional processing, such as filtering, in addition to delay lines to stimulate further the absorption and scattering of sound waves in the environment.

Performance Criteria Specific Examination of Delay Networks

Sound Quality – The reverberation effect produced by delay networks is based on mathematical algorithms that stimulate the behaviour of sound waves in a virtual environment. It may not consistently achieve the same level of accuracy in reproducing the nuances of real-world acoustics compared to convolution reverb that uses recorded impulse response.

Latency – Delay Networks have relatively low latencies compared to other reverberation algorithms, such as convolution reverb. They do not require real-time convolution with large impulse response files. Relatively low latency gives Delay networks a big advantage in live sound processing.

Flexibility – Delay networks are known for their flexibility in control over the produced sound. Algorithmic reverb plugins based on delay networks are easily modulated. Parameters available for modulation can vary from a few based on simple designs[99] to a lot based on more complex algorithms[100].

Efficiency – Reverberation based on delay networks is computationally efficient[101, 102]. Delay works usually use relatively simple mathematical operations, such as delay lines and feedback loops, which are less computationally expensive than convolution or geometrical modelling and wave propagation operation.

5.4.3 Performance Metrics

A performance metrics table is listed below based on the discussion of performance relevant to each criterion in 5.4.2. In the table, a score is assigned to the performance on every criterion, with the best performance being 5 and the worst performance being 1. Each criterion is assigned a different weighting ranging from 0 to 1 based on what we expect to achieve in designing the module.

The reverberation model is expected to give users sufficient control over the reverb parameters so that users can freely create different effects. Hence, we will assign 0.3 to the weighting of flexibility. In addition, the model serves commercial use instead of academic research. It is essential to ensure the responsiveness of the model. The product should also not take up too much memory of devices to leave space for other software. Hence, we will assign 0.25 to both efficiency and latency. Sound quality is assigned a weighting of 0.2. However, it does not mean that sound quality is not as crucial for the product. With all the practical constraints, we will optimise the sound quality via the algorithm. Constant development will be implemented even after the module is released to users.

The total score of the performance of each category is calculated by the sum of the individual score of each category multiplied by its weighting, which is shown in table 8.

Reverberation Model Category	Sound Quality	Latency	Flexibility	Efficiency	Total Score
Convolution Reverb	5	3	3	3	3.4
Computational Acoustics	4	3	4	2	3.25
Delay Networks	3	4	5	4	4.1

Table 8: Performance Metrics of Reverberation Models From three categories

The table shows that delay networks receive the highest scores based on practical considerations in choosing an algorithm for the reverb module. Hence, **our module will create a reverberation effect based on delay networks**, which will be explained further in the following sections.

5.5 Feedback Delay Network

A feedback delay network is used among delay network algorithms for creating reverberation effects. The recursive nature of feedback loops allows the creation of multiple reflections and interactions of sound waves, which is able to imitate the complex behaviour of sound in a reverberant environment. We present the use of a four-line feedback delay network as the reverberation algorithm of the model. Parameters of the feedback delay network, which are the parameters we can tune to create different reverberation

effects, are also introduced in terms of their physical meanings.

5.5.1 Comb Filter

The building block of the feedback delay network is the feedback comb filter shown in Figure 58. A comb filter is a filter whose output is the combination of the input signal and a delayed version of the signal itself, resulting in constructive and destructive interference. It consists of a delay line that stores a copy of the audio signal delayed by a certain amount of time and a feedback gain that determines the amount of the delayed signal fed back into the input of the delay line.

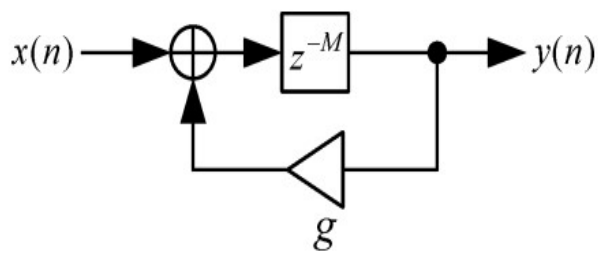


Figure 58: Block Diagram of Feedback Comb Filter[103]

The transfer function of a comb filter is

$$H(z) = \frac{z^{-M}}{1-gz^{-M}}$$

5.5.2 Feedback Delay Network

A feedback Delay Network is the combination of a vector of comb filters. The development of this approach can be traced back to Gerzon's work[104, 105], and it was further advanced by Stautner and Puckette[106] and later by Jot and Chaigne[107]. The structure of the feedback delay network used in our reverb module is shown in figure 59, which is a four-line network based on Jot and Chaigne's design.

Components of the feedback delay network

1. Multiplier(b1,b2,b3,b4,c1,c2,c3,c4,d): It allows the user to control the coefficient that multiplies a given signal. The multipliers can be used to imitate the attenuating sound in an environment after rounds of reflection. Multiplier set b and c are located on delay lines, while multiplier d connects the input directly to the output. This is because while there is attenuation in delayed sound, there is also attenuation in sound reaching people directly due to the absopriton of air.

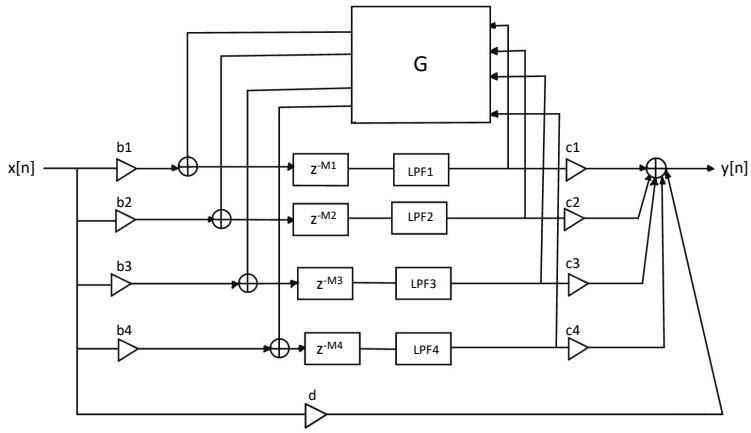


Figure 59: Structure of Feedback Delay Network in reverberation module

2. Delay line(z component): The delay line creates delayed versions of signals, which is an essential component in the formation of reverberation.

3. Low pass filter(LPF): The application of low pass filters in the model is to imitate the natural characteristics of sound propagation. In certain environments, such as rooms or halls, higher frequencies are absorbed more quickly than lower frequencies.

4. Feedback gain matrix(G): The feedback gain has evolved from a single gain value to a 4x4 vector in the model. This matrix determines how much of the output of each delay line is fed back into the input of other delay lines. It allows for intricate and evolving feedback patterns to be created, resulting in a rich and immersive audio effect. The values inside the matrix are notated as G_{xy} where x is the source comb filter, and y is the destination filter. The role of the feedback gain matrix is to control the damping characteristics of the stimulated reverberation and ensure the stability of the system.

Signal path in feedback delay network

The signal path of the reverberation model is described below and also illustrated in figure 60.

1. The signal input is fed into the Feedback Delay Network as the initial input. The input signal is distributed to the four delay lines of the delay network or the multiplier d that leads to the output.

2. In each delay line, the signal first goes through a multiplier.

3. After the multiplier, the signal passes through the delay line.

4. The output from the delay line is filtered by a lowpass filter and can join either one of the signal paths introduced below.

   a. The signal goes through another multiplier and arrives at the output of the network.

   b. The signal can also be fed back to the system via the feedback gain matrix G.

5. After the signal goes through the feedback gain matrix, it will be added to one of the delay lines and repeat the path from step 3.

6. The output is the summation of all signals arriving at the output y[n] after going through different signal paths in the feedback delay network. It can create complex and evolving reverberation and spatialization effects in audio signals.

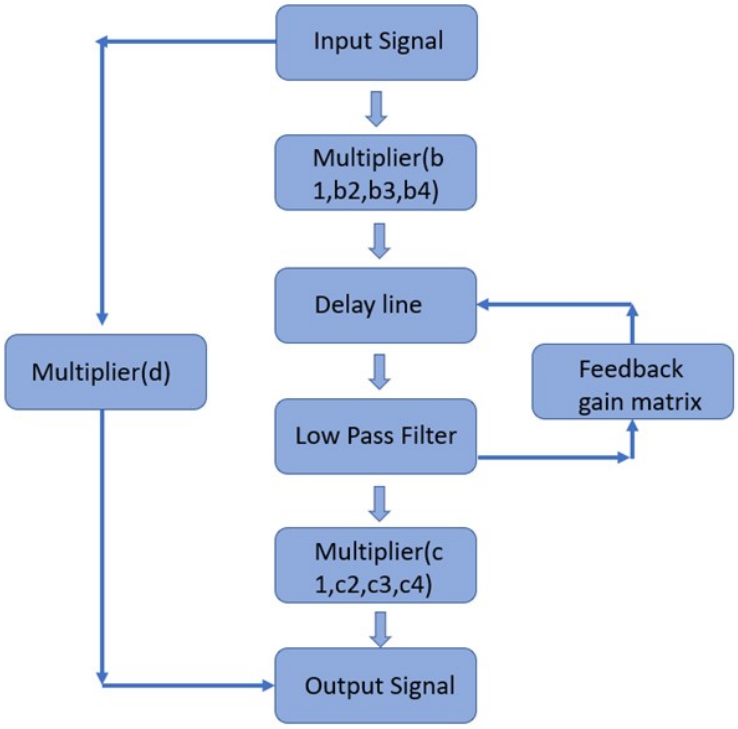


Figure 60: Signal Path flow diagram of Feedback Delay Network

5.6 Parameter Matching of Reverberation Model

As mentioned in 5.1.1, one of the most significant advantages of this reverberation module compared to other products in the current market is that it provides an additional parameter matching[14]function which better serves our target customers-beginners in using a synthesiser. Parameter matching means that

[14]This parameter matching function is for the reverberation module, which is different from parameter matching for analogue synthesiser in section 3.

when we want to apply the reverberation effect of a particular environment to dry sound, we can find the parameters of our model by matching them to a target impulse response(5.3.2). The reverberation effect is automatically achieved after the dry sound is passed through a feedback delay network with calculated parameters as shown in figure 56. This procedure effectively utilises the efficiency of the feedback delay network as compared to convolving the dry sound with target impulse response laboriously to create a specific reverberation effect.

5.6.1 List of parameters

Based on the introduction of feedback delay network components in section 5.5.2, a list of 6 parameters is presented below. In addition, since the feedback delay network applied to our module consists of multiple delay lines(figure 5.5), components of the same function at the same position of each delay line will be compiled into matrices.

1. d: multiplier
2. **b** $= \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix}$: multiplier
3. **c** $= \begin{bmatrix} c_1 & c_2 & c_3 & c_4 \end{bmatrix}$: multiplier
4. **G** $\in \mathbb{R}^{4\times4}$: feedback gain matrix
5. **f** $= \begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix}$: cut-off frequency of low pass filter
6. **M** $= \begin{bmatrix} M_1 & M_2 & M_3 & M_4 \end{bmatrix}$: length of delay line

5.6.2 Genetic Algorithm

The parameter matching algorithm selected in this product is the Genetic Algorithm. Genetic Algorithm is a class of optimisation and search techniques inspired by natural selection and genetics- the fittest individuals in a population are more likely to survive and reproduce. It has remarkable ability to solve complex, non-linear and multidimensional problems across various domains. This algorithm optimises parameters in the feedback delay network by initializing a population of potential solutions that evolve through a series process iteratively. Over continuous generations, the iterative process drives the population towards improved solutions and ultimately converges to a near-optimal set of parameters. Genetic Algorithm is chosen in this case because of its robustness: it can work with a variety of data types and it

works well even if the problem space is not well understood.

5.6.3 Stages of Genetic Algorithm

This subsection introduces stages of the Genetic Algorithm for parameter matching of the feedback delay network. The Algorithm introduced is developed based on Chemistruck [108] and Cerioli[109]. A flow diagram of the algorithm is presented first(figure 61), with further explanation of each stage.

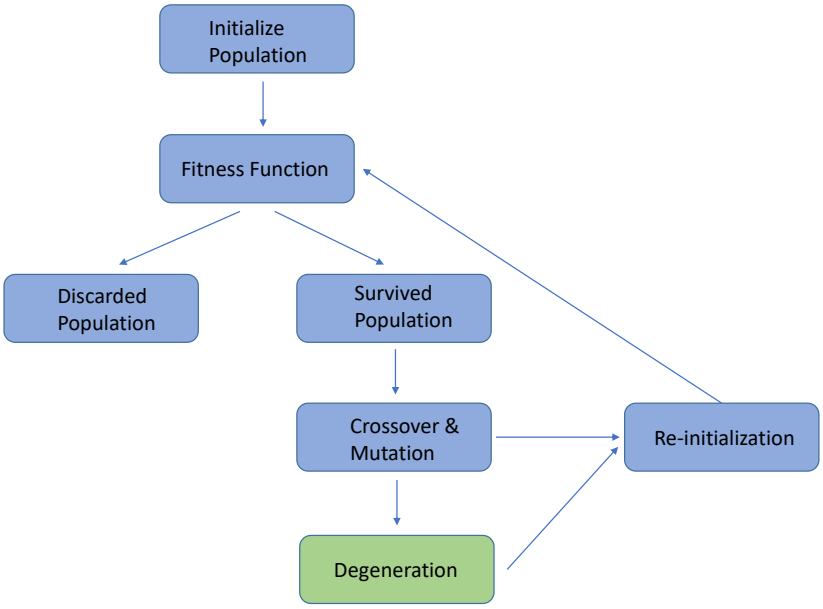


Figure 61: Overview of stages of Genetic Algorithm

Initialisation

Population initialisation is the first step in the genetic algorithm that involves creating a set of potential solutions to the problem at hand. The initial population is generated randomly to ensure diversity in the solution space. However, we can set a range for the population according to existing knowledge to help the algorithm find a more suitable answer. The range is set as follows.

1. The values of all multipliers, including **b**, **c** and d, are initialised between 0 and 1 because of the attenuation effect in reverberation.

2. In digital signal processing, the number of samples is typically used to express delay length instead of time. The upper limit of delay length should be high enough to accommodate long reverberation time, as our model should have the capacity to present parameters of large physical space.

One second is usually perceived as a long reverberation time[110]. Assuming the sampling rate is 44.1kHz, the delay length's upper limit is 44100 samples.

3. The feedback gain matrix is initialised to have a 25% chance of being a Hadamard matrix, 50% chance of being a Stautner-Puckette matrix, 25% chance of being a Circulant matrix, and 25% chance of being a Householder Matrix. We selected these choices as they are all popular feedback matrices with their respective advantages[111]. They are also stable and can be easily adjusted by a single multiplication factor, g, which is an advantage for the ease of control.

$$\text{Hadamard matrix} = g\begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \qquad \text{Stautner-Puckette matrix} = g\begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix}$$

$$\text{Circulant matrix} = g\begin{pmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{pmatrix} \qquad \text{Householder matrix} = g\begin{pmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

4. The cut-off frequency of the low pass filter is initialised between 0Hz and 1000Hz.

Fitness function

The Fitness function measures the performance of each solution in the population. In this situation, it compares the artificial impulse response produced by the candidate population to the target impulse response. The fitness function assigns a numerical fitness value to each candidate solution to indicate how well it matches the target impulse response. A lower fitness value means a better approximation to the target impulse response.

There are multiple ways to define the fitness function. In our parameter matching module, we will employ the method proposed by Chemisstruck[108] and Cerioli[109]. This method is based on an approximation of target impulse response and artificial impulse response to avoid one of the most significant problems in directly calculating the difference between signal samples – the samples in impulse response do not

adhere to a certain pattern, so information about the general pattern is lost. The approximation of impulse response is achieved by dividing impulse response samples into frames and performing linear regression on frames as follows.

$$y_{[sL,s(L+1)]} = nx_{[sL,s(L+1)]} + m$$

This equation performs linear regression on current frame s of L samples. If we combine the linear regression of all frames, we will obtain a line of approximation in the shape of an impulse response.

Criteria for surviving

In each generation, the fitness value of each candidate solution is computed. The values are then arranged so that 30% of the candidate solutions with the lowest fitness values survive while the other solutions are discarded.

Crossover function

The crossover function plays a vital role in merging the genetic information of surviving solutions to maintain the original count of candidate solutions. This process aims to explore potentially superior solutions by generating new ones. In this context, the surviving solutions from the previous stage are referred to as parent solutions, while the newly created ones are called child solutions. Child solutions **b**, **c**, d, and **f**, along with **M**, are obtained by computing the average of any two parent solutions. To preserve the stability of the feedback gain matrix, the child solution for matrix G is derived from the dot product of any two parent matrices.

Mutation function

The mutation function introduces minor, random variations in the candidate solutions of a population, which plays a crucial row in maintaining diversity within the population and preventing convergence to a suboptimal solution. In our case, all the parameters are mutated except the feedback gain matrix, as mutation may cause the feedback gain matrix to be unstable. The probability of mutation is arbitrary, but we must maintain a balance between searching for new populations and refining existing solutions. The mutation probability is set to 5% for all parameters.

Degeneration

Degeneration refers to the situation that most of the population converges to a suboptimal value, causing stagnation in the process. Child solutions produced from crossover function are highly similar to parent

solutions, making little improvement to reducing fitness value. In this case, we need to re-initialise the candidate solutions to jump out of the stagnation process. Degeneration may or may not happen in the iteration process. The degeneration condition is also set arbitrarily in the algorithm. In our model, degeneration condition is considered to happen when 95% of the population have fitness values within the range of ± 0.1 of the average fitness value.

5.7 Implementation and Evaluation

In this section, we implemented prototypes of both the parameter matching algorithm and the reverberation model in the MATLAB environment, and the output will be assessed. The implementation follows the steps outlined in the overview flow diagram(figure 56): matching parameters of a target impulse response and feeding the calculated parameters into the reverberation model to generate the final output.

5.7.1 Data for Implementation

This implementation requires a dry sound and room impulse responses of a target environment. The dry sound is an E4 guitar note generated by MATLAB by simulating a plucked string via importing the string's tension, cross-section, density, and length of the string[112] into the stimulation. The waveform of dry sound is presented in figure 62.

The impulse response data can be obtained in various ways, as mentioned in section 5.3.2. Due to practical reasons, we are unable to record an impulse response, so we have opted to use an online impulse response library for the target reverberation effect. Numerous online impulse response libraries provide recordings under different conditions. We have selected Cathedral Room in Shasta Lake Caverns as our target environment for its novelty in combining with a guitar sound. The impulse response library of the target environment can be found in the online database[113]. Every impulse response in the library is recorded at a 44.1kHz sampling rate, which strikes a balance between computational efficiency and sound quality. The environment primarily consists of limestone and concrete.

5.7.2 Parameter matching Implementation

In the implementation of parameter matching, our objective is to compute parameters that empower the reverb model(5.5) to imitate the reverberation effect of the Cathedral Room of the Shasta Lake Caverns(5.7.1). It is important to note that recorded impulse responses can be *both monophonic and stereophonic* in reality. Consequently, for our prototype, we selected both monophonic and stereophonic

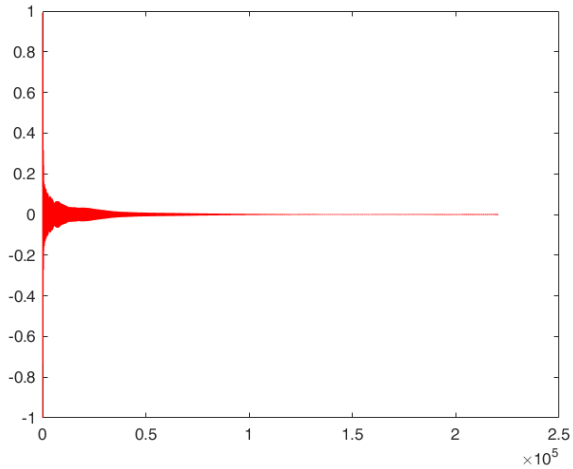
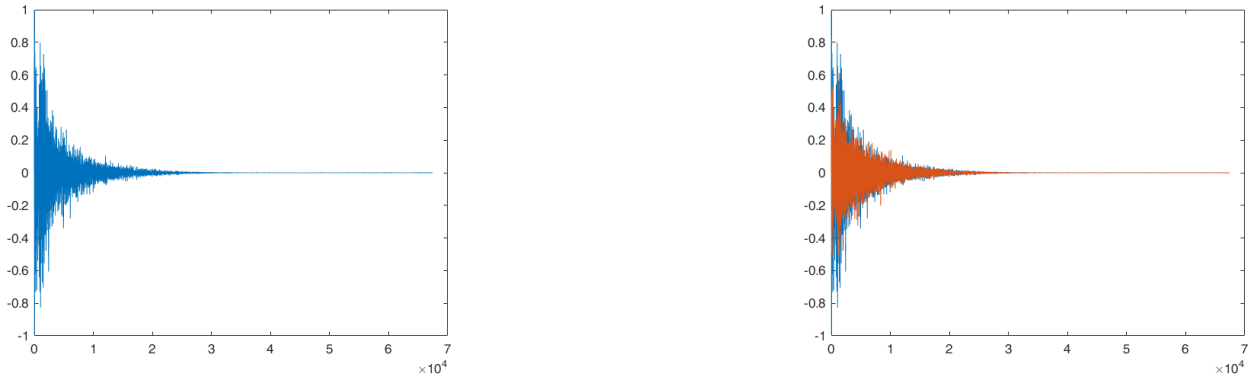


Figure 62: Guitar dry sound

impulse responses(figure 63) of the same target environment and applied the parameter matching algorithm(section 5.6) to each of them.

In stimulating the algorithm on both monophonic and stereophonic impulse responses, we selected an initial population of 40. This size was chosen with the understanding that a larger population necessitates greater processing power. We further determined that the number of iterations should be set to 20.



(a) Monophonic Impulse Response

(b) Stereophonic Impulse Response

Figure 63: Monophonic and Stereophonic Impulse Response of the Cathedral Room of the Shasta Lake Caverns

5.7.3 Parameter Matching Implementation Result

Our observations revealed that the fitness value declined progressively, ultimately reaching a value of convergence after 20 generations of iteration. This convergence of the fitness value serves as an indicator that the optimal parameters have been computed. The reverberation model parameters, derived from

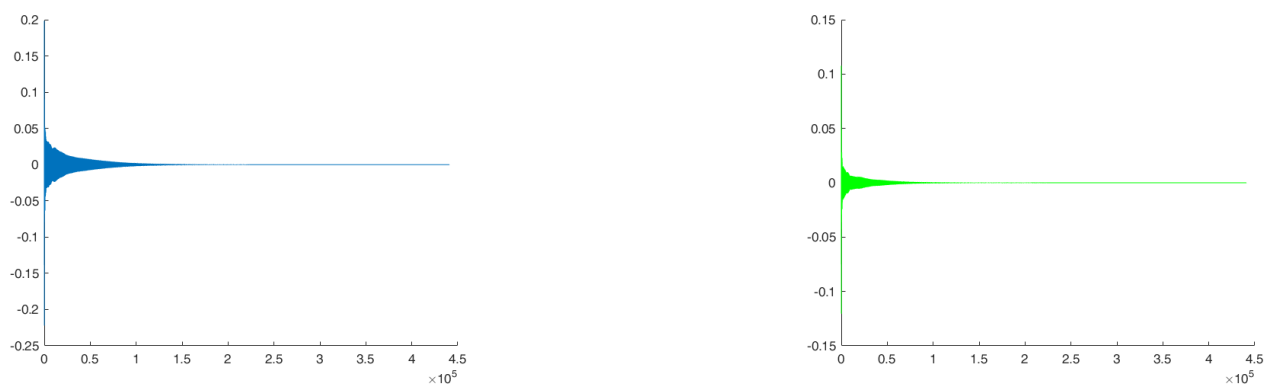
monophonic and stereophonic impulse responses, are presented in the following table(table 9) for ease of viewing and comparative analysis.

Parameters	Monophonic IR	Stereophonic IR
d	0.200	0.1085
b	$\begin{bmatrix} 0.6370 & 0.3197 & 0.5394 & 0.3933 \end{bmatrix}$	$\begin{bmatrix} 0.3973 & 0.3113 & 0.5367 & 0.5931 \end{bmatrix}$
c	$\begin{bmatrix} 0.4272 & 0.3865 & 0.5392 & 0.4670 \end{bmatrix}$	$\begin{bmatrix} 0.7645 & 0.6077 & 0.5158 & 0.3612 \end{bmatrix}$
G	$\begin{bmatrix} 0 & -0.0429 & 0 & -0.0429 \\ -0.0429 & 0 & 0.0429 & 0 \\ -0.0429 & 0 & 0.0429 & 0 \\ 0 & -0.0429 & 0 & 0.0429 \end{bmatrix}$	$\begin{bmatrix} 0.0059 & 0 & 0 & -0.0059 \\ -0.0059 & 0 & 0 & -0.0059 \\ 0 & 0.0059 & 0.059 & 0 \\ 0 & -0.0059 & 0.0059 & 0 \end{bmatrix}$
f	$\begin{bmatrix} 445.7124 & 638.4107 & 482.9626 & 717.6430 \end{bmatrix}$	$\begin{bmatrix} 462.8727 & 527.4031 & 284.5548 & 259.0788 \end{bmatrix}$
M	$\begin{bmatrix} 770 & 406 & 402 & 781 \end{bmatrix}$	$\begin{bmatrix} 882 & 966 & 502 & 686 \end{bmatrix}$

Table 9: Result of Parameter Matching for Monophonic and Stereophonic Impulse Response

5.7.4 Reverberation Implementation

In this subsection, we input the parameters calculated in the previous section (section 5.7.2) into the feedback delay network to generate the reverberated output. The resulting output is anticipated to mimic the acoustics of a guitar being played within the Cathedral Room. For a more detailed analysis, we have plotted the reverberated sound produced by two distinct sets of parameters in figure 64.



(a) Reverberation output(monophonic IR)

(b) Reverberation output(stereophonic IR)

Figure 64: Output of Reverberation Model based on monophonic and stereophonic IR

5.7.5 Evaluation

This section evaluates the result of implementation. Referring to performance criteria in section 5.4.1, we need to examine whether our model prototype meets the four main target criteria. Among the four criteria, we will focus on the examination of sound quality and efficiency particularly. This is because the flexibility is apparent with the availability to control 6 reverberation parameters. While latency is vital for

live performances, its evaluation requires coding the reverberation model on a real-time sound computing platform, a task for future work. We also conducted a comparison test between output based on monophonic and stereophonic impulse responses to determine the most suitable target impulse response for better parameter matching results.

Comparison Test

In section 5.7.1, we stated that target impulse responses could be either monophonic or stereophonic, and we have implemented both scenarios. However, it is evident from Figure 64 that these two types of inputs yield distinct results. Consequently, we intend to investigate which type of input produces superior results in simulating the reverberation effect for our module.

The comparative test we have selected involves calculating and comparing the Euclidean distance between the final reverberation outputs (Figure 64) and the target impulse response (Figure 63). This approach is pertinent since the goal of our implementation is to generate a reverberation that closely aligns with a target impulse response. By computing the distance between the produced reverberation and the target impulse response, we can effectively gauge the performance under different inputs. The Euclidean distance is frequently selected as the primary distance metric due to its simplicity and intuitive nature.

Reverberation Output	Based on monophonic target IR	Based on stereophonic target IR
Euclidean distance	15.0785	14.7744

Table 10: Euclidean distance between output based on monophonic/stereophonic IR and target IR

We can see from the Euclidean test that our reverberation module imitates the target reverberation effect better with stereophonic target impulse response. This can be interpreted intuitively that stereophonic impulse response provides more information about the environment by multiple channels of signal.

Overall Sound Quality Examination

After an internal comparison between the outputs, we also examined the overall quality of the reverberation. In evaluating the quality of the reverberation output generated from our model, both objective measurements and subjective listening tests were conducted. The listening tests demonstrated that the output possesses a rich sonic quality. The richness attributes to the successful rendering of late reverberation(introduced in section 5.3.2). Furthermore, we have calculated the echo density[114] of the output in objective measurements and confirmed the presence of high echo density in the output. A rich echo

density, as calculated in our outputs, ensures a smooth decay without audible echoes or coloration, thus enhancing the overall listening experience. In general, the quality of reverberation output looks promising, which meets one of the model selection criteria in section 5.4.1.

In our evaluation process, we also identified certain limitations in the quality of the reverberation output. Although the output demonstrates commendable performance in rendering late reverberation, it falls short in effectively producing early reflections. A comparison of the better-performing reverberation output based on stereophonic IR, with the original dry sound (figure 65) reveals this limitation. The figure clearly shows a pronounced late reverberation tail, but the early reflections are hardly noticeable compared to the original sound. As early reflections also carry environmental information, their under-representation could affect the accuracy of reverberation effect imitation.

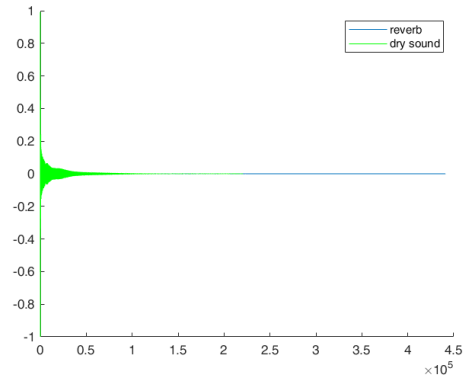


Figure 65: Comparison between reverberation output and dry sound to observe early reflection and late reverberation

Efficiency Test

The implementation of prototype is completed on a laptop equipped with Intel-Core i7 10750H processor and 16GB RAM. The MATLAB 's memory footprint of this implementation is approximately 800MB, which represents a mere 5% of the total RAM, ensuring that the majority of memory resources remain available for other computational tasks. This low memory requirement fulfills our performance criteria of efficiency in section 5.4.1, again justifying the suitability of our model.

By evaluating the implementation output based on main performing criteria(5.4.1), we are confident to state that the prototype of the reverberation module meets most of the performance requirements. This effectively means that we are good to move on to the further product development stage.

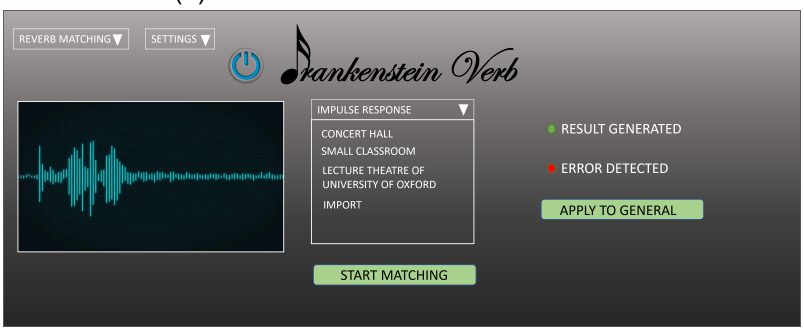
5.8 Software Interface Design

The final component of our design involves creating a user interface for the reverberation software. This user interface has been carefully crafted to bridge the gap between users and our reverberation system. Guided by the principle of prioritising user-centric considerations, we have ensured that the interface is not only aesthetically pleasing but also user-friendly. This section will walk you through the key components

of the interface and highlight some innovative designs that set our product apart.

Given that the reverberation software primarily comprises two parts: the reverberation model and the parameter matching algorithm, the user interface is correspondingly divided into two sub-interfaces: 'GENERAL' for the reverb model and 'MATCHING' for the parameter matching algorithm. These two sub-interfaces interact seamlessly with each other, allowing users to switch conveniently between them. It is also worth noting that the software interface and hardware interface designed in section(6.6) correspond to each other for clarity of the product.

(a) Software Sub-Interface 'GENERAL'



(b) Software Sub-Interface 'MATCHING'

Figure 66: Design of Software Interface of Reverberation Module

5.8.1 'GENERAL' Sub-Interface for Reverberation Model

The "GENERAL" interface(figure 66a) offers users control over the reverberation model (feedback delay network). Users can freely switch between audio and MIDI input based on their requirements. The interface includes six knobs on the right side, representing the six parameters for control. The central knob is designed to be the preset knob and will be explained later. In the following paragraphs, we will discuss *three core designs* in this user interface: the preset knob, the LED light stripe, and the mode-switch control.

Preset Knob

The preset knob(figure 67a) is specifically designed to enhance convenience for our customers. It addresses the scenario where users, particularly beginners, desire to apply common reverberation effects without having extensive knowledge of parameter control. Using the parameter matching algorithm for every common reverberation effect can be inefficient. To address this, we introduce the preset knob, which incorporates pre-calculated parameters for several commonly used reverberation effects. These effects are thoughtfully selected based on their popularity in music production[115][116]. By simply turning the preset knob to any of the indicated reverberation effects, the corresponding parameters are automatically applied to the feedback delay network, thus generating the desired effects. This solution significantly saves users' time and enhances the user-friendliness of our product, particularly for beginners.

LED Light Ring

The circular light ring(figure 67b) located above each knob is an additional user-friendly design feature of the product. These light rings indicate the knob positions of parameters, helping users identify where to turn them. This function plays a crucial role in connecting the parameter matching algorithm and the reverberation model, as it shows the calculated parameters in the main reverberation model. Users can effortlessly create the desired reverberation effect by aligning the knob with the LED indicator corresponding to the calculated parameters. The corresponding hardware implementation is in section 7.2.

Mode-switch Control

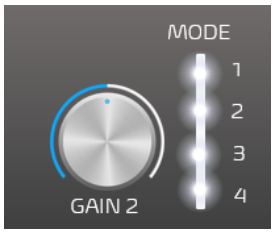
We designed mode-switch control(figure 67c) considering the characteristics of the parameters of the reverberation model. Some of these parameters are vectors(5.6.1), and it is essential for each value in the vector to be independently controllable. Due to limitations in the size of the synthesizer module, it is not feasible to have a separate knob for each value in the vectors. However, to ensure independent control is still achievable, we have implemented the mode switch control. In this design, each mode corresponds to the control of a specific value in the vector parameter. Users can effortlessly switch between controlling different values in the vector parameters by simply clicking on each mode.

5.8.2 'MATCHING' Sub-Interface for Parameter Matching Algorithm

The "MATCHING" sub-interface(figure 66b) is specifically designed for the parameter matching function of the reverberation module. Users have the option to select from our collection of sample impulse responses or import their own generated impulse response of the target environment. Our database consists of sample impulse responses that have been either recorded by us or sourced from online

(a) Preset Knob

(b) Light Ring



(c) Mode-switch

Figure 67: Core designs of Software Interface

libraries. In cases where we include specific impulse responses from external sources in our database, we will obtain the necessary permissions.

The system generates the calculated parameters after clicking the ”START MATCHING” button. By subsequently clicking on ”APPLY TO GENERAL”, these calculated parameters are then reflected in the GENERAL interface through LED light stripes, indicating the corresponding positions to turn the knobs. Users are able to create desired reverberation effects once they go back to the GENERAL interface and turn the knobs following the guidance of the light ring.

5.9 Conclusion

In conclusion, we have designed a reverberation module with our target customer - beginners in synthesisers in mind. The module contains a feedback delay network model to produce reverberation effects, along with an additional function of finding parameters of the reverberation model to achieve desired environmental effects. We have also presented an interactive software interface with multiple beginner-friendly design features.

Our module can produce rich late reverberation that enhances the listening experience. Users are suggested to input stereophonic impulse response to achieve better matching of desired reverberation effects. In the future, we will focus on developing a model that can better produce early reflection to improve the listening experience and authenticity of reverberation imitation. We will also implement our model on a real-time sound computing platform to test for latency.

6 Hardware and Overlay Design - Zhaoqiu Cheng

6.1 Why do we Want to Make a Hardware Interface

Survey of Experienced Musicians A survey of musicians and engineers in 2022 [117] compared their preferences for physical and virtual manipulators in music software. Most respondents **preferred a physical controller** (e.g., knob, slider, button) of a Virtual Studio Technology (VST) as they bring more **fun and "control intimacy"**, a concept that refers to the perceived alignment between the performer's psychophysiological ability to control the instrument and the resulting response. Example features that reduce "control intimacy'' include a mouse or a keyboard of a virtual instrument.

Discussion with an Experienced Musician In the early stage of our design, we had the opportunity to talk to Sam Salem, a composer and engineer at the Royal Northern College of Music who owns multiple modular synthesisers and specialises in machine learning and artificial intelligence (AI). We described our intention, which was to combine AI models with synthesiser hardware. Sam was intrigued by our project and agreed that AI models and plug-ins do have their advantages of being flexible and affordable. However, what fascinated him most was the agility of modular synthesisers and the process of creating new sounds by patching various modules together, which might be achievable with plug-ins such as VCV Rack but is much more manageable with hardware synthesisers.

6.2 The Objectives of Hardware Design

In the market research analysis, we discussed increasing the "control intimacy'' of the software as the motivation for building a hardware system to control the software system. The parameter matching, timbre transfer, and reverberation functions described above use different synthesis techniques and algorithms, which are impossible to combine and control with one circuit board. Thus, we decided to design three different modules. This made marketing more difficult as we decided to sell our product as a bundle rather than a single unit, nevertheless, our product could be an entry point into modular synthesis for beginners. We had aimed to present the three modules' circuit and user interface designs, but we did not have sufficient time or page allowance to do so. Instead, only a part of the design will be presented in the report. Here are the designs we have done and would like to discuss in the following report:

1. Frankenstein VCO-VCF-EG hardware module based on the software plug-in Diva with motorized knob design, where Siheng's Parameter Matching algorithm3 is applied.

2. The Frankenstein Verb hardware module involves Yuanjun's Reverberation5 algorithm.

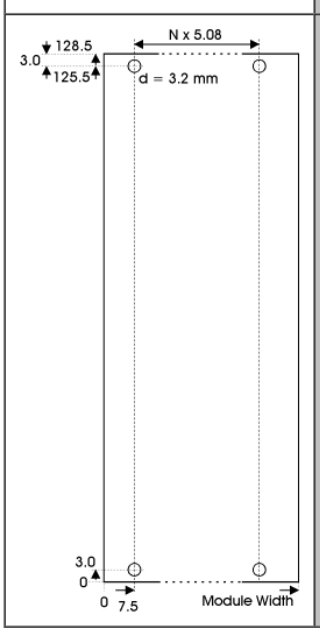
3. The Frankenstein Instrument hardware module adopts the DDSP instrument algorithm4 that Haedam worked on. Unlike the other two modules, this module cannot work without the algorithm as the main computation of instrument transformation is carried out by Haedam's algorithm.

6.3 Specifications to Eurorack Hardware Design

Some mechanical and electrical properties of the Eurorack modules need to be specified before diving into the design of Eurorack modules.

6.3.1 Mechanical Specifications

As mentioned in the introduction, we decided to choose the Eurorack format for our project. Thus, each module's mechanical design must be consistent with the Eurorack requirements initially specified in 1995 by Dopefer Musikelektronik. The figure below68 shows how the sizes of the front panel modules are determined [118]. With the positions of the mounting holes held constant, the length between the holes can be varied, which is measured in the unit of HP, where 1HP = 5.08 mm. Heights of all modules are 128.5 mm, allowing them to fit in the rack system, 3U (133.35mm) height standard rack system. As the



Module width [HP]	calculated module width [mm] (= multiples of 5.08 mm)	actual module width [mm]
1	5.08	5.00
1.5	7.62	7.50
2	10.16	9.80
4	20.32	20.00
6	30.48	30.00
8	40.64	40.30
10	50.80	50.50
12	60.96	60.60
14	71.12	70.80
16	81.28	80.90
18	91,44	91,30
20	101.60	101.30
21	106.68	106.30
22	111.76	111.40
28	142.24	141.90
42	213.36	213.00

Figure 68: Mechanical properties of the front panel of modules [118]

modules need to be mounted into a Eurorack box, the depth of the modules also needs to be considered.

For example, the usable depth for modules of the A-100 cases from Doepfer varies from 70 mm to 150 mm [118]. There are also “skiff cases’’ designed for shallower modules with depths between 25 mm and 40 mm.

6.3.2 Electrical Specifications

The figure below69 shows the internal layout of a standard 19 inches rack system A-100, which contains two system buses (1), the internal power supply (2), and the main electrical supply socket (3). The rack system power supply produces voltages of +12V ground and -12 V pins and can put out a maximum current of 650 mA, so the total current of all the modules mounted into the case cannot exceed 650 mA. The audio signals of the modules typically give a maximum of 10 V peak-to-peak electric signal, whereas

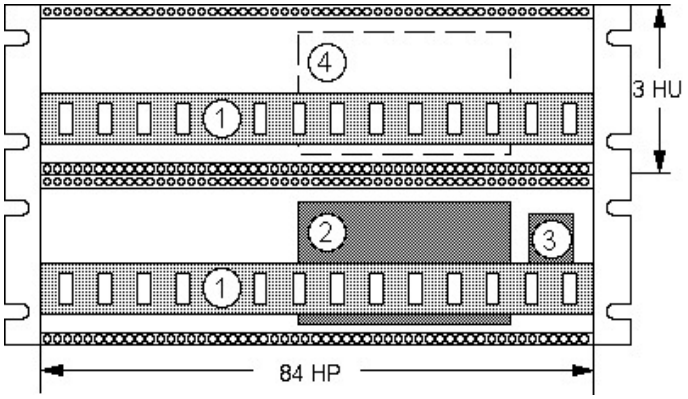


Figure 69: Internal layout of a standard 19 inches Eurorack case (A-100 by Doepfer Musikelektronik) [118]

the control voltages for the audio modulators generally are from -2.5 V to +2.5 V for LFO and from 0 V to +8 V for the ADSR. [119]

6.4 Frankenstein VCO-VCF-EG Module

The Frankenstein VCO-VCF-EG module design is based on the software plug-in Diva, an analogue synthesiser emulator inspired by some of the most well-known models in the industry. This module aims to create hardware designs for the Diva software since the current parameter-matching model calculates the parameters in the Diva synthesiser software.

6.4.1 Advantage of Frankenstein over traditional analogue synthesisers

To try the parameter-matching algorithm, users can use the Diva software, but hardware control introduces some advantages. First, as mentioned in the market analysis, hardware control gives users more

“control intimacy”. Users perceive a higher agreement in their control of the system and the resulting response they get from the system. Moreover, what we aim to create is a modular synthesiser instead of a conventional synthesiser, which allows for altering of the signal path through DIY patching. This gives users a more dynamic human-machine interaction experience than just having a fixed signal path. And finally, our product differs from other products in the market because we designed the motorized knobs for beginners unfamiliar with the Diva software and the algorithm. The motorized knobs free their hands and mind as they do not need to consider the meanings of the parameters.

6.4.2 Component Selection

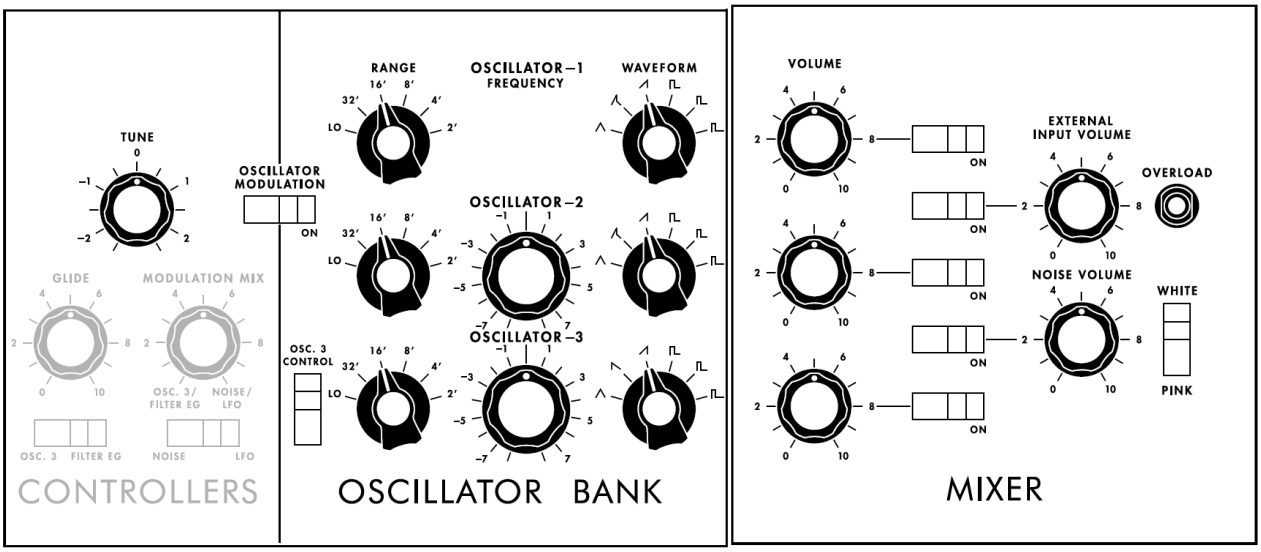
The first thing that needs to be considered when designing the module is what functions it serves. There are 281 Diva parameters that the algorithm can model to recreate a designated sound adequately. Still, it is impossible to include all the modules in one synthesiser, and it would be computationally intensive for the computer. Therefore, Siheng picked 32 relatively important parameters for which we can design the hardware. The 32 adjustable parameters include 8 from the two attack-sustain-decay-release (ASDR) envelope generators (4 for each of the EGs), 6 from the two low-frequency oscillators (delay, phase, and rate for each of the LFOs), 11 from the triple oscillator, and 7 from the voltage-controlled filter. It is possible to have different components on a single module. However, having all the features that form the 32 parameters is still impossible for a modular synthesiser, considering its size and complexity in the signal path . Therefore, we figured that part of the components can be integrated into the Frankenstein Basic module, while the rest can be designed and sold as extension modules to the basic module.
We decided only to include the triple VCO, the VCF, and one EG in our design. Our criterion for evaluating the importance of each component was the number of parameters related to it. We chose the VCO as part of the module because it is the sound source of the product, and 11 out of the 32 picked parameters are related to it, followed by the VCF, which has 7, and the EG, which has 4.

6.4.3 General Signal Path

We would like the three components, VCO, VCF, and EG to each work individually as sound sources or modulators for other modules. Thus, they are internally separated from each other, i.e., their circuitries are independent from each other. However, the three components need to be electronically connected by patching certain jacks with patch cables to use the parameter matching model. This design is inspired by the Behringer 110 VCF/VCO/VCA Eurorack module [120]. After patching the components, the audio

signal exits the oscillator and enters the filter, which is modulated by the envelope generator, then exit the filter. To better understand this signal path, we should look at the individual components.

Individual Component Breakdown We analysed the Diva software and found the antique analogue synthesisers this software is based on. The triple VCO is based on Moog Minimoog Model-D [121].70 This name comes from its three oscillators, with the second and the third waves superimposed onto the first one in a mixer. The mixer controls the volume of each oscillator and the adjustable noise source. Users can also plug in an external input into the mixer to add more texture, which is further discussed in 6.7.



(a) The triple oscillator panel (b) The mixer panel

Figure 70: The Minimoog hardware panels [121]

Both the VCF and the EG are based on the Korg analogue synth MS-20[122]. The VCF consists of a voltage controlled high-pass filter (VCHPF) followed by a low-pass filter (VCLPF), where the cut-off frequency and peak for both filters can be adjusted. The envelope generator is controlled by a trigger signal. An updated model of the signal path is shown in the block diagram below.71

By setting jacks for trigger input for the envelope generator, external signal input for high-pass filter and low-pass filter, users can control the hardware components with what they have, or they can use our sources that are integrated within the hardware.

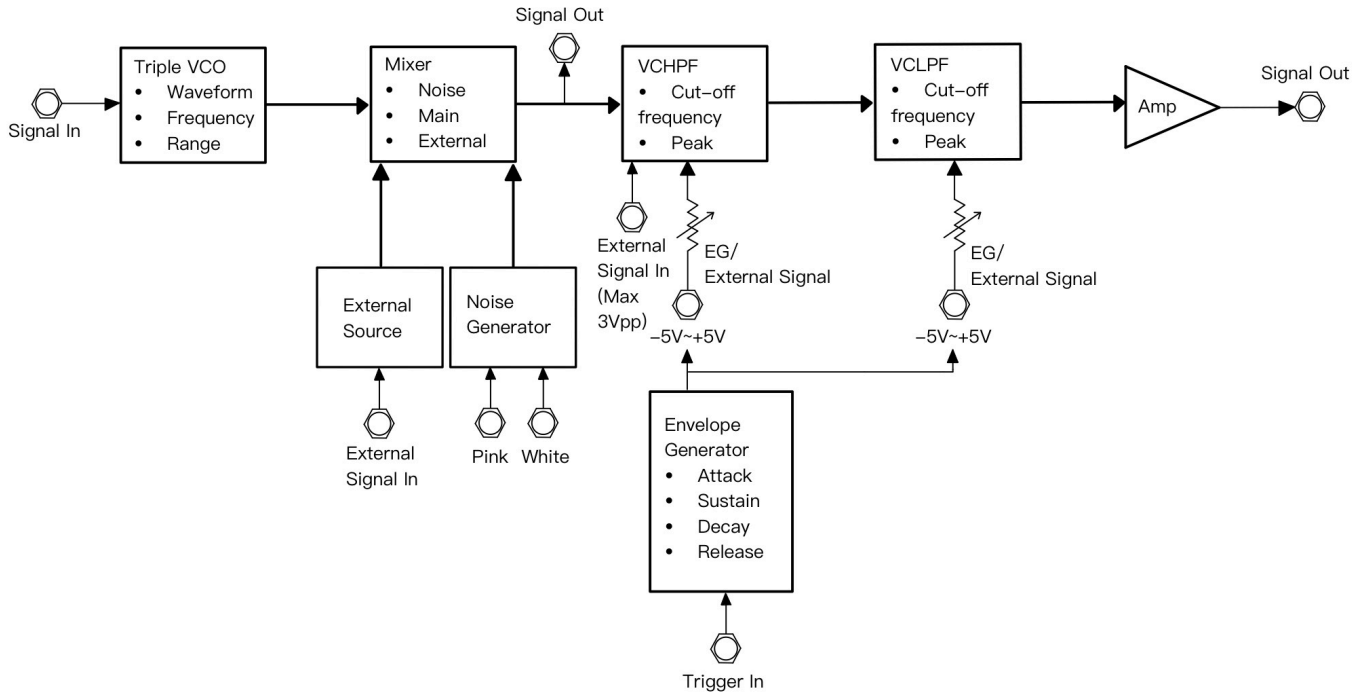


Figure 71: Block diagram of the module

6.4.4 Overlay Design and User Guide

We designed a 20HP Eurorack module72 to fit the 21 adjustable knobs and patch 8 cable plugs.72 The module is divided into 3 sections, namely VCO with a mixer, VCF, and an EG, clearly indicated by the lines and the labels at the bottom left corners of each section. In the VCO component, the 3 oscillators are arranged horizontally. Each has 3 to 4 adjustable parameters. The RANGE knobs in the first column are set for users to choose the fundamental frequencies within a range spanning five octaves. The sixth LO setting generates an infrasonic frequency, which allows the oscillator to be used for other purposes, similarly to an LFO.

The FREQ knobs in the second column can be used to detune the oscillators 2 and 3 from the pitch of oscillator 1. The FREQ knobs are designed to be slightly larger than the rest so users can get a finer tuning of the frequencies of the oscillators. To guide the users, two arcs are drawn outside each knob, with the green one indicating pitches below the fundamental frequencies and the pink one indicating pitches above them. The default setting of the ticks between the “+” and “- ‘’ signs. The tick in the figure is an example of the sound tuned six pitches below the fundamental frequency. The range of these fine-tuning knobs is from 7 pitches below the fundamental frequencies to 7 pitches above the fundamental frequencies.

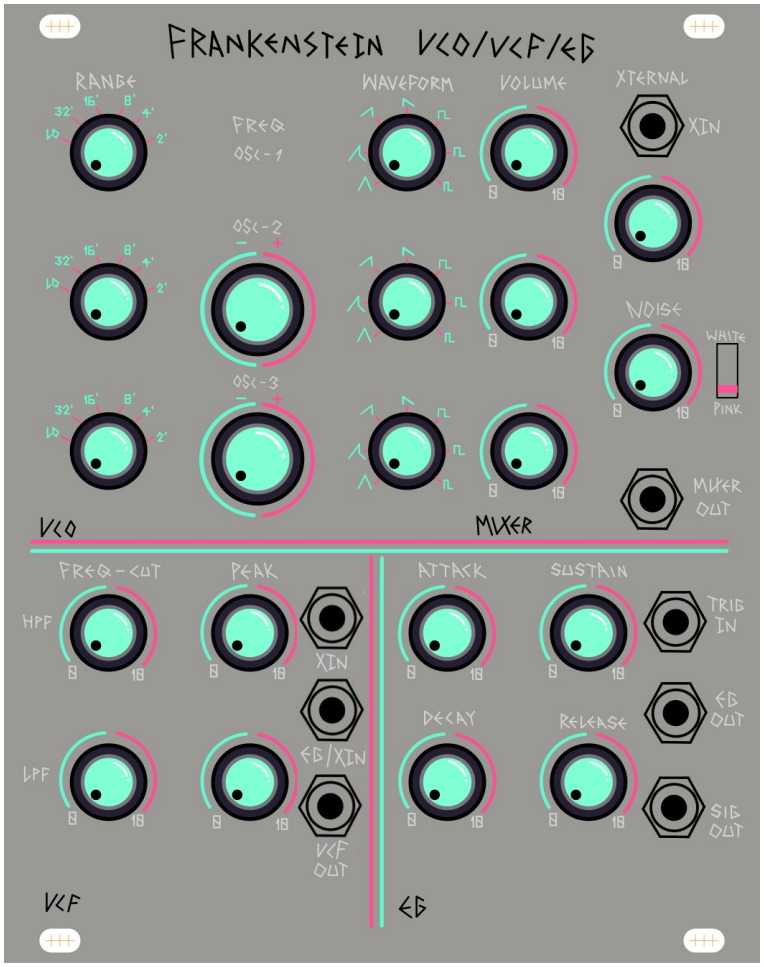


Figure 72: The overlay design of the Frankenstein VCO-VCF-EG Module.

The WAVEFORM knobs in the third column provide the users with a selection of 7 unique waveforms, each waveform has a unique harmonic content, as mentioned in the introduction section2. The waveforms include triangle, sawtooth, square, and rectangle waves.

Finally, the VOLUME knobs are in the fourth column. The volumes of the sound waves generated by the oscillators can be adjusted with these three knobs in the mixer, and the volume is linearised from 0 to 10. Besides the basic volume adjustment for the three integrated oscillators, an external source can also be plugged in through the jack XIN. The volume of the external source can be adjusted with the knob underneath the jack. When there is no external source applied, the signal input from the oscillator is slightly attenuated and normalized to the external input signal path. The main signal output is sent back to the input of the mixer. When an external source is introduced into the mixer, the mixer can get overloaded if the source is large enough, leading to an undesirable distortion. In this case the users will have to adjust the volume of their external sources to fit in the 10 V limit [123].

In the NOISE section, the switch enables users to choose between white and pink noise. The NOISE

knob is then used to adjust the volume of the noise.

In the VCF section, the two filters, high-pass filter (HPF) and low-pass filter (LPF), are arranged horizontally. The FREQ-CUT knobs are used to adjust the cut-off frequencies of the filters and the PEAK knobs to adjust the emphasis on the peak.73 The peak refers to the point right before the cut-off frequency. The peak occurs when the resonance parameter is set to a high value, causing a significant boost in the amplitude of the frequencies just before the cut-off frequency. This results in a more pronounced and expressive sound, adding character and richness to the timbre of the synthesiser. [124]

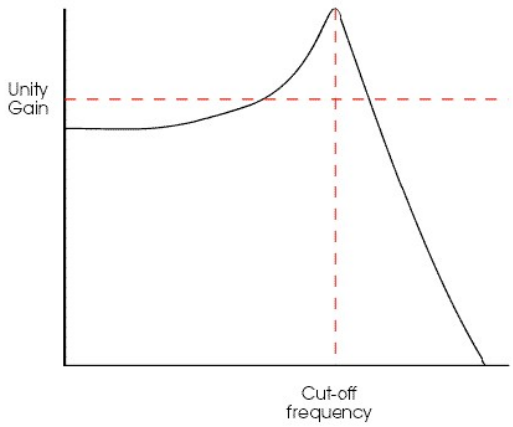


Figure 73: A low-pass filter with a high resonance coefficient[124].

An external signal can act as a modulator of the VCF when it is patched with the XIN jack in the VCF component. To use the integrated EG to modulate the VCF, users have to patch the EG/XIN in the VCF section with the EG OUT in the EG section. If the users want to monitor the sound after the signal passes through the VCF, they can connect their speaker to the VCF OUT jack.

The EG component contains the four essential components for an envelope, namely attack, decay, sustain and release. The knobs are designed to adjust the time for each stage of the envelope, and the range is mapped from 0 to 10. The EG has an integrated trigger signal, but we also have a TRIG IN jack where the users can patch their own trigger signal in.

In the following section, we will break down the design of VCO in the Frankenstein VCO/VCF/EG module.

6.4.5 VCO Design

Objectives of Oscillator Design When designing the VCO, several objectives need to be taken into consideration:

1. Frequency Stability: The VCO should exhibit good frequency stability over time and temperature

variations. Stability is crucial in applications where precise and consistent frequency generation is required.

2. Waveforms: The VCO should not only have a sine wave generator, but also contain other waveforms that users choose from. For our design, we want to include rectangle wave, and square wave.

3. Control Voltage Range and Sensitivity: The VCO should have a suitable control voltage range that matches the requirements, which in our case, is a maximum of 10V peak-to-peak signal.

To build a VCO, we need to understand the input and output signals of the VCO, as well as the parameters we are adjusting and how they are controlling the output sound. This is illustrated in the block diagram below.74

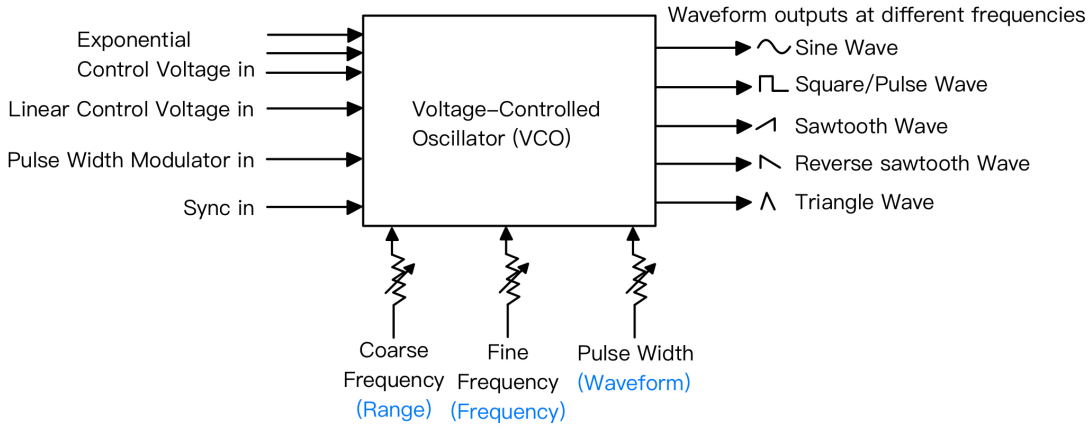


Figure 74: Block diagram of the VCO. This shows the inputs, the adjustable parameters, and the outputs. The names of the parameters appear on the hardware module correspond to what is being tuned.

As mentioned in the introduction 2, most analogue VCOs adopted a 'one volt per octave design', which means the frequency of the sound must increase exponentially to be in tune as the control voltage increases linearly. Usually, two types of potentiometers are used to adjust the frequency of the oscillator, both of which are control voltage inputs. The coarse frequency adjustment corresponds to an exponential frequency response input, and the fine frequency adjustment corresponds to a linear response input. For example, if the oscillator is oscillating at 100 Hz when the control voltage is at 1V, turning the coarse frequency control to 3V will result in the frequency to get to 400 Hz, while the same action on a fine frequency control will only take the frequency to 300 Hz.

Another common adjustable parameter is the pulse width of the rectangular pulse wave or the square

wave. As can be seen in the VCO panel of Minimoog Model-D, there are three pulse widths to choose from, which is done by an integrated pulse width modulator. [125]

Type of Oscillator Core Chosen - Colpitts Oscillator

There are various types of oscillator cores in the modern synthesis world, such as Colpitts, Hartley, and Pierce oscillators. The triple VCO we are making features the Colpitts oscillator circuit as the core for each VCO. We chose the Colpitts oscillator for a few reasons. Firstly, comparing to the Hartley and Pierce oscillators, the Colpitts oscillator typically requires fewer components, which simplifies the circuit design, reduces the component count, and shorten the implementation time. Secondly, the Colpitts oscillator exhibits better frequency stability compared to the Hartley and Pierce oscillators. The Colpitts configuration allows for better control over the resonant frequency and can provide more precise frequency generation. Most importantly, the Colpitts oscillators can provide lower distortion levels and lower output impedance in the output waveform, leading to cleaner and more accurate signals, which is especially important for oscillators since one of our main objectives is to make an oscillator in tune. [126] Considering all the above advantages, we have decided to adopt the Colpitts Oscillator as the core for our oscillator.

How Colpitts Oscillators Work and Improvement on Stability of the Circuit

The Colpitts oscillator is a type of Inductor-Capacitor (LC) oscillator that uses two capacitors, an inductor, and a single stage bipolar transistor amplifier as the gain element to construct the voltage driver. The configuration of the Colpitts transistor oscillator is shown below. The forward bias is initially applied to the emitter-base (a-b) junction and the reverse bias is applied to the collector-base (b-c) junction. To sustain the oscillation, a 180-degree phase shift need to be applied to obtain the positive feedback needed since the base (b) and the collector (c) elements are out of phase in the common-emitter circuit. Thus, grounding the common capacitor junction can achieve this since the polarity of the voltage across C_1 is opposite to the polarity of the voltage across C_2. [127] The advantage of this circuit design is the frequency stability of the oscillator, due to the reduction in self and mutual inductance within the tank circuit. The oscillating frequency of the Colpitts circuit is

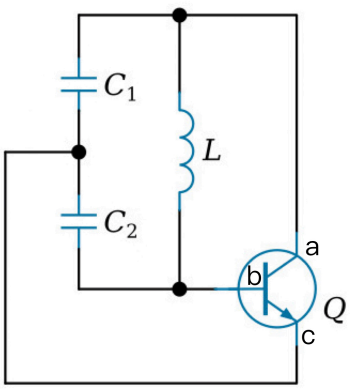
$$f = \frac{1}{2\pi\sqrt{LC_t}}$$

, where C_t is the total capacitance of capacitors C_1 and C_2 connected in series, which is

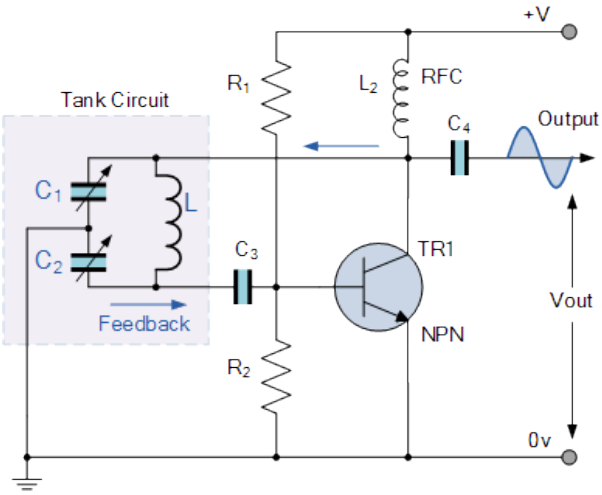
$$C_t = \frac{C_1 \times C_2}{C_1 + C_2}$$

.

The figure below shows the configuration of a basic oscillator circuit.75 When the power supply is applied, the capacitors C_1 and C_2 charge up and discharge through coil L. The oscillation generated by the capacitors is fed to the emitter-base junction, which is amplified at the collector output. To maintain a steady and constant level of DC voltage in the electronic circuit, despite changes in temperature, ageing of components, and other factors that may affect the circuit's performance over time, stabilising DC bias is a must. The DC bias stabilising method used in this case is the voltage divider bias, where the two resistors, R_1 and R_2, are connected in series between the voltage supply and the ground, and the base of the transistor is connected to the junction between the resistors.



(a) LC tank circuit for Colpitts oscillator. [127]



(b) Basic Colpitts Oscillator circuit. [128]

Figure 75: Partial and general circuit design of Colpitts Oscillator Circuit.

For audio generators, it is necessary to remove the DC component of a signal before it is processed further. This is because DC can cause saturation or biasing of the circuit, leading to distortion and other unwanted effects in the output signal. To block the DC component of a signal, a capacitor is connected in series with the signal path. Capacitors have the property of blocking DC while allowing AC to pass through, as they charge and discharge in response to AC changes. However, if a capacitor alone is used to block DC, it would also block the AC component of the signal, resulting in no signal passing through the circuit.

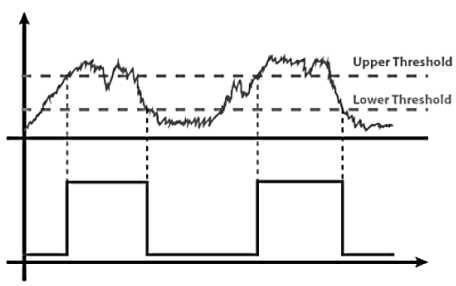
Additional Waveform Generation Design – Square/Rectangle Wave

We have established in the previous section that a basic Colpitts oscillator can generate sine waves. The design objective in this section is to use circuitries to convert the sine waves to specific waveforms we want.

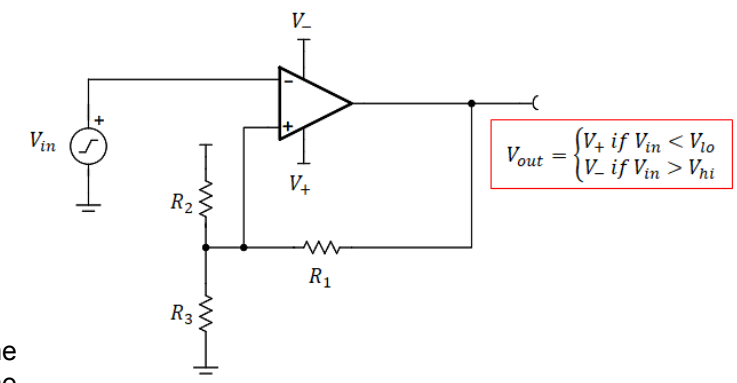
Square/Rectangle Wave – Schmitt Trigger

Schmitt Trigger is a comparator circuit that uses positive feedback such that small variations in the input result in large variations in the output without changing the phase. The key characteristic of a Schmitt trigger is its hysteresis property. Hysteresis refers to the dependence of the output state on the previous state of the input signal which causes the output to switch at different threshold levels for rising and falling input voltages. As the input sine wave crosses the higher threshold of the Schmitt trigger, the output switches to a high voltage level. When the input sine wave drops below the lower threshold, the output switches to a low voltage level. This process repeats as the sine wave oscillates, resulting in a square wave output.76a

Here is a simple Schmitt Trigger circuit with positive feedback applied to the non-inverting input. As mentioned above, the output voltage depends on the threshold voltage values V_hi and V_lo.76b The output



(a) The signal above is the original waveform, and the signal below is the waveform that has been altered by the Schmitt Trigger.



(b) Schmitt trigger circuit with an op-amp.

Figure 76: Threshold and circuit of Schmitt trigger. [129]

voltage is determined by the resistance of the resistors in the feedback loop. By adjusting the reference voltage at the non-inverting input, the voltage divider formed by these resistors sets the upper and lower threshold voltages simultaneously. [129]77

Hardware Selection for Motorised Knob Design As mentioned in section 1.1.2, we used a motorised knob design for the Frankenstein VCO/VCF/EG module to aid our users. As we have 21 knobs in the Frankenstein VCO/VCF/EG module, we need 21 motors, and correspondingly, 21 potentiometers. A few objectives need to be considered in this design:

1. The potentiometers must not only receive the signals, but also send feedback to the controller board.

2. The motors must be as small enough to fit within the module, which is 106.3 mm wide and 128.5

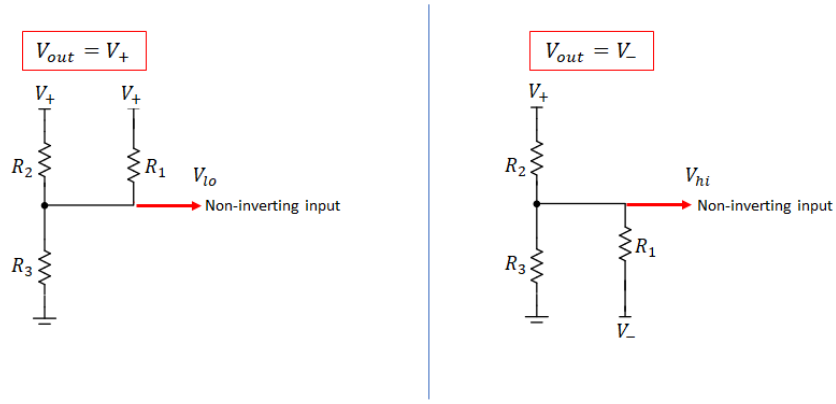


Figure 77: An illustration of the relationship between the input and output of Schmitt trigger.[129]

mm long. There is a trade-off between the size and the power of the motor, but we do not have specific requirements for power of the motors. Thus, we would like the motors to be as small as possible to reduce the cost of production and implementation complexity.

3. The motor must be precise enough to turn the knob to the designated position. After the knob is turned to the calculated positions, users will have the freedom to readjust the knobs.

Types of Motor – Servo Motor versus DC Motor

There are two types of motors we considered using, the Servo and the DC (direct current) motor.

Servo motors are commonly used in applications that require precise control of position, speed, and torque. Unlike DC motors, servo motors incorporate a closed-loop control system. They consist of a small DC motor connected to a position feedback mechanism which continuously provides information about the current position of the motor shaft to a control circuit. The control circuit of Servo motors receives input signals, typically in the form of pulse-width modulation signals, which specify the desired and the actual position or angle. Based on the difference between the desired position and the actual position obtained from the feedback mechanism, the control circuit adjusts the motor's voltage or current to accurately achieve the desired position.[130] Thus, the properties of Servo motors meet our criteria for precision.

However, once a Servo motor has moved to its commanded position, its position cannot be adjusted manually. [131] Although we can cut off the power supply when the motors get to their desired positions, this design is suboptimal as it introduces more uncertainties, such as the motors breaking down. Moreover, servo motors usually operate with a higher voltage supply compared to DC motors, as DC motors often operate at a voltage below 6V, whereas this is rarely the case for servo motors. Servo motors are also more expensive and difficult to implement than DC motors. [132] Considering all this, we decided to

choose DC motors for our product.

Choice of Motor and Potentiometer

Potentiometers are adjustable resistors with a resistive track and a movable contact called the wiper. By changing the wiper's position along the resistive track, the potentiometer can vary its resistance, allowing control over the amount of electrical current flowing through it. [133] As mentioned in the design objec-

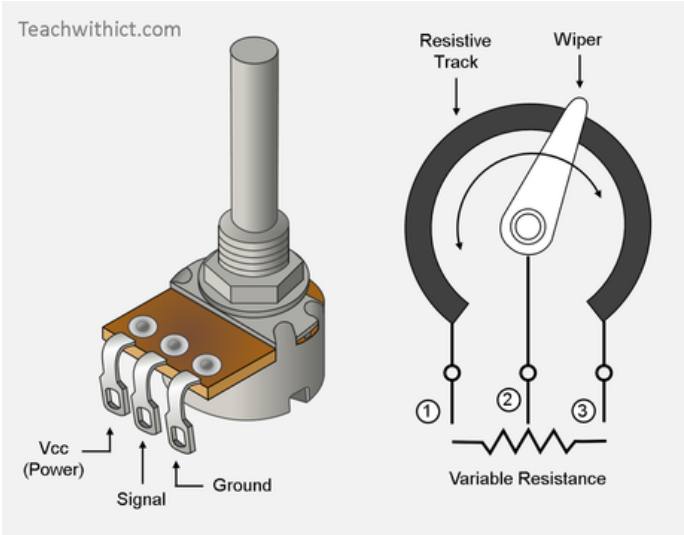
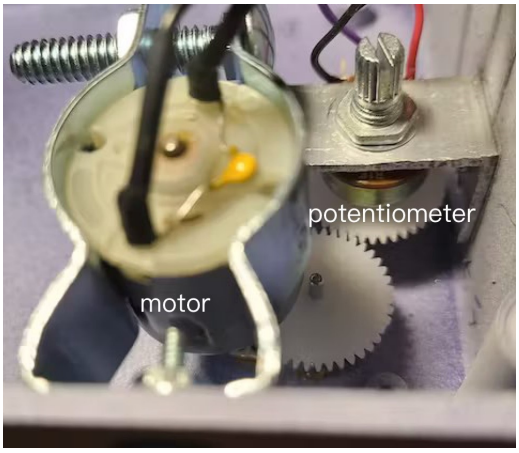


Figure 78: Schematic of a single-pot potentiometer. [134]

tives, the potentiometers need to receive command and give feedback to the controller board. Therefore, we thought of using two single-pot potentiometers, connected with gears, with one receiving commands and the other giving feedback. These are then connected to the motor with another gear so that the knob attach to the receiver potentiometer is powered. [135] Another option is to use dual-gang pot potentiometers. Two potentiometers are integrated on a shared shaft, which enables a simultaneous adjustment of two channels, allowing us to use one channel as the signal receiver and the other as the responder. This solution also takes up less space than two single-pot potentiometers. Thus, we chose to use dual-gang pot potentiometers in the Frankenstein Verb module, which can be seen in the figure below. The figure below79a shows the dual potentiometer, which will also be used in the Frankenstein Verb module for the LED guidance 6.6.

As stated in the design objectives, specific requirements for the output speed of the motor, but we would like it to be as small as possible. Therefore, we chose the model in figure X to be the motor for our prototype. The diameter of this model is 21mm, allowing the hardware synthesiser board to easily fit 5 of them in a row. [137]

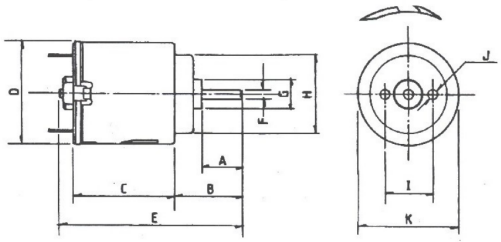
(a) Dual-pot potentiometer selected. (Bourns 100kΩ Rotary Potentiometer)[136]



(b) Connection scheme of the potentiometer and the motor. [135].

Figure 79: The selected potentiometer model and connection schematic.

(a) The motor chosen for the prototype. (RS PRO Brushed 238-9692, 1.21 W, 1.5 → 3 V dc, 10.4 gcm, 8200 → 14000 rpm, 2mm Shaft Diameter)[137]



Dimensions	A	B	C	D	E	F	G	H	I	J	K
mm	8.3	14	21	21	38	2	6.15	16	10	2	20.15

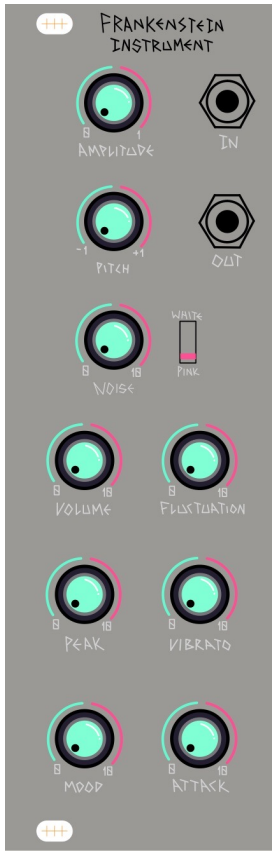
Stall Current RE-140 at 1.5v = 1.6A RE-140 at 3v 3.0A

(b) Dimension of the motor selected.[137]

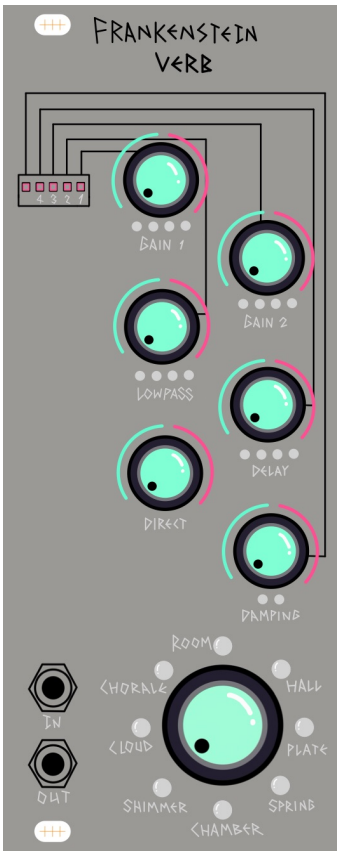
Figure 80: The selected motor model and its dimension.

6.5 Frankenstein Instrument Overlay Design

The Frankenstein Instrument module81a cannot work independently without the DDSP algorithm since the calculation of timbre transformation is carried out by the algorithm. Thus, the hardware module is only used as a pre-tuning interface for the users. This module has a size of 8 HP, making it smaller than the other two. In terms of the module's signal path, the user first plugs in a sound source, which can either be generated from oscillators or played on an acoustic instrument. The nine parameters shown underneath the knobs were chosen according to Haedam's requirement. After the sound is pre-tuned, it is ready to pass down to the algorithm for timbre transformation.



(a) Frankenstein Instrument module overlay design.



(b) Frankenstein Verb module Overlay Design.

Figure 81: The selected motor model and its dimension

6.6 Frankenstein Verb Overlay Design

The Frankenstein Verb module can work independently or alongside Yuanjun's algorithm. The module has a size of 10 HP. It consists of six small knobs, one for each of the parameters, and one big knob to select the environment. Users can plug in an oscillator or an acoustic sound source through the IN jack and monitor the sound as they tune the sound on the hardware interface by plugging in a speaker through the OUT jack. A special feature of this module is the pink buttons on the left, which are used to select the vectors for the corresponding knobs. The lines on the overlay in figure 81b indicate which button corresponds to which knob. Each button cycles through a bank of 4 vectors, and the vector selected is indicated by the LED light underneath the knobs. When using Yuanjun's parameter finding algorithm, the LED light guidance ring from each knob will highlight the position to which the knobs need to be turned to in order to get the desired sound effect.

6.7 Interactions Among the Hardware Modules

As mentioned in the introduction, an advantage of modular synthesisers is that different modules can be patched together. For users who own all three hardware modules, the oscillator from the VCO/VCF/EG module can act as the sole sound source for all patching. For example, users can patch the oscillator with the Frankenstein Instrument module for pre-tune, and transfer this to the DDSP software for timbre transformation. Users can also patch the oscillator with the Frankenstein Verb module to get reverberation effects, and then pass the signal back to the VCO/VCF/EG module for the filters and envelop generator.82 For users that own other modules, they can use the Frankenstein Verb and Frankenstein Instrument module to create effects for oscillator-generated sounds or recorded acoustic sounds.

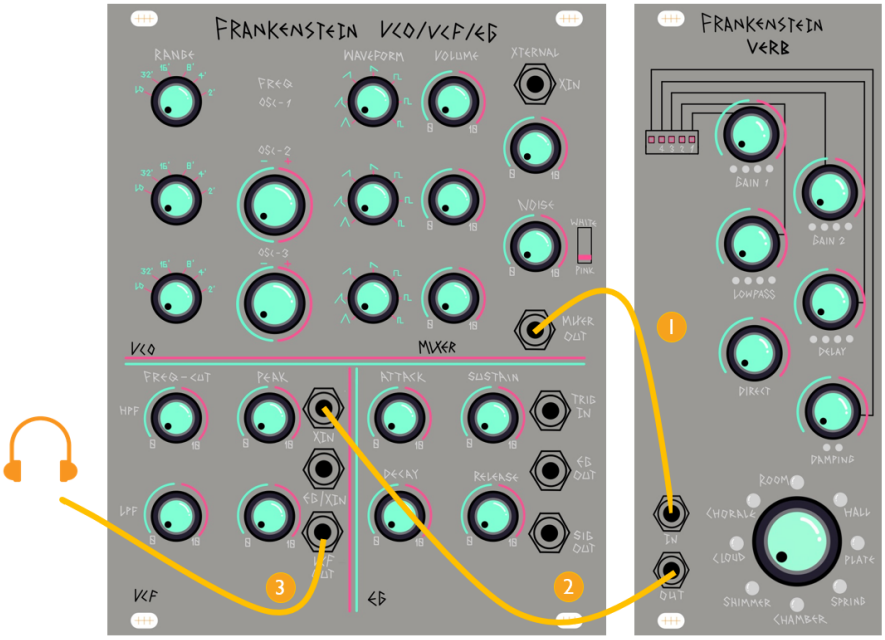


Figure 82: A patching suggestion. The oscillator generates the sound signal, which is transferred to the Verb module, and then to the VCF.

7 Software Back-End - Siheng Feng, Haedam Oh

7.1 Introduction - Siheng Feng

The software Back-end is in support of Frankenstein (as shown earlier in figure 3), besides the previously introduced algorithms and software interfaces. Under the scope of this project, four main areas are covered: Hardware Programming, the VST design, the database that holds all audio data to be processed, and cloud-side applications that offload the task that requires more computation power.

7.2 Programmable Hardware - Siheng Feng

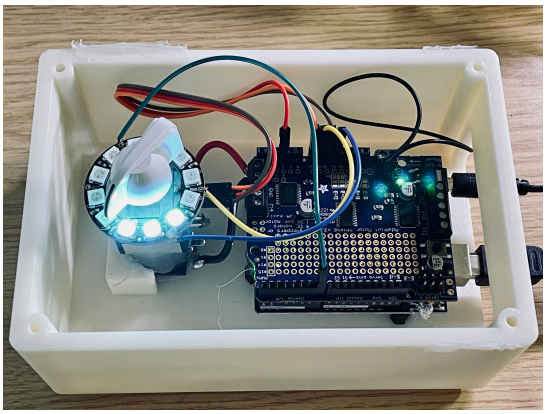
7.2.1 Programmable Hardware - Overview

One of the key components of Frankenstein is its novel intelligent user guidance system, including automatic parameter setting on the physical synthesiser knob and an LED ring for user indication. This system works alongside the circuitry of the existing circuit components of a traditional synthesizer.

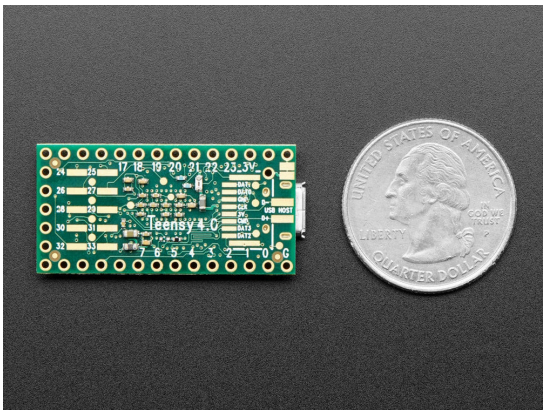
We summarised the task required to be done as follows:

1. Receive position information from upstream applications, Param_Match and Reverberation
2. Perform user guidance, including knob automatic rotation and LED indication on synthesiser.
3. Send the current knob position to the upstream applications. This is required for all three modules.

The common solution for hardware programming is to use a micro-controller(MC) for instance, Arduino or Teensy, to handle the request from Frankenstein's VST(in section 7.3) on the user's device and perform actions on the hardware connected to the MC. Once receive the request, the MC controls the motor's rotation and LED ring. We present a demo based on Arduino UNO as shown in figure 83(a).



(a) Motor and LED guidance Demo



(b) Teensy Micro Controller[138]

Figure 83: Left: The demo we used to present the LED/motorised knob concept. Right: The size of the Micro Controller can easily fit into the modular synthesiser.

7.2.2 Hardware Programming - Micro Controllers

There are several available choices as listed in the table11. Frankenstein's MC is not required to handle complex audio signals, but instead, we want to carry out the control of multiple motors and LEDs. The Input-Output I/O capability is what we are focusing on. We need control over up to 21 knobs per

Frankenstein's module. Thus, we prefer Teensy[138] and 2560[139]. AutoPID is the Library used for motor control which requires enough computation power. In our testing, Arduino 2560 is capable of handling up to 4 motors' PID control. We also take size and cost considerations into the design. Depending on or hardware design objective in section 6, a smaller form factor is preferred. Based on the arguments above, **Teensy** MC, as shown in figure.83, is selected to be our hardware microcontroller.

Board	CPU Speed	I/O (D (PWM), A)	I2C (Buses)	Size (mm)	Price[140]
Teensy 4.0[138]	600 MHz	D:40 (31), A:14	Yes (2)	35.3 x 17.8	£23.30
Arduino Uno[139]	16 MHz	D:14 (6), A:6	Yes (1)	68.6 x 53.4	£14.99
Arduino 2560[141]	16 MHz	D:54 (15), A:16	Yes (1)	101.6 x 53.3	£18.99
Raspberry Pi 4[142]	1.5 GHz	D:28 (2), A:0	Yes (configurable)	85.6 x 56.5	£52

Table 11: Comparison of Micro-Controller Boards, CPU speed is in MHz (MegaHertz) or GHz (GigaHertz), I/O D is the available Digital ports, (PWM) is the number of pins supports Pulse width modulation, I/O A is the number of Analogue ports, I2C(buses) is the number of buses of Inter-Integrated Circuit

7.2.3 Programmable Hardware - Communication

We need to pass the current position of the knob to Frankenstein's VST at any time to monitor user interaction. This is done by performing communication via the serial ports of Teensy and the user's device. This is a bi-directional process. We can send target position for user guidance from Parameter Matching Module and Reverb Modules to the Micro-Controller[15] and receive user's interactions on hardware panel from the Micro-Controller. All three modules require the latter information.

7.2.4 Programmable Hardware - Actuators Control

The overview of the control and communication of the Micro-Controller is summarised in figure 84. Three main components are involved in this design: The Teensy MC, the Inter-Integrated Circuit (I2C) motor shield between the MC and the motor, and the AutoPID Library to calculate the power delivered to the motor. The LED control is relatively simpler, we light up the LED bulbs for a given target position. Conversely, motor control involves more details in order to perform control up to 21 motors[16]. The MC first send the instruction to a medium, for instance, a motor shield, to simplify the control process. This medium is called an I2C device. We chose to use two of the Adafruit 16-Channel PWM Driver[143] in order to drive the 21 motors we have. The position is recorded from the potentiometer of the motor and

[15] Intelligent user guidance is required for Parameter Matching Module3.1.2 and Reverberation module, but not for Instrument module

[16] The required motor control is for the VCO-VCF-EG hardware module in section 6.4

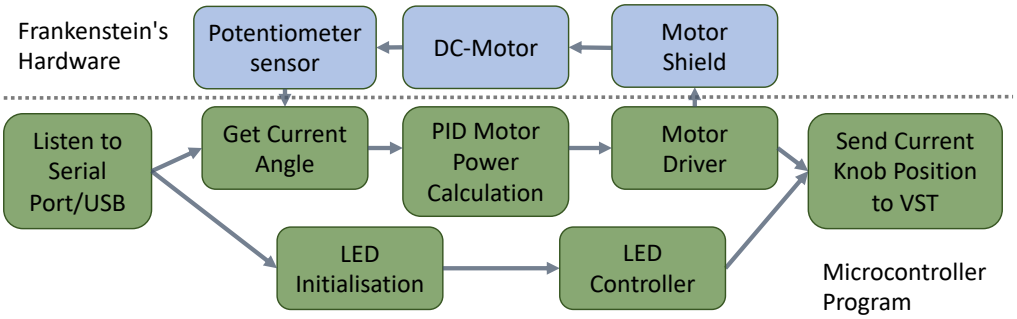


Figure 84: Micro Controller (MC) system overview

sent back to the MC. The control loop is shown in the figure.84 is repeated until the knob of the synthesiser (physically connected to the motor) has rotated to the desired position. This entire process takes less than 7 seconds for the entire process, thus satisfying the 10-second performance objective1.3.

7.3 VST Plug-in Design: JUCE - Haedam Oh

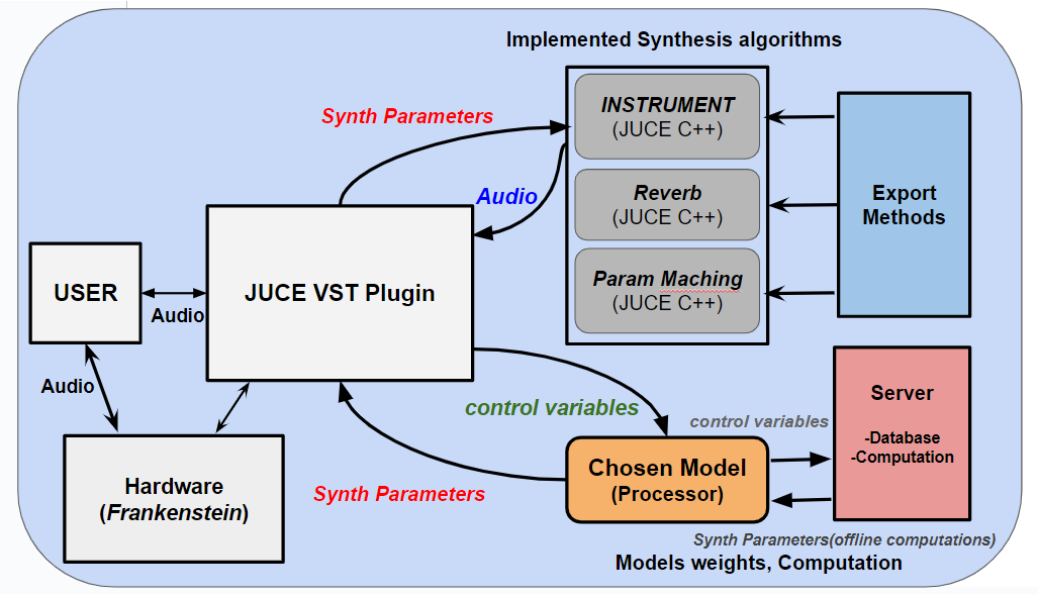


Figure 85: JUCE Framework

7.3.1 JUCE C++ Framework design

The JUCE framework is a popular C++ framework used for developing cross-platform audio applications, including VST plugins. JUCE provides a comprehensive set of tools, libraries, and APIs. Figure85 shows the overall distributed system. Following Figure85, we first designed main **JUCE VST Plugin**, **Chosen Model(Processor)** and **Synthesis algorithms** in this section. Implementation details are shown as UML

diagram86 with reference to JUCE C++ framework[144].

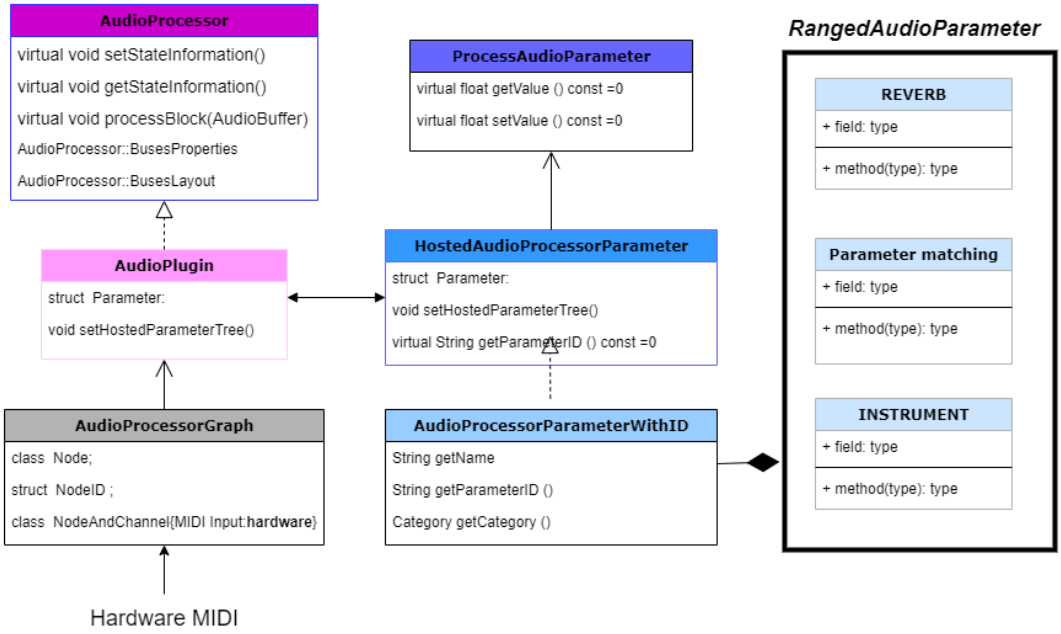


Figure 86: **Designed JUCE UML**: 1. `AudioProcessor` provides necessary controls and tools for VST plugins as a main library. Add `processBlock()` for audio processing and `getStateInformation()` `setStateInformation()` for storing and retrieving audio files. 2. Implement input-output channels, Buses: `AudioProcessor` can take multiple channels. `BusesProperties`, `BusesLayout` provides configuration of channels: quadraphonic, stereo or mono (hardware) MIDI input. 3. Our 3 modules from *Export* can be called through `ProcessorAudioParameterswithID`. 4. `AudioProcessorGraph`: allows us to connect several AudioProcessor objects together as nodes in a graph and playback the result of the combined processing. Finally, converting Plugin to GUI App by calling `AudioProcessorGraph Main.cpp`

7.3.2 Export

We need to export our models in JUCE C++ framework beforehand. Figure85 shows **Export** of modules separately. *Instrument* built upon python Tensorflow2, *Param Matching* on python pytorch framework, and *Reverb* with Matlab. Conversion to JUCE C++ must be done thoroughly. Even though we could not actually export models to JUCE under time constraints, we will suggest some implementations that can be done as future work. We suggested each model be exported in their own way.

Using ***INSTRUMENT(DDSP)*** as an example: The easiest way could be using Tensorflow API for C++. *Ganis*[145] showed the implementation of real-time DDSP Plug-in by 1) translating DDSP library to Matlab. 2) Use Matlab-synth tools to convert Matlab to C++ codes that are suitable for JUCE framework. Tensorflow API for C++ is based on *graph mode*, whereas our trained models used *eager execution mode*. Conversion to *graph mode* has limited several algorithms, such as encoder controllability in JUCE framework. PYTORCH C++ API[146] is another reliable tool to convert Resnet or Vggnet-based models.

7.4 Server - Haedam Oh

- **Server-side code**: We have decided to use C++ as server-side programming language considering the following factors: Compatibility with the VST plugin, Efficiency and Speed. First, compatibility with VST Plugin(JUCE) is important to consider. JUCE supports C++, so choosing a language that can interface with C++ would be beneficial. This is not only solving dependency issues between software but also keeping updates on the newest version. Therefore, it can provide a robust platform for users. Next, considering efficiency and speed, C++ is a relatively fast, compiled language that provides low-level control over hardware resources, which can be optimized for efficient code execution. Even though users may not need a speedy server as they will spend more on synthesis music, quick access to the server is necessary for users' convenience. However, there are some disadvantages to using C++ as a server-side language regarding concurrency and scalability. For example, as the number of customers increases, we need to implement concurrent and scalable server applications in C++. This requires more manual effort and expertise in managing threads, locks, and synchronization. The simple solution could be choosing a scalable database(NoSQL) and storage that can handle concurrent access and high request rates. As a result, we proposed C++ for the initial server-side language but change to Java or Go for better concurrency and scalability.

- **Operating system**: JUCE provides cross-platform support Windows, macOS, and Linux. Thus, we need to decide in which environments computations on models should be done. Linux is the most common and best option. Linux provides efficient resource management, making it well-suited for computationally intensive tasks. Also, it enables better utilization of system resources, such as CPU, GPU, and memory. For example, package managers, such as apt (used in Ubuntu) provide easy access and simplifies the installation and management of ML libraries, frameworks(TensorFlow, PyTorch), and dependencies.

- **Hardware requirements and Maintainace**: We specified hardware for training our models and checked the dependencies between hardware. Intel Xeon Silver 4216 Processor CPU: a server-grade CPU with a minimum of 32 cores, high clock speed required. Kingston 64GB DDR4 ECC Registered Memory(RAM): at least 32GB is required but preferably 64GB(to prevent explosive loading). Samsung 870 QVO 2TB SATA SSD: many 2TB SSD because multiple SSDs can potentially offer higher read and write speeds than a single SSD and are cost-effective. Considering short-

term profits and the size of this project, renting a server can significantly reduce hardware and maintenance costs. Available cloud service providers are Azure, Amazon(AWS), GCP(Google) and IBM. Prioritising serverless computing and easy management, we selected Google Cloud Platform(GCP) as a service provider. It can offer the hardware requirements above according to the documents[147] and provide computations to users.

7.5 Application Programming Interface(API) - Haedam Oh

7.5.1 Introduction to API

A client sends a request, which consists of the endpoint(a resource path), the method(GET, POST, PUT, PATCH, or DELETE), the headers, and the body(JSON). Then the request is processed to the web server via the Uniform Resource Identifier(URI). After calling API, the server passes a response to the API.

7.5.2 API Design1 (users-server)

- **Type**: RESTful APIs are popular for simplicity and ease of use. For example, it provides 4 standard HTTP methods (GET, POST, PUT, DELETE) and reasonably good scalability for handling large numbers of concurrent requests. However, significant disadvantages of real-time updates and over-fetching of data. RESTful API is quite stable and cost-efficient considering the number of customers and current market(Netflix and Github use a RESTful API, while Facebook uses GraphQL).
- **API keys**: Once registered in our server, the API provider will generate an API key, which controls access to the API's resources. Depending on the API provider's configuration, the key may grant different levels of access or permissions, allowing access to specific endpoints or performing certain actions within the API. Store them securely, such as in environment variables or secure configuration files.
- **HTTP Methods and Endpoint**: URL that consists of a base URL combined with a path that represents the specific resource. Example endpoint: https://api.frankenstein.org/users/collections/*. It is common to list the method next to the endpoint. GET: Retrieves a resource POST: Creates a resource. PUT: Updates or creates within an existing resource. DELETE: Removes the resource
- **Response generation and processing**: Receive the response from the server as JSON file and process them in codes.

7.5.3 API Design2 (JUCE Plugin - Modules)

- **Parameter Control**: `juce::AudioProcessorParameter` Expose getter and setter methods to allow the model to access and modify these parameters.
- **Model Integration**: Define an interface or class that represents the loaded model and its associated functionality. Design methods within the model interface to receive audio data for processing and return the processed audio. Implement the model using the appropriate machine learning framework (e.g., TensorFlow, PyTorch) and algorithms for audio processing. Establish a communication mechanism between the plugin and the model, such as function calls or callbacks

7.6 Database - Haedam Oh

Our software such as *Instrument*,and*Reverb* have a back-end database to store, update, or remove music created with different modules(See Figure87). This enables efficient retrieval and organised structure of data. We decided to choose NoSQL(MongoDB[148]) database for the following reasons. Details are shown in Figure88.

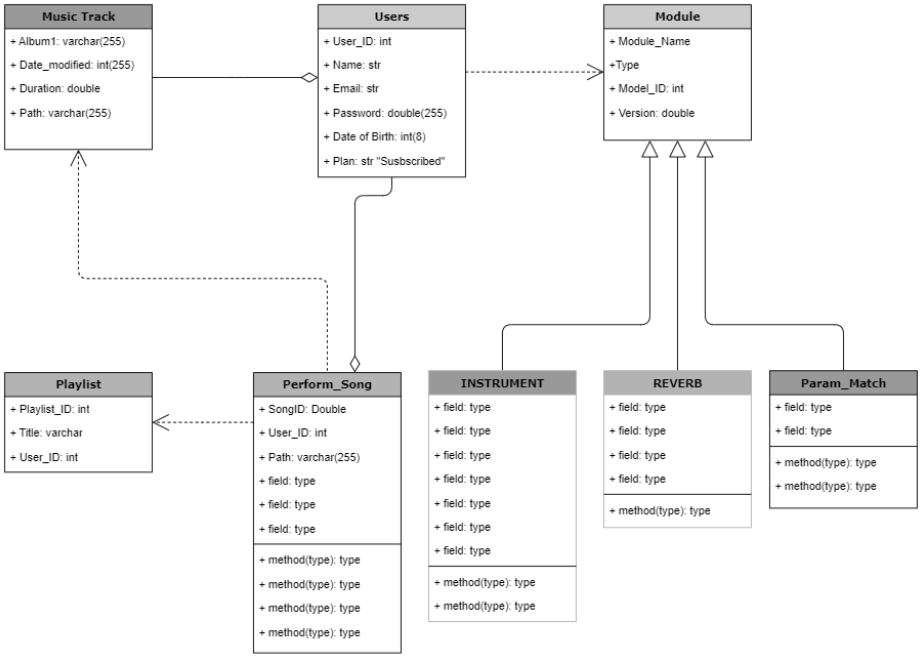


Figure 87: Database Schema

1. **Handling Unstructured Data**: Most VST-plugin working real-time does not directly use non-relational NoSQL databases. However, we synthesise in not only real-time but also offline. NoSQL is suitable for unstructured data and allows for flexible and adaptable data structures. For example, INSTRU-

MENT Module stores, pre-trained instruments, but users can also add their own instruments during synthesis. Flexibility in data structure benefits users to use a variety of models.

2. **Scalability&Flexibility**: NoSQL can scale horizontally, meaning they can handle large amounts of data which was not able to handle by the Hardware itself. Also, it can store different types of data in the same collection or table without the need for a predefined schema. For example, audio can be written in .WAV, .MP3, .MID etc. It allows multiple formats of sound.

3. **Performance&Falut-tolerance**: NoSQL databases are optimized for high read and write throughput, which is very crucial regarding synthesis time. Distributing databases across multiple servers provides high availability and fault tolerance.

```
{
    "userID": 123,
    "userName": "Haedam1234",
    "File": [
    {
        "entryID": 15255,
        "pieceID": 8182,
        "title" : "synthesised violin ",
        "format": ".wav",
        "length": "3:15",
        "module": [
                {
                "INSTRUMENT": [
                {
                    "Type": "Violin",
                    "Timbre_transfer": "None",
                    "Pitch_shift": "one octave",
                    "Loudness_shift": 10,
                    "Reverb&Noise": 0
                    ....              },
                ],

                "Reverb": [
                {   "Delay" : "None",
                    "Reverb_factor" : 10,
                    .....
                        },
                ],

                "Param_maching": [
                {   "Type": "Param_maching",
                    "Model" : "CNN2" ,
                    .....
                      },
                ],

                }
            ]
        }
    ]
}
```

(a) Music Json File

```
{
    "Module": {
        {
        "Module_ID": 2080809,
        "Name_Name": "INSTRUMENT",
        "Models": {
            "Guitar": 00103380,
            "Piano": 09052023,
            "Violin": 0090912,
            "Flute": 0100204,
            "Drum" : 0400203,
            "Danso" : 1000304
            }
        },

        {
        "Module_ID": 2080901,
        "Name_Name": "Reverb",
        "Models": {
            "d_multiplier": 0000001,
            "b_multiplier": 0000002,
            "c_multiplier": 0000003,
            "G_gain": 0000010,
            "f_cutoff": 0000100,
            "M_delay": 00001000
            }
        },

        {
        "Module_ID": 2081001,
        "Name_Name": "Param_maching",
        "Models": {
            "Param_Maching_CNN": 0000001
            ....
        }
        },
    ......
    }
}
```

(b) Module Json File

Figure 88: Templete JSON documents for Database schema

8 Business - Haedam Oh, Yuanjun Wan, Siheng Feng

8.1 Project Management - Haedam Oh

In this section, we showed project management with guidelines of Prince 2 approach[149].

1) Initialising concepts & objectives, date: 21/10/2022-17/12/2022

1. **Feasibility study**: we had to verify whether our product is technically and financially feasible in practice. Research on both existing software and hardware has been done during this period. Macroscopically, we showed the feasibility of integrating both hardware and software as a synthesiser product. Next, 3 separate modules for software need to be feasible individually in terms of inter-dependency, the cost to run. Limited capability of personal CPU and GPU with low memory caused technical feasibility issues on running. We modify our design and models accordingly.
2. **Specification outline**: a very brief outline was considered during this period; hardware is a modular synthesiser and software should support hardware with more advanced tools(e.g translate one sound to another, add audio effects)
3. **Team management**: Zhaoqiu on hardware others Siheng, Haedam and Yuanjun worked on software development considering individual skills in engineering and their own interests.

2) Project Planning, date 17/12/2022-27/01/2023

Individual roles and responsibilities were more specified: Gantt chart89 shows more detailed individual specifications. To keep on track, individual work breakdowns were recorded in their own log book through regular weekly meetings. Zhaoqiu started on hardware circuit design implementation, Siheng started research on parameter matching. Haedam researched DDSP and Wan researched AI models for audio effects.

3) Implementation, date 27/01/2023-10/04/2023

Implementations of individual parts were done during this period. Incorporate regular testing and quality checks throughout the project. This is critical for ensuring the reliability and robustness of models. Continuous adaptation and optimisations of models have been done to the latest updated version.

4) Monitoring and control, date 05/03/2023-10/05/2023

Training and evaluations of each module took a month longer than we expected. Unexpected errors and continuous modifications of codes have delayed evaluations. Therefore, we decided not to evaluate

software implementation designs and module interaction parts due to this delay.

5) Closure

MVP and target customers of the product were reviewed again. Check 3 software modules fit together with the hardware. Check any incompatibility inside the software framework and between hardware and software. Regular reviews of documents to see final product matches with recorded specifications.

8.1.1 Gantt Chart - Yuanjun Wan

The designing process of Frankenstein can be generalised to figure 89 from the research stage to the final product realisation stage.

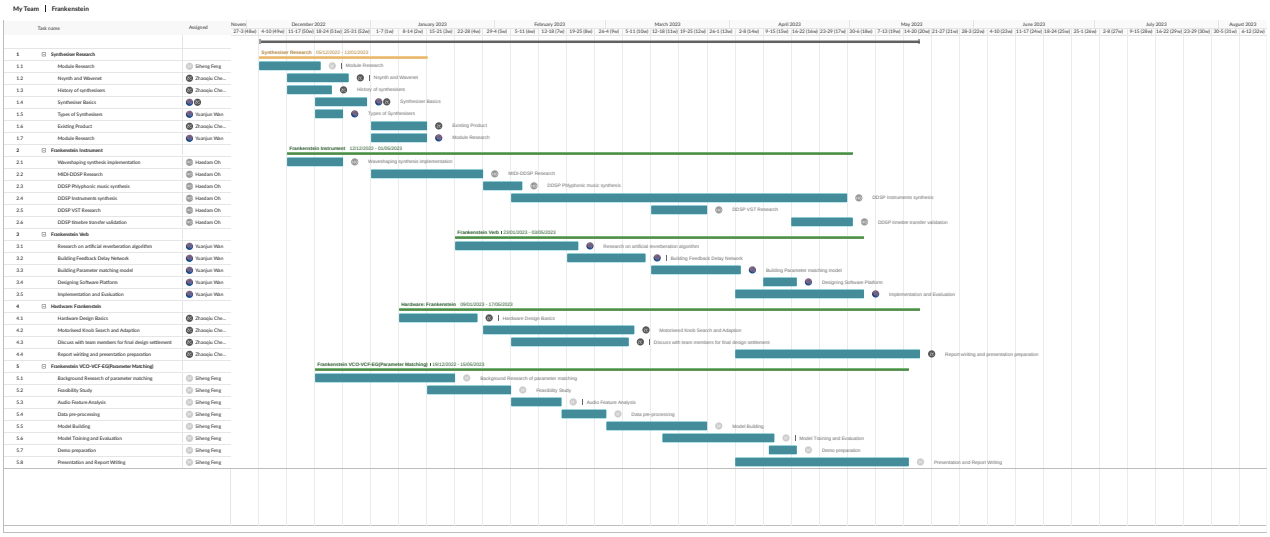


Figure 89: Gantt Chart of Design Process of Frankenstein

8.2 Project Finance - Yuanjun Wan

This section will provide a comprehensive examination of the financial aspects related to the product. We will delve into the cost structure, turnover from sales and potential external sources of funding for the product. We will show that our product is financially sustainable after all the examinations.

8.2.1 Cost Estimate

The cost associated with producing Frankenstein primarily comprises both direct and indirect expenses. Direct costs are related to the manufacturing of the synthesiser hardware and the provision of software services. On the other hand, indirect costs, while not directly linked to the production of Frankenstein, are equally essential as they support the business's overall operations.

Direct Costs - Hardware Manufacturing

The primary expenditure in synthesizer production arises from hardware costs. Table 12and table 13 present a list of the main mechanical and electronic hardware components necessary for constructing ”Frankenstein” and the approximated cost of each component from market research. The total cost of producing one ”Frankenstein” is subsequently estimated by considering both the cost of each component and the number of components required for each module(figure 90).

It is worth noting that we have no access to the production prices of components in the factories. Hence, we estimated the manufacturing cost with **accessible prices online discounted by retail profit margin of electronics**. The profit margin of general electronics is around 30%[150], thus the cost of production is the price we found online discounted by this margin.

Mechanical Component	Knob	Modular Synth Enclosure	Hardware Surface Overlay
Price per component	£0.5	£20	£8

Table 12: Mechanical Components of Frankenstein and Prices[151]

Electronic Component	Price per component
Potentiometer	£2.27[151]
Micro Controller: Teensy 4.0(7.2.2)	£20 [152]
Motor Driver: Adafruit 16-Channel PWM(7.2.4)	£10 [153]
Pinted Circuit Board	£15 [154]
LED Ring	£6 [151]
Motors	£1.67[151]

Table 13: Electronic Components of Frankenstein and Prices

Module	Main Component Requirement Details	Total Cost	Discounted to production price
Frankenstein VCO-VCF-EG	Potentiometer*21, Micro Controller*1, Motor Driver*2, Motor*21, PCB*1, Knob*21, Overlay*1	£156.24	£109.37
Frankenstein Instrument	Potentiometer*9, Micro Controller*1, PCB*1, Knob*9, Overlay*1	£67.93	£47.55
Frankenstein Verb	Potentiometer*7, Micro Controller*1, LED Ring*6, Knob*7, Overlay*1, PCB*1	£98.39	£68.88
All Modules	All modules share one Modular Synth Enclosure	£20	£14
All Modules	Total cost of producing one Frankenstein	NA	£239.8

Figure 90: Hardware costs of manufacturing of each module and Frankenstein in total

From figure 90, we conclude that the estimated cost to produce one Frankenstein hardware is around

£240.

Direct Costs - Software Services

The software cost mainly arises in renting a server(section 7.4). Based on the estimation by Google Cloud Pricing Calculator[17], the estimated cost is around £620 per month. We tried to control the cost by coming up with an intelligent allocation of the server running time instead of keeping it in service for 24 hours continuously.

Indirect Costs

Indirect costs supporting the business can be classified into multiple areas. In the case of our product, the main indirect costs fall into the following four categories: administration expenses, research and development, marketing and sales, and logistics. Table 14 provides an estimate of the annual indirect costs associated with these categories. **All the indirect costs are obtained from relevant sources or estimated based on research of actual market data.**

Estimating indirect costs can be complex, particularly without specific details regarding the business context. For our project, we have considered our intended location: Oxford, UK and our strategy for the initial stage of the business. We aim to maintain a small, efficient business organisation to mitigate indirect costs. Our team, currently composed of four members, will be expanded modestly. We will hire one experienced engineer to collaborate with us on research and development and one administrative staff member to assist with organisational tasks. This approach allows us to balance the need for expertise and administrative support with the desire to minimise our initial operating costs.

The total indirect costs based on main cost components are estimated to be £109,120(table 14) annually. However, we should also consider miscellaneous costs incurred in the ongoing business process. As a result, we will give the budget of indirect costs a buffer and aim to control the indirect cost within £115,000.

8.2.2 Pricing Strategy of Frankenstein

In determining the pricing strategy of Frankenstein, we adopt a comprehensive approach that extends beyond mere cost considerations. We also conduct a market analysis to understand the price trend of existing products. Beyond that, we also look into our target customers' preferences in price.

[17] https://cloud.google.com/products/calculator?hl=ko

Category	Details	Annual Costs
Administration	Renting a medium office for 1-10 people in Oxford Business Park	£$2000/month \times 12\ months$[155]
	Utilities and Office Supplies	£$900/person \times 6\ people$[156]
	Salary of administrative staff	£24000[157]
Marketing and Sales	Advertisement on music technology website 'Sound on Sound'[158] with a rectangle banner	£670 [158]
Research and Development	Salary of experienced engineer	£54000[159]
Logistics	Renting a storage place for manufactured synthesisers at Cherwell Business Village[160]	£$7.00/^2ft \times 150^2ft$ [160]
Total	Total indirect costs incurred per annum based on main components listed above	£109120

Table 14: Main components of indirect costs obtained from relevant sources or estimated from market average

Market Analysis for Pricing Strategy

Synthesizers typically come with a high price tag, primarily due to the manufacturing processes involved in creating the hardware components[161]. Considering the absence of direct competitors to Frankenstein in the market, we will investigate the price of three modules separately to better understand the pricing trends of the whole product. We will need to analyse modules of approximately the same size in the market to ensure the accuracy of price analysis.

Analogue Synthesiser: The price of a 20HP(6.4.4) analogue synthesiser mostly ranges from £200 to £500[162]. The variations depend on their complexities and brands.

Reverb Module: The price of a reverb with the size of about 10HP(6.6) in the market costs about £150 on average[162].

Instrument Synthesis: The Frankenstein Instrument is a relatively novel product. We have not found a similar hardware synthesiser in the market. However, we have researched on the price of several instrument synthesis software which cost around £120[163].

Target customer Analysis for Pricing Strategy

We should also take into account the purchasing appetite of our customers. It is recommended that 'Don't pay anything over £200 if you are a total novice on synthesisers[164]'. Beginners typically prefer cost-friendly products for their initial synthesiser purchase, and then consider more expensive options as they become more experienced. This mindset should be a consideration when we price our product.

Purchasing Plan

Our pricing strategy has been determined after thorough consideration of production cost, market trends, and customer preferences. The purchasing plan can be found in the provided figure 91. We offer both the software and hardware modules together in one package, giving users the choice between a basic or premium package. Notably, the cost of our Frankenstein set is less expensive compared to purchasing three individual modules in the market, while still covering production costs. Furthermore, the total price includes all necessary software support programs, enhancing its appeal to our target customers.

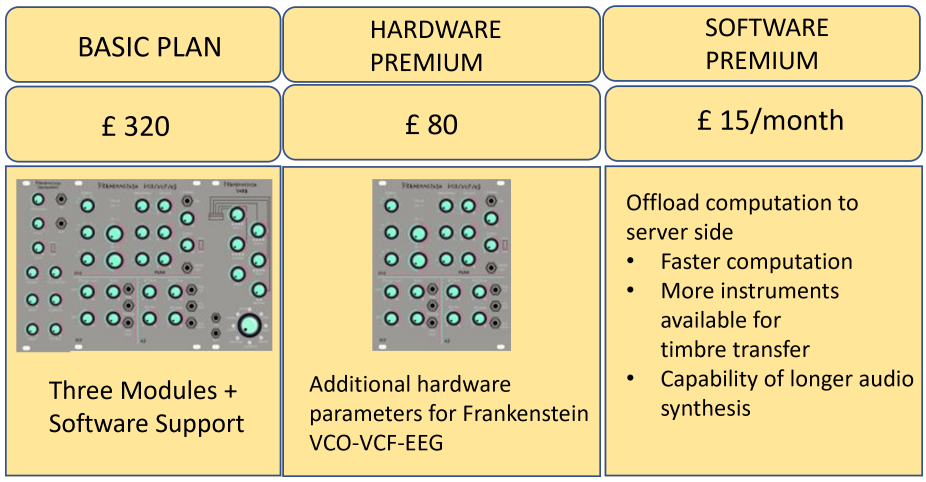


Figure 91: Purchasing Plan of Frankenstein

8.2.3 Seed Stage Funding

In the early stage of our business, acquiring venture capital funding is a crucial step towards Frankenstein's development. Venture capitalists are typically attracted to products with a competitive edge in the market[165]. Frankenstein caters to synthesizer beginners and has limited direct competition, aligning well with the interests of venture capitalists. As a start-up, our initial funding round will be seed funding[166]. We will primarily engage with early-stage venture capital firms with a history of investment and interest in the music technology sector. We have identified Balderton Capital[18] as a potential investor. As one of Europe's leading venture capitalists, they have previously invested in music industry businesses such as Kobalt[19]. According to SeedLegals[167], UK startups typically secure between £500,000 and £1.5 million during their seed round, offering between 10% and 20% equity. We aim to raise £500,000 for a 20% equity stake in Frankenstein. Given that our fundraising goal is below the

[18] https://www.balderton.com/
[19] https://www.kobaltmusic.com/

market average and our equity offer is relatively generous, we are optimistic about successfully securing initial funding.

8.2.4 Financial Outlook

The financial outlook for our AI-powered modular synthesiser Frankenstein appears promising. Our comprehensive pricing strategy, formed through careful consideration from different angles, has positioned us to achieve a balance between accessibility for our customers and profitability of our business, which sustains our financial competitiveness in the market. We have also obtained funding that would cover our estimated costs and provide a safety net for unforeseen expenses. Consequently, the project is well-positioned to become financially sustainable in the long term.

8.3 Risk Analysis - Siheng Feng

Risks	Descriptions	Probability	Severity	Actions required
Quality issues for the hardware synthesiser	The defects in circuit board manufacture will greatly hurt the quality of our output sound.	Median	High	Apply rigorous Quality Control at manufacture stage, which might increase our overall cost.
Communications problem between sub teams	Our project is spited into four different sub tasks which are carried out by 4 different teams. Miscommunication leads to compatibility issues in the final product.	High	High	Enforce scheduled group meetings to communicate progress and problems found. A clear hierarchy of teams' roles and responsibility.
Privacy issue with machine learning models	Our product requires input of audio which may contains users' sensitive information. Fail to gain public trust and meet regulations in our product lead to loss of customer and potential lawsuits.	High	Very High	Follow the laws and regulations in privacy and enforce strict protocol for user data collection. Ensure that user's data is not used for training purpose unless user agree.
Software compatibility issues	Our machine learning models may run on different environments than the software we are developing. Compatibility issue could lead to limited or delayed functionality deployment.	High	Median	Unify a developing standard across each sub teams. Clear definition on data interfaces between machine learning models and VST software.
Delays due to technical difficulty	We are developing a product that involves sophisticated machine learning models. We cannot assume the performance will meet our need. More time might be required to improve the algorithm.	High	Median	Prepare for a Plan-B, an easier developed but not as good algorithm. We can ensure the product will be ready with Plan-B at launch time and prepare for an update once the Plan-A algorithm is ready.

Figure 92: Frankenstein's Risk analysis. Remarks: The colour code for the Risk is the same as the severity and probability. Follows a similar approach from Oxford B2 Risk and Security.

For a project that involves both hardware and machine learning models, a risk analysis is crucial before we head into development. We selected some risks that we believe are most representative of the project and listed them in figure 92. We acknowledged the fact that a more comprehensive study could be done, but the current risk analysis can help us avoid many problems.

8.4 Privacy for Data Collection - Siheng Feng

As previously discussed in section 8.3, user privacy is our greatest concern. In the United Kingdom(UK), several regulations apply, including the "General Data Protection Regulation"[20] (GDPR), Data Protection Act(2018)[21] and many others. But for data collection for AI model training, there have been a lot of controversies, also related to the recent ban of Chat-GPT in certain countries due to privacy issues. Arguments are mainly around whether the model still contains user data; is it already generalised or is it still possible to recover those? These are areas quickly evolving with new regulations still yet to be in place. To adapt to the potential effect, we design our privacy protocol based on the regulations, with some ideas including:

1. **Users' Consent**: Under GDPR, we can only collect user data that users explicitly consent to with the purpose of these data used explicitly stated. To avoid potential legal problems, we must state clearly how the data is used. We must first gain the user's approval to use data in each of the training or non-training tasks. If the data is used to train a larger model and we can not remove personal data from it, we must first gain user's consent on a "freely given" basis that informs the users: once they make this decision, it is irreversible. This is highlighted separately so user doesn't accidentally choose this option.

2. **Security and Right of Forget**: The data provided by users can only be accessed by the people user previously consented to. We must enforce strict data access protocol and apply encryption to the database. If an additional third party requires the data, the user must be informed. User also has full right to access and delete any audio data they have provided to us if they are not on a "freely given" basis, e.g., for Parameter Matching. We also delete the data after providing the matched parameters unless the user chooses to keep it for further use.

3. **Marketing**: We assure our customer that the data provided is never be used in marketing service, as the input audio contains a personal preference for music. We also do not profile our users for

[20] https://gdpr-info.eu/
[21] https://www.gov.uk/data-protection

using our service. The data we collect is only used for purposes that user consent to.

We keep monitoring the recent change in regulations and be ready for potential changes to this area to protect our customers' privacy and data security.

9 Summary - Siheng Feng, Yuanjun Wan

In light of the project's initial objective, we attempt to balance the intricacy and flexibility of a traditional modular synthesiser. In the 'Parameter Matching' section, we achieve the objectives that a parameter preset can be matched from an input audio clip fast and accurately. For the 'Instruments' module, both synthesis and controllability on various instruments have been provided to users with timbre transfer options. The 'Reverberation' module achieved efficient production of environmental effects of good quality. We cover the use cases for both live performances: real-time timbre transfer; and at-home composition for a beginner. Combining with Frankenstein's novel intelligent user guidance system, a more straightforward and intuitive interaction can be made between beginners and the inherently complex modular synthesiser. By providing Frankenstein, we hope to attract more music lovers to join the synthesiser community and begin their voyage to the ocean of sound effects.

While we are currently in the prototyping stage of our product, we are confident that our development is progressing on the right path. Our decision-making process considers more than just technical supremacy; we prioritise the practical needs and preferences of our target customers. This approach will lead to a user-centric product that will build a solid reputation within the industry. Our objective is to demonstrate to potential investors the promise our product holds, substantiated by a profitable balance sheet. In doing so, we hope to secure additional funding rounds, enabling us to hire more talent and further develop our product.

We recognise our current limitations, primarily due to constrained time and resources within the scope of the project. However, we are aware of potential avenues for improvement and are committed to the continued development of our product. Moving forward, we plan to collaborate with experienced audio, electronic, and information engineers to enhance our technical capabilities. Additionally, we aim to conduct comprehensive market research to gain deeper insights into our target customer group. These steps will enable Frankenstein to mature both technologically and in terms of user experience.

References

[1] Joe Albano. Ask audio. modular synth patching basics (with vcv): Classic synth bass. Available at https://tinyurl.com/4cjcv9tp(2021/03).

[2] Arturia. Arturia dx7. Available at https://www.arturia.com/products/software-instruments/dx7-v/overview. Accessed: 2023-05-10.

[3] Technavio. Music synthesizers market by distribution channel, type, and geography - forecast and analysis 2022-2026. Available at https://www.technavio.com/report/music-synthesizers-market-industry-analysis (2022/08).

[4] Technavio. Music synthesizers market size worth $78.11 million by 2023 - technavio. Available at https://www.businesswire.com/news/home/20190709005399/en/Music-Synthesizers-Market-Size-Worth-78.11-Million-by-2023---Technavio (2019/07).

[5] Steam. Steam hardware and software survey: April 2023. Available at https://tinyurl.com/2p88wud9. Accessed: 2023-05-10.

[6] Dell. Dell minimum ram laptop. Available at https://www.dell.com/en-uk/shop/laptops-2-in-1-pcs/sr/laptops/4-gb-or-less?appliedRefinements=37704. Accessed: 2023-05-10.

[7] Robert B. Miller. Response time in man-computer conversational transactions. In *AFIPS '68 (Fall, part I)*, 1899.

[8] Jansen Troy. Modular synthesis basics (how it works and how to dive in. Available at https://producerhive.com/ask-the-hive/modular-synthesis-basics (2022/08).

[9] Kim Bjørn and Chris Meyer. *Patch & Tweak. exploring modular synthesis*. Bjooks Media, København Denmark, 3 edition, 2018, pp.14–21.

[10] Wikipedia. Envelope (music). Available at https://en.wikipedia.org/wiki/Envelope_(music) (2023/03).

[11] Technavio. Adsr: The best kept secret of pro music producers. Available at https://www.musicianonamission.com/adsr/ (2020/05).

[12] Giovanni de Poli. A tutorial on digital sound synthesis techniques. *Computer Music Journal*, 7(4):8–26, 1983.

[13] Eduardo Reck Miranda. *Computer sound design: Synthesis techniques and programming*. Focal Press, Oxford, United Kingdom, 2 edition, 2002, pp.50–51, 71–72.

[14] Practical Music Production. How do synthesizers work? Available at https://www.practical-music-production.com/how-do-synthesizers-work/. Accessed: 2023-05-10.

[15] Reddit. Where to learn to use synthesizers? Available at https://www.reddit.com/r/synthesizers/comments/12o1jo6/where_to_learn_to_use_synthesizers/. Accessed: 2023-05-10.

[16] Reddit. r/synthesizers. Available at https://www.reddit.com/r/synthesizers/. Accessed: 2023-05-08.

[17] Arturia. Arturia v collection. Available at https://www.thomannmusic.com/arturia_v_collection_9.htm. Accessed: 2023-05-10.

[18] Syntorial. The ultimate synthesizer tutorial. Available at https://www.syntorial.com/. Accessed: 2023-05-10.

[19] Ableton. Ableton learn synth. Available at https://learningsynths.ableton.com. Accessed: 2023-05-10.

[20] Audulus LLC. Audulus 4. Available at https://audulus.com. Accessed: 2023-05-10.

[21] ACIDS. Universal audio synthesizer control. Available at http://acids.ircam.fr/flowsynth/. Accessed: 2023-05-10.

[22] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos. Universal audio synthesizer control with normalizing flows, 2019.

[23] Fanny Roche, Thomas Hueber, Maëva Garnier, Samuel Limier, and Laurent Girin. Make that sound more metallic: Towards a perceptually relevant control of the timbre of synthesizer sounds using a variational autoencoder. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 4:52–66, 2021.

[24] Naotake Masuda and Daisuke Saito. Quality diversity for synthesizer sound matching. In *2021 24th International Conference on Digital Audio Effects (DAFx)*, pages 300–307, 2021.

[25] Katsutoshi Itoyama and Hiroshi G. Okuno. Parameter estimation of virtual musical instrument synthesizers. In *Proceedings - 40th International Computer Music Conference, ICMC 2014 and 11th Sound and Music Computing Conference, SMC 2014 - Music Technology Meets Philosophy: From Digital Echos to Virtual Ethos*, pages 1426–1431. National and Kapodistrian University of Athens, 2014.

[26] Matthew John Yee-King, Leon Fedden, and Mark d'Inverno. Automatic programming of vst sound synthesizers using deep networks and other techniques. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2:150–159, 2018.

[27] Oren Barkan, David Tsiris, Ori Katz, and Noam Koenigstein. Inversynth: Deep estimation of synthesizer parameter configurations from audio signals, 2019.

[28] u he. Diva-vst. Available at `https://u-he.com/products/diva/`. Accessed: 2023-05-12.

[29] Julius Dobos. Analog vs. digital synthesizers – my take on the old debate. Available at `https://www.juliusdobos.com/the-philosophy-behind/analog-vs-digital-synthesizers-debate/`. Accessed: 2023-05-12.

[30] Fedden. Fedden/renderman: Command line c++ and python vsti host library with mfcc, fft, rms and audio extraction and .wav writing. Available at `https://github.com/fedden/RenderMan`. Accessed: 2023-05-08.

[31] Mallat Stéphane. Chapter 1 - sparse representations. In Mallat Stéphane, editor, *A Wavelet Tour of Signal Processing (Third Edition)*, pages 1–31. Academic Press, Boston, third edition edition, 2009.

[32] Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Feature learning and deep architectures: new directions for music informatics. *Journal of Intelligent Information Systems*, 41:461–481, 2013.

[33] Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In *International Society for Music Information Retrieval Conference*, 2012.

[34] OpenAI. chatgpt. Available at `https://openai.com/blog/chatgpt`. Accessed: 2023-05-10.

[35] Mohammed Lubbad. The ultimate guide to gpt-4. Available at `https://tinyurl.com/mwmbu78w`. Accessed: 2023-05-12.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[37] McFee et al. librosa/librosa: 0.10.0.post2, March 2023.

[38] acids ircam. Diva-datset. Available at `https://nubo.ircam.fr/index.php/s/nL3NQomqxced6eJ`. Accessed: 2023-05-12.

[39] M. Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks, 2017.

[40] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[41] Joseph Turian, Jordie Shier, George Tzanetakis, Kirk McNally, and Max Henry. One billion audio sounds from gpu-enabled modular synthesis, 2021.

[42] David Braun. Dawdreamer: Bridging the gap between digital audio workstations and python interfaces, 2021.

[43] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[44] Charles R. et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[47] Denisha Jayaweera and Norashid Aziz. Reliability of principal component analysis and pearson correlation coefficient, for application in artificial neural network model development, for water treatment plants. *IOP Conference Series: Materials Science and Engineering*, 458:012076, 12 2018.

[48] Ian Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] Karl F.R.S. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[51] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[52] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016.

[53] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[54] hsinyilin19. Resnetvae. Available at `https://github.com/hsinyilin19/ResNetVAE`. Accessed: 2023-05-12.

[55] Suki Lau. adaptive-learning-rate-methods-for-deep-learning. Available at `https://tinyurl.com/4bpctazr`. Accessed: 2023-05-12.

[56] PyTorch. Pytorch documentation — pytorch 2.0 documentation. Available at `https://pytorch.org/docs/stable/index.html`. Accessed: 2023-05-08.

[57] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[58] Peter I. Frazier. A tutorial on bayesian optimization, 2018.

[59] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[60] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[61] Wikipedia. Physical modelling synthesis. Available at `https://en.wikipedia.org/wiki/Physical_modelling_synthesis` (2023/03).

[62] Wikipedia. Sample-based synthesis. Available at `https://en.wikipedia.org/wiki/Sample-based_synthesis` (2023/03).

[63] Ben Hayes, Charalampos Saitis, and György Fazekas. Neural waveshaping synthesis, 2021.

[64] Antoine Caillon and Philippe Esling. Rave: A variational autoencoder for fast and high-quality neural audio synthesis, 2021.

[65] Franco Caspe, Andrew McPherson, and Mark Sandler. Ddx7: Differentiable fm synthesis of musical instrument sounds, 2022.

[66] Deepak Kumar Jain, Akshi Kumar, Linqin Cai, Siddharth Singhal, and Vaibhav Kumar. Att: Attention-based timbre transfer. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2020.

[67] Jesse H. Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: differentiable digital signal processing. *CoRR*, abs/2001.04643, 2020.

[68] Xavier Serra et al. Musical sound modeling with sinusoids plus noise. *Musical signal processing*, pages 91–122.

[69] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation, 2018.

[70] Lamtharn Hantrakul, Jesse Engel, Adam Roberts, and Chenjie Gu. Fast and flexible neural audio synthesis. In *International Society for Music Information Retrieval Conference*, 2019.

[71] Daisy Blacklock. The musical instruments you play. Available at `https:https://yougov.co.uk/topics/entertainment/articles-reports/2011/11/01/musical-instruments-you-play` (2023/03).

[72] Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. Creating A musical performance dataset for multimodal music analysis: Challenges, insights, and applications. *CoRR*, abs/1612.08727, 2016.

[73] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning, 2020.

[74] Anaconda.org. dtw python. Available at `https://anaconda.org/conda-forge/dtw-python` (2023/02).

[75] Javier Nistal, Cyran Aouameur, Stefan Lattner, and Gaël Richard. VQCPC-GAN: variable-length adversarial audio synthesis using vector-quantized contrastive predictive coding. *CoRR*, abs/2105.01531, 2021.

[76] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Fréchet audio distance: A metric for evaluating music enhancement algorithms, 2019.

[77] Flávio Ribeiro, Dinei Florêncio, Cha Zhang, and Michael Seltzer. Crowdmos: An approach for crowdsourcing mean opinion score studies. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2416–2419, 2011.

[78] Haedam OH. 3yp-music-synthesis-. Available at `https://github.com/HaedamOh/3YP-Music-synthesis-.git`.

[79] Lenny Renault. Differentiable piano model for midi-to-audio performance synthesis. Available at `https://github.com/lrenault/ddsp-piano.git` (2023/05).

[80] François Rigaud, Bertrand David, and Laurent Daudet. A parametric model of piano tuning. In *Proc. of the 14th Int. Conf. on Digital Audio Effects (DAFx-11)*, pages 393–399, 2011.

[81] T.S. Verma and T.H.Y. Meng. An analysis/synthesis tool for transient signals that allows a flexible sines+transients+noise model for audio. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, volume 6, pages 3573–3576 vol.6, 1998.

[82] Anubhav Chhabra, Aryan Veer Singh, Ritesh Srivastava, and Veena Mittal. Drum instrument classification using machine learning. In *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 217–221, 2020.

[83] Iver Jordal. audiomentations 0.30.0. Available at `https:https://pypi.org/project/audiomentations/` (2023/03).

[84] David Snyder, Guoguo Chen, and Daniel Povey. MUSAN: A Music, Speech, and Noise Corpus, 2015. arXiv:1510.08484v1.

[85] National Gugak Center. Gugak digital audio. Available at `https://www.gugak.go.kr/digitaleum/front/main.do`.

[86] A. RohitM., Amitrajit Bhattacharjee, and Preeti Rao. Four-way classification of tabla strokes with models adapted from automatic drum transcription. In *International Society for Music Information Retrieval Conference*, 2021.

[87] scikit-learn developers. sklearn.manifold.tsne. Available at `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`.

[88] Philippe Esling, Axel Chemla-Romeu-Santos, and Adrien Bitton. Generative timbre spaces with variational audio synthesis. *ArXiv*, abs/1805.08501, 2018.

[89] Keitaro Tanaka, Ryo Nishikimi, Yoshiaki Bando, Kazuyoshi Yoshii, and Shigeo Morishima. Pitch-timbre disentanglement of musical instrument sounds based on vae-based metric learning. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 111–115, 2021.

[90] John M Grey. Multidimensional perceptual scaling of musical timbres. *the Journal of the Acoustical Society of America*, 61(5):1270–1277, 1977.

[91] Am '22: Proceedings of the 17th international audio mostly conference. pages 36–43, New York, NY, USA, 2022. Association for Computing Machinery.

[92] Vesa Valimaki, Julian D. Parker, Lauri Savioja, Julius O. Smith, and Jonathan S. Abel. Fifty years of artificial reverberation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(5):1421–1448, 2012.

[93] Jay Sterling Coggin. Automatic design of feedback delay network reverb parameters for perceptual room impulse response matching. 2015.

[94] Convolution architecture. Available at `https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/convolution.html`.

[95] Convolution reverb. Available at `https://professionalcomposers.com/convolution-reverb/`, Dec 2022.

[96] Madeleine Bloom. Convolution reverb: How does it work and what is it good for?, 2019.

[97] James B. Cole, Rudolph A. Krutar, Susan K. Numrich, and Dennis B. Creamer. Finite-difference time-domain simulations of wave propagation and scattering as a research and educational tool. *Computers in Physics*, 9(2):235, 1995.

[98] Harri Renney, Benedict R. Gaster, and Tom Mitchell. OpenCL vs. In *Proceedings of the International Workshop on OpenCL*. ACM, May 2019.

[99] dave moffat and mark sandler. an automated approach to the application of reverberation. *journal of the audio engineering society*, october 2019.

[100] Tom Erbe. Building the erbe-verb: Extending the feedback delay network reverb for modular synthesizer use. 2015.

[101] Corey Kereliuk, Woody Herman, Russel Wedelich, and Daniel J Gillespie. Modal analysis of room impulse responses using subband esprit. In *Proceedings of the International Conference on Digital Audio Effects*, 2018.

[102] Karolina Prawda et al. Flexible real-time reverberation synthesis with accurate parameter control. In Gianpaolo Evangelista, editor, *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx-20)*, International Conference on Digital Audio Effects, pages 16–23, September 2020.

[103] Jingfei Liu and Nico F. Declercq. Pulsed ultrasonic comb filtering effect and its applications in the measurement of sound velocity and thickness of thin plates. *Ultrasonics*, 75:199–208, March 2017.

[104] M. A. Gerzon. Synthetic stereo reverberation part i. *Studio Sound*, 13:632–635, 1971.

[105] M. A. Gerzon. Synthetic stereo reverberation part ii. *Studio Sound*, 14:24–28, 1972.

[106] J. Stautner and M. Puckette. Designing multichannel reverberators. *Computer Music Journal*, pages 52–65, 1982.

[107] J.-M. Jot and A. Chaigne. Digital delay networks for designing artificial reverberators. In *Audio Engineering Society Convention 90*. Audio Engineering Society, 1991.

[108] Michael Chemistruck, Kyle Marcolini, and Will Pirkle. Generating matrix coefficients for feedback delay networks using genetic algorithm. In *AES Convention 133*, Coral Gables, FL, USA, 2012. Audio Engineering Society.

[109] Alessandro Cerioli. Parameters estimation of feedback delay network reverberation with artificial intelligence algorithms. 2020.

[110] Commercial Acoustics. Reverberation time graphic. `https://commercial-acoustics.com/reverberation-time-graphic/`.

[111] Johannes Horngren. Optimization approaches for feedback delay network. 2021.

[112] Protons for Breakfast. The physics of guitar strings, 2022.

[113] Cathedral room, shasta lake caverns. http://www.echothief.com/cathedral-room-shasta-lake-caverns/, 2023.

[114] Julius O. Smith. *Physical Audio Signal Processing*. http://ccrma.stanford.edu/ jos/pasp///~ccrma.stanford.edu/~jos/pasp/, accessed <date>. online book, 2010 edition.

[115] 7 different vocal reverb effects and when to use them. https://tinyurl.com/3n6cuwjt.

[116] Types of reverb. https://www.stockmusicmusician.com/blog/types-of-reverb.

[117] Pawel Kossecki Jacek Wachowicz, Andrzej Karp. On the friendliness of physical vs. virtual manipulators in music software - from a perspective of software developers. *2022 IEEE 16th International Scientific Conference on Informatics (Informatics)*, pages 351–357, 2022.

[118] Doepfer. A-100 cases overview. Available at https://doepfer.de/a100_man/a100m_e.htm.

[119] Doepfer. *System A-100 Owner's Manual*. Gräfelfing, Upper Bavaria, Germany, pp.3, 8-9.

[120] Behringer. *Behringer 110 VCO/VCF/VCA Owner's Manual*. Kirchardt, Germany, pp.1.

[121] Ivan Sholomove. Understanding diva's different oscillators, filters and envelopes. Available at :https://tinyurl.com/45rhcnr5(2021/05).

[122] Korg. *Korg Monophonic MS-20 Owner's Manual*. Tokyo, Japan, pp.1-8.

[123] Moog. *Minimoog Model D Owner's Manual*. Elma, New York, United States, pp.16-17, 26-27.

[124] Gordan Reid. Of responses and resonance, sound on sound. Available at https://www.soundonsound.com/techniques/responses-resonance.

[125] Ray Wilson. *Make: Analogue synthesizers*. Maker Media, Sebastopol, CA, 2014, pp.141-143.

[126] Hommer Zhao. Colpitts oscillator: Circuit application and its advantages, wellpcb. Available at https://www.wellpcb.com/colpitts-oscillator.html.

[127] EC Studio Systems. Colpitts oscillator. discover - basic electronics - oscillators. Available at https://ecstudiosystems.com/discover/textbooks/basic-electronics/oscillators/colpitts-oscillator/.

[128] Wayne Storr. Colpitts oscillator tutorial and colpitts oscillator design, basic electronics tutorials. Available at https://www.electronics-tutorials.ws/oscillator/colpitts.html.

[129] Cadence. Schmitt trigger hysteresis provides noise-free switching and output, cadence pcb design and analysis. Available at https://tinyurl.com/ywuhw4kn.

[130] Circuit Digest. What is a servo motor? - understanding the basics of servo motor working, servo motor basics, working principle and theory. Available at https://circuitdigest.com/article/servo-motor-working-and-basics.

[131] Circuit Digest. What is a servo motor? - understanding the basics of servo motor working, servo motor basics, working principle and theory. Available at https://circuitdigest.com/article/servo-motor-working-and-basics.

[132] Circuit Digest. What is a servo motor? - understanding the basics of servo motor working, servo motor basics, working principle and theory. Available at https://circuitdigest.com/article/servo-motor-working-and-basics.

[133] EEPower. Potentiometer: Resistor types: Resistor guide. Available at https://eepower.com/resistor-guide/resistor-types/potentiometer.

[134] teachComputing. How to attach a potentiometer to a motor. Available at https://www.hackster.io/allpartscombined/motorized-potentiometer-3240df.

[135] AllPartsCombined. Motorized potentiometer, hackster.io. Available at https://www.hackster.io/allpartscombined/motorized-potentiometer-3240df.

[136] Rs online: Potentiometers. https://uk.rs-online.com/web/p/potentiometers/7377726. Accessed: May 17, 2023.

[137] Rs online: Dc motors. https://uk.rs-online.com/web/p/dc-motors/2389692. Accessed: May 17, 2023.

[138] PJRC. teensy4.0. Available at https://www.pjrc.com/store/teensy40.html. Accessed: 2023-05-13.

[139] Arduino. Arduino-uno. Available at https://store.arduino.cc/products/arduino-uno-rev3. Accessed: 2023-05-13.

[140] Pihut. Price for electronic components. Available at thepihut.com. Accessed: 2023-05-16.

[141] Arduino. Arduino-mega2560. Available at https://store.arduino.cc/products/arduino-mega-2560-rev3. Accessed: 2023-05-13.

[142] Raspberry Pi. Raspberry-pi-4-model-b. Available at https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed: 2023-05-13.

[143] Adafruit. Adafruit 16-channel 12-bit pwm/servo driver - i2c interface - pca9685. Available at https://www.adafruit.com/product/815. Accessed: 2023-05-12.

[144] JUCE Development. Plugins, adding plug-in parameters, saving and loading your plug-in state... Available at https:https://juce.com/learn/tutorials/ (2023/05).

[145] Francesco Ganis, Erik Frej Knudsen, Søren Lyster, Robin Otterbein, David Südholt, and Cumhur Erkut. Real-time timbre transfer and sound synthesis using ddsp. 06 2021.

[146] Pytorch. Pytorch c++ api. Available at https://pytorch.org/cppdocs/ (2023/03).

[147] Google. Google cloud resources. Available at https://cloud.google.com/free?hl=ko (2023/03).

[148] Sotrage MongoDB. Google cloud resources. Available at https://www.mongodb.com/docs/manual/storage/ (2023/03).

[149] Infrastructure, Projects Authority, and Cabinet Office. Project and programme management. Available at https://www.gov.uk/guidance/project-and-programme-management (2023/05).

[150] Ecwid Blog. Profit margin for retail businesses.

[151] Rs components. https://uk.rs-online.com/web/.

[152] Google price page of teensy 4.0. https://tinyurl.com/yckcawx7/.

[153] Adafruit industries: Adafruit music maker featherwing. https://www.adafruit.com/product/815.

[154] Synthcube. https://tinyurl.com/449s97zu.

[155] Instant Offices. Instant offices - oxford business park south. https://tinyurl.com/5cck852t.

[156] Small Business - Chron. Average cost per month for office supplies. https://tinyurl.com/bdemu59f.

[157] Glassdoor. Glassdoor - oxford administrative assistant salary. https://tinyurl.com/2da3cv5c.

[158] Sound On Sound. Sound on sound. https://www.soundonsound.com/about-sos.

[159] Glassdoor. Glassdoor. https://tinyurl.com/vsx3psbb.

[160] Zoopla. https://tinyurl.com/4r5vrum3.

[161] Adafruit industries: Adafruit music maker featherwing. https://www.adafruit.com/product/815.

[162] Gear4music: Eurorack effects modules. https://www.gear4music.com/synthesizer/Modular/Eurorack-Modules/Effects-Modules.

[163] Ample sound: Virtual guitar, bass, and virtual instruments. https://www.amplesound.net/en/index.asp. Accessed May 15, 2023.

[164] Fact Magazine. Buying your first synth: Advice and best deals. https://tinyurl.com/3xdwb8sb, 2017.

[165] Investopedia. How venture capitalists make investment choices. https://tinyurl.com/3bhsy5ff.

[166] Investopedia. Series b, c, and beyond: What it all means and how it works. https://tinyurl.com/52wzxyvj.

[167] Seed funding: The essential guide. https://seedlegals.com/resources/seed-funding-the-essential-guide/.