Neural Computing and Applications (2021) 33:67–84
https://doi.org/10.1007/s00521-020-05424-2

S. I : NEURAL NETWORKS IN ART, SOUND AND DESIGN

Latent Timbre Synthesis

Audio-based variational auto-encoders for music composition and sound design applications

Kıvanç Tatar[1] · Daniel Bisig[2] · Philippe Pasquier[1]

Received: 26 June 2020 / Accepted: 5 October 2020 / Published online: 20 October 2020

Abstract
We present the Latent Timbre Synthesis, a new audio synthesis method using deep learning. The synthesis method allows composers and sound designers to interpolate and extrapolate between the timbre of multiple sounds using the latent space of audio frames. We provide the details of two Variational Autoencoder architectures for the Latent Timbre Synthesis and compare their advantages and drawbacks. The implementation includes a fully working application with a graphical user interface, called *interpolate_two*, which enables practitioners to generate timbres between two audio excerpts of their selection using interpolation and extrapolation in the latent space of audio frames. Our implementation is open source, and we aim to improve the accessibility of this technology by providing a guide for users with any technical background. Our study includes a qualitative analysis where nine composers evaluated the Latent Timbre Synthesis and the *interpolate_two* application within their practices.

Keywords Audio synthesis · Neural networks · Signal processing · Computer assisted music composition

1 Introduction

Modern sound synthesizers come loaded with many parameters, with very large nonlinear, non-modal, search spaces. This richness comes to the detriment of searchability as one cannot easily or efficiently find a particular sound, particular sonic textures, or generate a transition between two textures. Consequently, sound designers and musicians most often rely on audio samples (of instruments, sound effects) and their manipulation rather than the more flexible sound synthesis approach of these sounds and their sonic surroundings. In previous work on synthesizer preset generation [35], we demonstrated how, given a target sample, PresetGen can find a preset that generates the closest to the sample. In this work, we investigate a new DL based-method by which a synthesizer model is trained on selected audio textures, allowing musicians and sound designers to achieve their synthesis goals by exploring a sonic space, with interpolation and extrapolation between sonic textures.

The rise in popularity of deep learning (DL) architectures has led to promising new research using deep learning (DL) for musical applications, audio transformation and sound synthesis [2]. The demand for sound synthesizers is projected to grow at an accelerated rate in the next five years [39], and in parallel, there is increasing interest in flexible, versatile, yet controllable synthesis engines. In addition to improving upon previous models in this area, Latent Timbre Synthesis (LTS) is a framework for audio synthesis models that can be used and manipulated by musicians and composers, by offering a graphical user interface and intuitive workflow, using fully open-source software.

In this study, our motivation centers around the apparent lack of ML-based tools for professional composers and musicians of Electroacoustic Music, Sound Art, New

✉ Kıvanç Tatar
ktatar@sfu.ca

Daniel Bisig
daniel.bisig@zhdk.ch

Philippe Pasquier
pasquier@sfu.ca

[1] Simon Fraser University, Vancouver, BC, Canada
[2] Zurich University of the Arts, Zurich, Switzerland

Media, and other interdisciplinary artistic styles that integrate a generalized understanding of sound. We aim to provide insights into how state-of-the-art ML models can be integrated into the professional workflows for musical creation, and whether this integration can foster new creative ideas. For this reason, our project focuses on the integration of modern ML and AI techniques into tools for computer-assisted composition and their applications within musical practices. LTS helps composers by utilizing an abstract latent timbre space that is generated by training unsupervised deep learning (DL) models with a set of audio recordings. These new DL tools allow composers to synthesize sounds using a latent space of audio that is constrained to the timbre space of the audio recordings in the training set.

The project follows a practical approach that combines the development of ML-based tools in parallel with musical creation and subsequent user evaluation with professional composers and musicians among communities from the fields of experimental electronic music and sound art. To limit the scope of the project, we focus specifically on audio corpus-based sound synthesis approaches that rely on large libraries of audio excerpts. We encouraged the invited composers to gain familiarity with and employ our ML-based tools in the context of a practical musical production scenario. The user evaluation in Sect. 7 takes the form of textual interviews that probe the participants experiences while working with LTS tools and their influence on the technical, aesthetic, and conceptual aspects of their creation process.

We hope these project outcomes highlight the benefits and challenges of integrated state-of-the-art ML algorithms into the creative ideation and production workflows of professional composers and musicians.

2 Musical approach

In 1940s, physicist Dennis Gabor proposed [7] that a sound is composed of acoustical quanta that are bounded by time and frequency. We are inspired by the idea of acoustical quanta as well as the definition of music as “nothing but organized sound” [41] involving sound objects [30] that situate on multiple layers [33] where any sound can be used to produce music [19, 41], and strong connections exist between pitch, noise, timbre, and rhythm [28, 29, 32, 33]. In that sense, the Latent Timbre Synthesis project builds on our previous work titled Musical Agents based on Self-Organizing Maps (MASOM) [36, 38]. MASOM combines organizing sound samples in latent audio space with statistical sequence models for musical structure. The latent audio space in MASOM is generated by a Self-Organizing Map that organizes a set of audio excerpts.

In LTS, we move further by aiming for an audio synthesis framework where we can synthesize sounds that do not exist in the training set. Unsupervised DL architectures have the flexibility of meta-creativity where practitioners can create their own personalized audio synthesizers by curating the training dataset of DL frameworks. LTS tools for music composition and sound design utilize an abstract latent timbre space that is generated by training unsupervised DL models on a set of audio recordings. These new DL tools can help practitioners in computer-assisted composition by synthesizing from a latent space of timbre that is constrained to the timbre space of the audio recordings in the training set.

Following the research motivation and musical approach outlined above, our contributions presented in this paper include two Variational Auto-encoders (VAEs) to generate a latent space of audio frames. Unlike other deep learning architectures such as Generative Adversarial Networks (GANs), VAEs are beneficial for our applications because these architectures can encode an existing audio frame to a latent space, as well as synthesize audio frames from latent vectors. VAEs also allow audio synthesis through interpolation and extrapolation of timbres, by using the latent vectors of audio frames.

Latent Timbre Synthesis differs from the previous works such as Granma MagNet [1] because we prioritize the flexibility of variable length audio generation, in comparison with sound samples of fixed-duration. We think that the flexibility of changing the duration of the generated audio is crucial for our applications, which stands out as another contribution of LTS. Our approach focuses on creating a latent space of audio frames, where we can represent an audio recording with any length as a time-series sequence of latent vectors.

The LTS framework consists of three main modules, calculation of wavelet transform-based spectrogram representation, latent audio frame space generation using two specific Variational Auto-encoders and inverse synthesis using wavelet-based magnitude spectrogram generated by the decoder of the VAE. We explain an iterative design of two VAE architectures to create an audio synthesis tool for composition practices and sound design applications. In addition, we present and share a fully working application, called *interpolate_two* with a Graphical User Interface (GUI) that allows composers to synthesize audio using timbre interpolation and extrapolation with multiple sounds. In comparison with the high computational complexity of previous works mentioned in Sect. 3, the low computational complexity of *interpolate_two* allows the incorporation of the musical tool within real-time applications. The documentation of the setup of *interpolate_two* is detailed to guide practitioners of all backgrounds. Our implementation is open source[1], and sound examples are available[2]. We encourage our readers to dive into the code and experiment with the framework for further audio synthesis possibilities.

3 Related works

We limit this section to the previous works that utilize audio spectrogram as an input for the deep learning architecture, with the exception of WaveNet. We situate the Latent Timbre Synthesis project within the raw-audio generation applications of deep learning, and WaveNet is one of the state-of-the-art systems in the area. We also omit deep learning systems for speech synthesis or vocoder applications, such as MelGAN [16], while mentioning in Sect. 8 how we plan to incorporate them in LTS as a next step of our research.

WaveNet is a deep learning architecture that uses an audio corpus for the tasks of music composition, multi-speaker speech and text to speech generation, and speech recognition [25]. WaveNet applies Convolutional Neural Networks (CNNs) with two strategies to handle temporality of raw audio data: causal convolution and dilation. Causal convolutions ensure that the output only depends on the past observations. Dilated convolutions skip a number of inputs on each layer. The number of skipped inputs increases exponentially with each layer; hence, the receptive field of the network also increases exponentially [43]. Oord et al. [25] tested WaveNet on two audio corpora: the MagnaTagATune dataset and the YouTube piano dataset. The authors point out that "Even with a receptive field of several seconds, the models did not enforce long-range consistency which resulted in second-to-second variations in genre, instrumentation, volume and sound quality." That is, WaveNet struggled to generate long-term variations like in the case of interactive music systems that apply Markov Models [37, Sect. 6.1]. There has been follow-up research on the WaveNet architecture, where the authors stack multiple WaveNet architectures on top of each other [26], or they combine WaveNet with Vector Quantized Variational Autoencoders [4]. The main drawback of all WaveNet systems is their computational complexity and high usage of GPU memory [3]. The technology requirements of WaveNet limit its usefulness in compositional practices, where composers do not necessarily have access to computers equipped with state-of-the-art GPUs.

Differentiable Digital Signal Processing (DDSP) is a toolbox made by Google for research into Digital Signal Processing (DSP) applications of deep learning [5]. The authors describe the DDSP Autoencoder, which is a VAE architecture where the inputs are the Mel-Frequency Cepstral Coefficients (MFCCs) of an audio excerpt. We mention a comparison of using MFCCs and other audio features as the representation of timbre for audio synthesis with VAEs in Sect. 4.1. The architecture employs three autoencoders for fundamental frequency (*f-encoder*), loudness (*l-encoder*), and the latent space of timbre (*z-encoder*). The fundamental frequency and the loudness encoders use the CREPE architecture that is originally presented as a pitch detector [10]. The *z-encoder* architecture is inspired by the ResNet architecture in Computer Vision research [11]. The decoder, on the other hand, controls the input parameters of an additive synthesis module, a subtractive synthesis module, and a reverb. These three synthesis modules generate the final audio. The loss function compares the generated audio with the original one, using a specific function called Multi-Scale Spectrogram Loss, which is similar to comparing the spectrograms of original and generated audio.

The Generative Timbre Spaces project [6] is perhaps one of the most similar previous studies to the Latent Timbre Synthesis project. The application of Generative Timbre Synthesis focuses on generating a latent timbre space of conventional musical instruments. This model uses a VAE where the encoder is a 3-layer feed-forward network with 2000 units in each layer. The latent space has 64 dimensions. The authors introduce a new regularization item in the cost function. The additional regularization loss tries to force the network to satisfy perceptual similarity ratings of conventional musical instruments in Western Classical Music. These perceptual ratings are proposed in previous studies [8, 12, 15, 17, 21]. The training dataset of Generative Timbre Spaces consists of audio recordings of conventional musical instruments where each file is an instrument playing a note. The authors takes one frame from each audio file to train the VAE model. Hence, the architecture aims to capture the generalized timbre of a conventional musical instrument instead of the regeneration of an arbitrary audio excerpt. As further limitation, the cost function is not suitable to regenerate a dataset with arbitrary audio recordings because there are no perceptual ratings available. We further discuss the issues related to the hyper-parameters of VAE in Generative Timbre Spaces in Sect. 4.2.

The DDSP Autoencoder and Generative Timbre Spaces aim for the synthesis of conventional music where the

[1] The source code is available at https://www.gitlab.com/ktatar/latent-timbre-synthesis.

[2] We provide sound examples at https://kivanctatar.com/Latent-Timbre-Synthesis.

model is conditioned to output audio with a fundamental frequency constraint. In sound design, experimental electronic music, and sound art applications, having a fundamental frequency of a sound gesture is rather limiting. The music theory of contemporary electronic music emphasizes the continuum between noise, pitch, and rhythm [19, 28, 29, 32, 33]. Hence, in LTS, we aim for an audio synthesizer where composers and practitioners are free to explore the full potential of digital audio synthesis.

4 Audio synthesis framework

The Latent Timbre Synthesis framework consists of three main parts, spectrogram calculation using a specific type of wavelet transform, audio frame latent space generation using Variational Autoencoders, and inverse synthesis of audio using the magnitude spectrogram generated by the decoder of VAE (Fig. 1).

4.1 Wavelet transform-based spectrograms

The audio feature extraction module generates spectrogram vectors using Constant-Q Transform (CQT) [31] that has gained popularity in Music Information Retrieval (MIR) research in the recent years. CQT[3], and its variant, the Non-stationary Gabor Transform (NSGT) [42], have been compared to the other audio features such as Mel-Frequency Cepstral Coefficients (MFCC); and previous studies showed that CQT and NSGT could perform better in MIR applications such as segmentation and musical structure analysis [24]. Naturally, segmentation and musical structure analysis tasks require computing the audio similarity [23]; thus, they are suited to create a latent space of audio frames.

A previous work [6] compared spectrograms computed with fixed windows and wavelet transform-based spectrograms for applications of latent audio frame space generation using deep learning (DL). This comparison included Short-Time Fourier Transform, Discrete Cosine Transform, Constant-Q Transform (CQT), and Non-Stationary Gabor Transform (NSGT) variations using different frequency scales. The wavelet-based transforms in this previous study were CQT and NSGT variants. The authors found that wavelet transform-based spectrogram representations perform better than the spectrograms calculated using fixed-length windows with regard to the log-likelihood and mean quality of the audio frame reconstructions, as shown in Table 1, while the audio frame reconstructions of wavelet transform-based spectrograms gave similar results. We utilize CQT in comparison with other wavelet-based spectrograms in Table 1 because a python library for audio analysis, titled Librosa[4] [22], includes a CQT and inverse CQT implementation [31] combined with a Fast-Griffin-Lim phase estimation [27] that we explain in Sect. 4.3.

[3] Appendix A summarizes the calculation and parameters of CQT.

4.2 Autoencoders and Variational Autoencoders

Autoencoders are deep learning architectures for generative modelling. The architecture consists of two main modules: an encoder and a decoder (Figs. 1 and 2). The encoder maps the input data $x \in R^L$ to a latent vector $z \in R^M$ where $z = encoder(x)$, and $M < L$. The decoder aims to convert a latent vector back to the original data, and ideally, $decoder(encoder(x)) = x$. The Autoencoder architecture encodes the input data vector to a single point, that is the latent vector. In comparison, Variational Autoencoder (VAE) is an improved version of the Autoencoder architecture that converts the input data vector to a stochastic distribution over the latent space. This difference is also referred to as the "reparametrization trick" [13, 14, 34].

In VAE, the encoder tries to generate a latent space by approximating $p(z|x)$ while the decoder tries to capture the true posterior $p(x|z)$. The vanilla VAE approximates $p(z|x)$ using $q(z|x) \in Q$ with the assumption that $p(z|x)$ is in the form of a Gaussian distribution $N(0, I)$. This approximation is referred in the literature as *Variational Inference* [13]. Specifically, the encoder outputs the mean μ_M and the covariance σ_M as the inputs of the Gaussian distribution function $N(z; \mu_M, \sigma_M^2 I)$ over a latent space with M number of dimensions. Hence, the encoder approximates $p(z|x)$ using $q^*(z|x) = N(z; f(x), g(x)^2 I)$ where $\mu_M = f(x), f \in F$, $\sigma_M = g(x)$, and $g \in G$. The decoder's input, the latent vector z is sampled from the latent distribution $q(z) = N(z; f(x), g(x)^2 I)$. Hence, the loss function consists of the reconstruction loss and the regularization term of Kullback–Leibler divergence between $q^*(z|x)$ and $p^*(z)$,

$$L_{f,g} = \mathbb{E}_{q^*(z)}[logp^*(x|z)] - \alpha \cdot D_{KL}[q^*(z|x)||p^*(z)] \tag{1}$$

We direct our readers to the original VAE publication [13] for the mathematical induction of the loss function in Eq. 1 [14]. Note that, some previous works introduced additional regularization terms to the loss function to condition the VAE further, such as the introduction of perceptual ratings of musical instruments in [6].

The LTS framework focuses on Variational Autoencoders in comparison with Generative Adversarial Networks because we aim for audio synthesis by interpolation and extrapolation in the latent space of audio frames (see Sect. 6). The input vectors of the VAE model are CQT vectors calculated from one audio frame where the window

[4] https://librosa.github.io/librosa/.

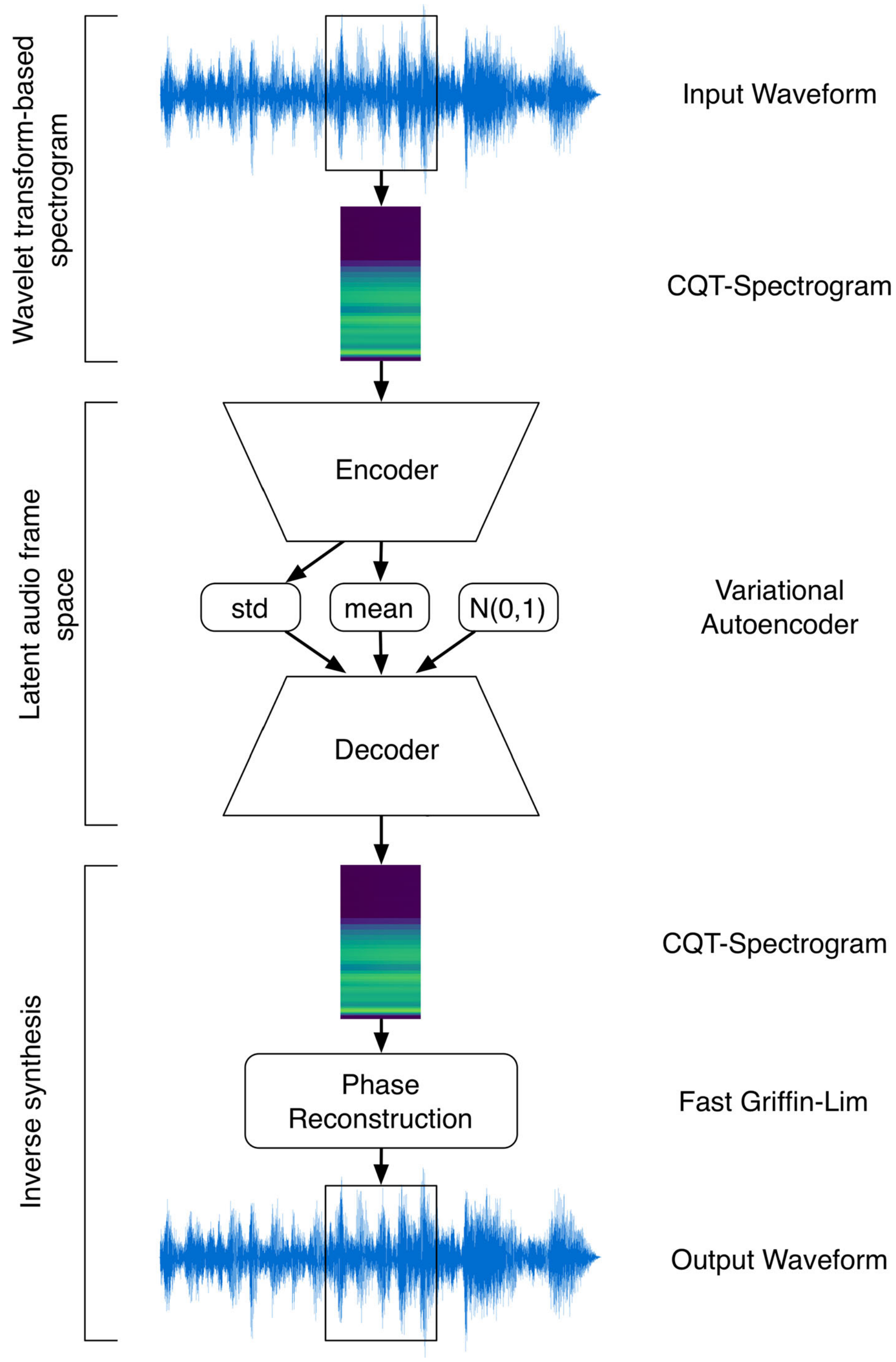


Fig. 1 Latent Timbre Synthesis framework

size varies with the frequency bins. We aim to generate a latent space of audio frames so that we can synthesize audio with any duration. Previous systems, such as Gran-nma MagNet [1], utilize audio excerpts with fixed-duration, where the training input vectors of deep learning model are 2D audio spectrograms with time along the x-axis and frequency along the y-axis. This design choice constraints these DL models to limited applications such as generating a fixed-length audio excerpt. Our approach differs from the previous systems because the training observation of DL model in LTS is one audio-spectrum vector that is calculated from one audio-frame. This allows LTS to generate variable length audio in composition tasks.

4.3 Inverse synthesis and audio reconstruction

This version of LTS uses the Fast Griffin-Lim [27] phase estimation algorithm (F-GLA) for generating audio from

Table 1 A previous study provided a comparison of audio frame reconstruction losses with Variational Autoencoders using spectrograms with fixed-length windows and wavelet transform-based spectrograms [6]

	Spectrogram	$logp(x)$	$\|\|x - \tilde{x}\|\|^2$
Fixed Window	STFT	− 1.9237	0.2412
	DCT	4.3415	2.2629
Wavelet Transform	CQT	− 2.8723	0.1610
	NSGT-MEL	− 2.9184	0.1602
	NSGT-ERB	− 2.9212	0.1511

CQT magnitude spectrograms that the VAE decoder outputs. Briefly, the GLA and F-GLA estimate the phase component of a magnitude spectrogram by iterating the inverse synthesis and the spectrogram calculation multiple times[5]. We use the F-GLA implementation in librosa python library[4] to apply the inverse synthesis with phase estimation to the generated CQT magnitude spectrograms.

The F-GLA module has the highest computation time in the LTS framework. We are aware that a revision of this module using another deep learning architecture for vocoder applications can improve the computational complexity of LTS while making LTS more lightweight within real-time applications. We further address this in Sect. 8.

5 Iterative design of deep learning models

5.1 Audio signal processing

All experiments in this paper used the same audio feature extraction configurations[6]. The original CQT paper [31] mentions that the reconstruction of an original signal from its CQT coefficients results in a signal-to-noise ratio of 55 dB, approximately. We tried several parameters to find the configuration that gave the least amount of audio artifacts with the pipeline of calculating the CQT spectrogram and then reconstructing the audio back using the magnitude spectrogram of CQT combined with the phase estimation algorithm. Notice that these artifacts would appear even with an ideal DL model because the inverse audio synthesis introduces these artifacts to the LTS. We used 16-bit and 44.1 kHz stereo or mono audio recordings. We converted the stereo files to mono first, and then calculated the CQT spectrograms using a hop-size of 128 samples. The f_1 parameter was 32.7 Hz that corresponds to the musical note, C1. q value in Eq. 5 (Appendix A) was equal to 1, and the window function was “hann”[7]. CQT included 48 bins per octave for a total range of 8 octaves; which sums up to a 384 bins in total. Hence, the input of the DL models is vectors with 384 dimensions. These parameters resulted in the least amount of artifacts in our experiments. These parameters can be changed in the source code, and we encourage our readers to try different parameter configurations.

5.2 Variational Autoencoders in Latent Timbre Synthesis

We focused on two deep learning (DL) architectures in the first version of the Latent Timbre Synthesis framework. Both models are Variational Autoencoders (VAE); however, the layers and model parameters differ. The first VAE model in LTS is a lightweight architecture that consists of two dense layers in its encoder and decoder (Fig. 2).

In our experiments, the first architecture resulted in the highest quality of reconstructions of audio signals in comparison with the second architecture. However, this first lightweight architecture had 256 latent dimensions, and we tried to decrease the number of latent space dimensions in the second architecture. We first added more dense layers to the architecture in an iterative manner, while observing an improvement in the final loss values. Following, we introduced a convolutional layer, which improved the final loss values even further. However, additional convolutional layers after the first one did not improve the final loss[8]. Hence, we decided on using the second VAE architecture in LTS, which has one convolutional later and four dense layers (Fig. 3). This architecture provided the lowest loss values in our experiment; yet, the reconstructions of audio signal had audible floor noise. Our trials with instance, batch, layer, and weight normalization techniques as well as upsampling layers and leaky rectified linear activation functions did not improve the final loss or the audio quality of the reconstructions. These trials indicate that around 256 dimensions may be the lowest number of dimensions to create an audio frame latent space that can reconstruct any audio recording without a floor noise, due to the size of digital audio space.

In the first VAE model, we were inspired by previous work [6] where the authors trained a Variational Autoencoder to generate conventional musical instrument timbres with digital audio synthesis. We initially tried the VAE

[5] We outline the inverse CQT algorithm in Appendix B

[6] We summarize the details of CQT calculation in Appendix A

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.hann.html.

[8] Example audio reconstructions using trained models, training statistics with loss values, and hyper-parameter settings are available on the project page: https://kivanctatar.com/latent-timbre-synthesis.

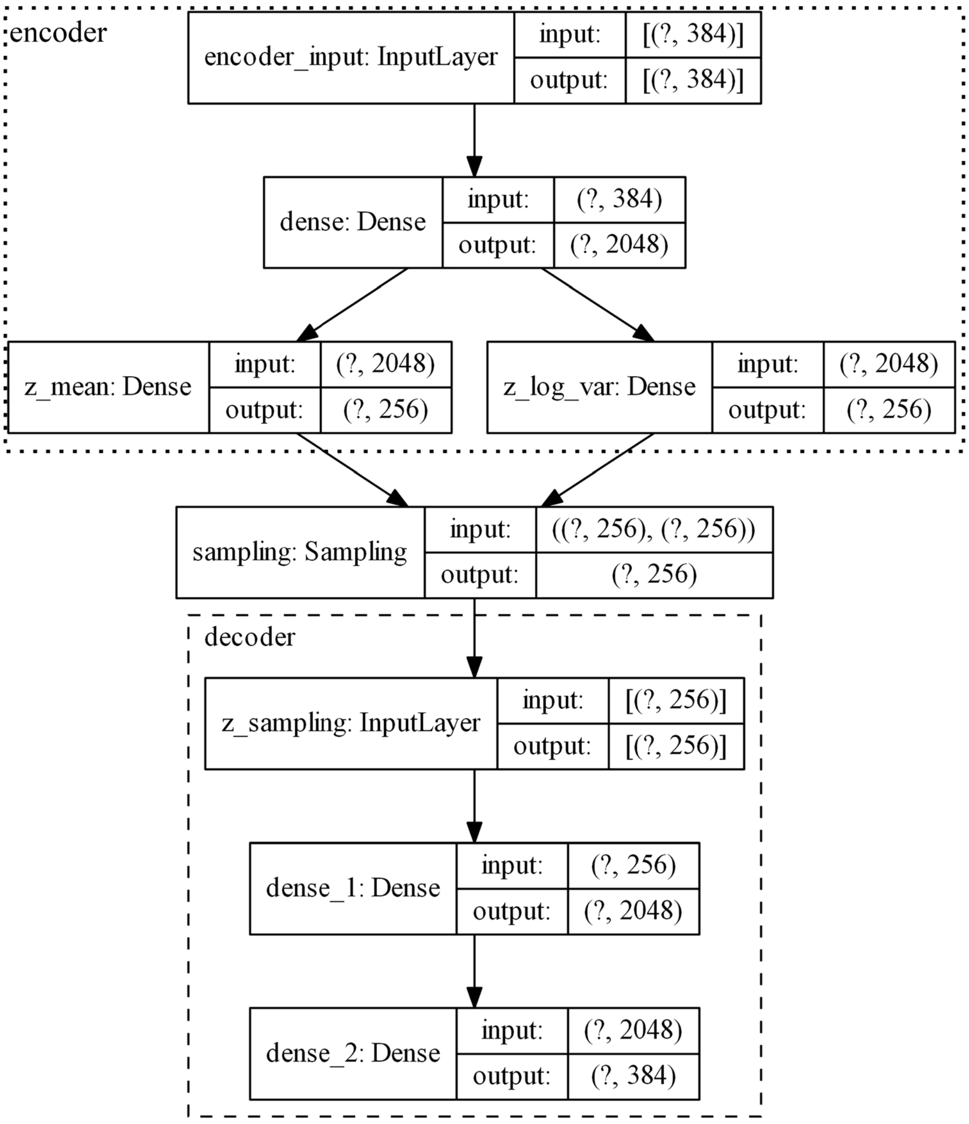


Fig. 2 The first deep learning architecture available in Latent Timbre Synthesis

architecture with the same hyper-parameter settings that were used in the Generative Timbre Spaces project. The setting of Generative Timbre Spaces [6] was unsuccessful in our experiments. The model could not learn to regenerate the audio recordings in the training dataset and could only generate noise. Upon further investigation, we found that the Kullback–Leibler Divergence regularization term multiplier caused this issue. In Generative Timbre Spaces, the authors increase the multiplier from 0 to 2 during the first 100 epochs of the training, following the warm-up procedure [34]. We suspect that the KLD multiplier range of [0, 2] is specific to the application of Generative Timbre Spaces where the training dataset consists of audio files with distinct harmonic content and low noisiness in the spectrum. Furthermore, the training dataset size of Generative Timbre Spaces is rather small, less than 100 MB, whereas we work with GBs of audio to train the LTS models. For example, the *Sample-FX* dataset that we provide with our source code includes 2 GBs of audio that corresponds to 3, 084, 591 audio frames as training data for LTS models, using a hop-size of 128 samples for the CQT calculation. Hence, we found that the range of [0, 2] for KLD multiplier was too high for our application and prohibited the VAE to learn. In addition, the warm-up procedure had adverse effects on the learning.

After investigating the CQT spectrogram reconstructions, we decided to test similar hyper-parameters using a image dataset, the MNIST [18] to further investigate this issue. Working with this image dataset provided us with insights on the hyper-parameter settings such as the value of the KLD multiplier based on their impact on the visual quality of the reconstructions with MNIST. The images in MNIST are $28 * 28$ pixels, adding up to a total of 784 pixels. This is similar to LTS where the input of the VAE is

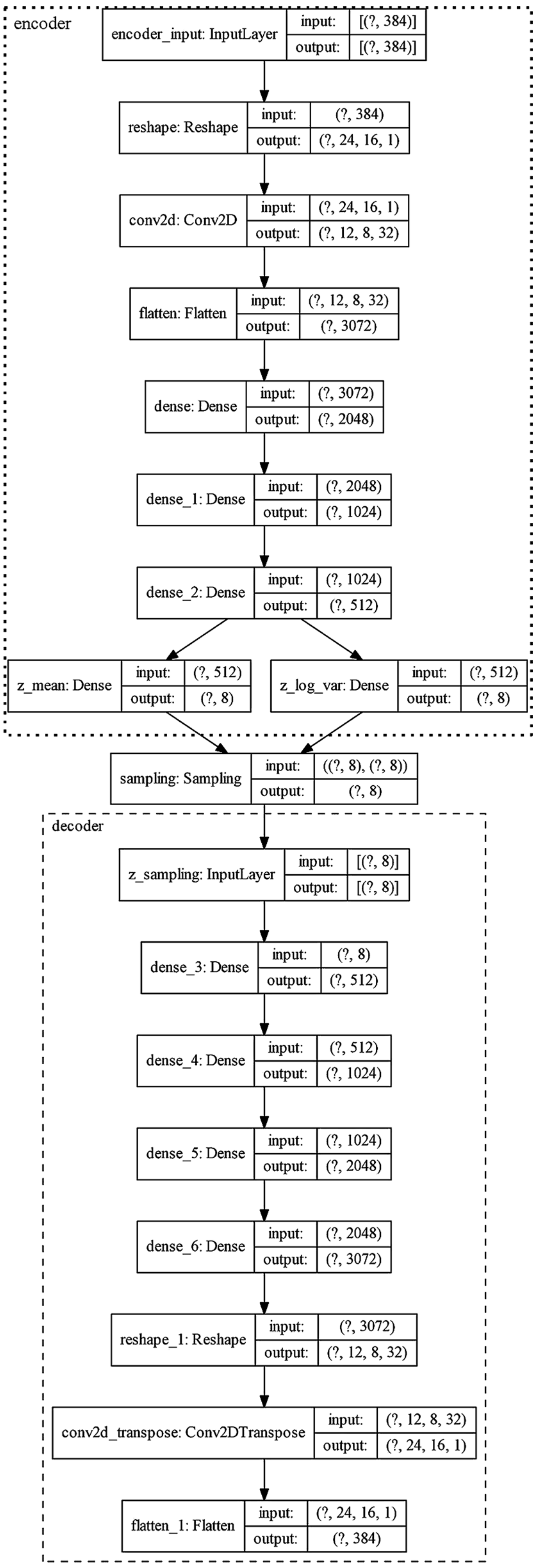

encoder
encoder_input: InputLayer | input: [(?, 384)] | output: [(?, 384)]
reshape: Reshape | input: (?, 384) | output: (?, 24, 16, 1)
conv2d: Conv2D | input: (?, 24, 16, 1) | output: (?, 12, 8, 32)
flatten: Flatten | input: (?, 12, 8, 32) | output: (?, 3072)
dense: Dense | input: (?, 3072) | output: (?, 2048)
dense_1: Dense | input: (?, 2048) | output: (?, 1024)
dense_2: Dense | input: (?, 1024) | output: (?, 512)
z_mean: Dense | input: (?, 512) | output: (?, 8)
z_log_var: Dense | input: (?, 512) | output: (?, 8)
sampling: Sampling | input: ((?, 8), (?, 8)) | output: (?, 8)
decoder
z_sampling: InputLayer | input: [(?, 8)] | output: [(?, 8)]
dense_3: Dense | input: (?, 8) | output: (?, 512)
dense_4: Dense | input: (?, 512) | output: (?, 1024)
dense_5: Dense | input: (?, 1024) | output: (?, 2048)
dense_6: Dense | input: (?, 2048) | output: (?, 3072)
reshape_1: Reshape | input: (?, 3072) | output: (?, 12, 8, 32)
conv2d_transpose: Conv2DTranspose | input: (?, 12, 8, 32) | output: (?, 24, 16, 1)
flatten_1: Flatten | input: (?, 24, 16, 1) | output: (?, 384)

Fig. 3 The second Variational Autoencoder architecture available in Latent Timbre Synthesis

a vector of 384 dimensions. This approach provided valuable additional insights that complemented our tests on hyper-parameter settings to generate audio spectrograms.

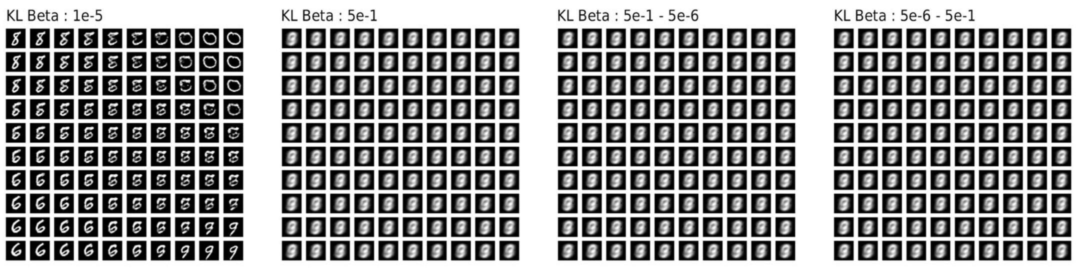
Like in the case of our tests with audio spectrograms, we obtained similar results with the MNIST data. Higher KL-divergence values as well as the warm-up procedure significantly deteriorated the reconstructions of the trained model. Figure 4a, b shows the effect of KLD multiplier on the training of the Variational Autoencoder. We used the same architecture depicted in Fig. 1 and only changed the input and output dimensions to 784 which corresponds to the flattened vector of $28x28$ images in the MNIST dataset. In addition, we tested the warm-up procedure and the reverse settings of the warm-up procedure; however, both were detrimental to the training in our experiments. We proceeded with our experiments using KLD values around $1e-5$, given the success of this setting with the MNIST dataset.

We train the network for 2000 epochs, while the improvements after epoch 50 are rather in the exploitation[9] phase of the learning, and help to minimize the floor noise in the generated CQT magnitude spectrogram. The learning rate is $1e^{-4}$ and the KLD multiplier in the cost function is $5e^{-4}$. The learning rate and the KLD multiplier parameters are dependent on the training dataset. We recommend the readers who would be interested to try their own dataset to start with the hyper-parameter settings above, and change the parameters when needed.

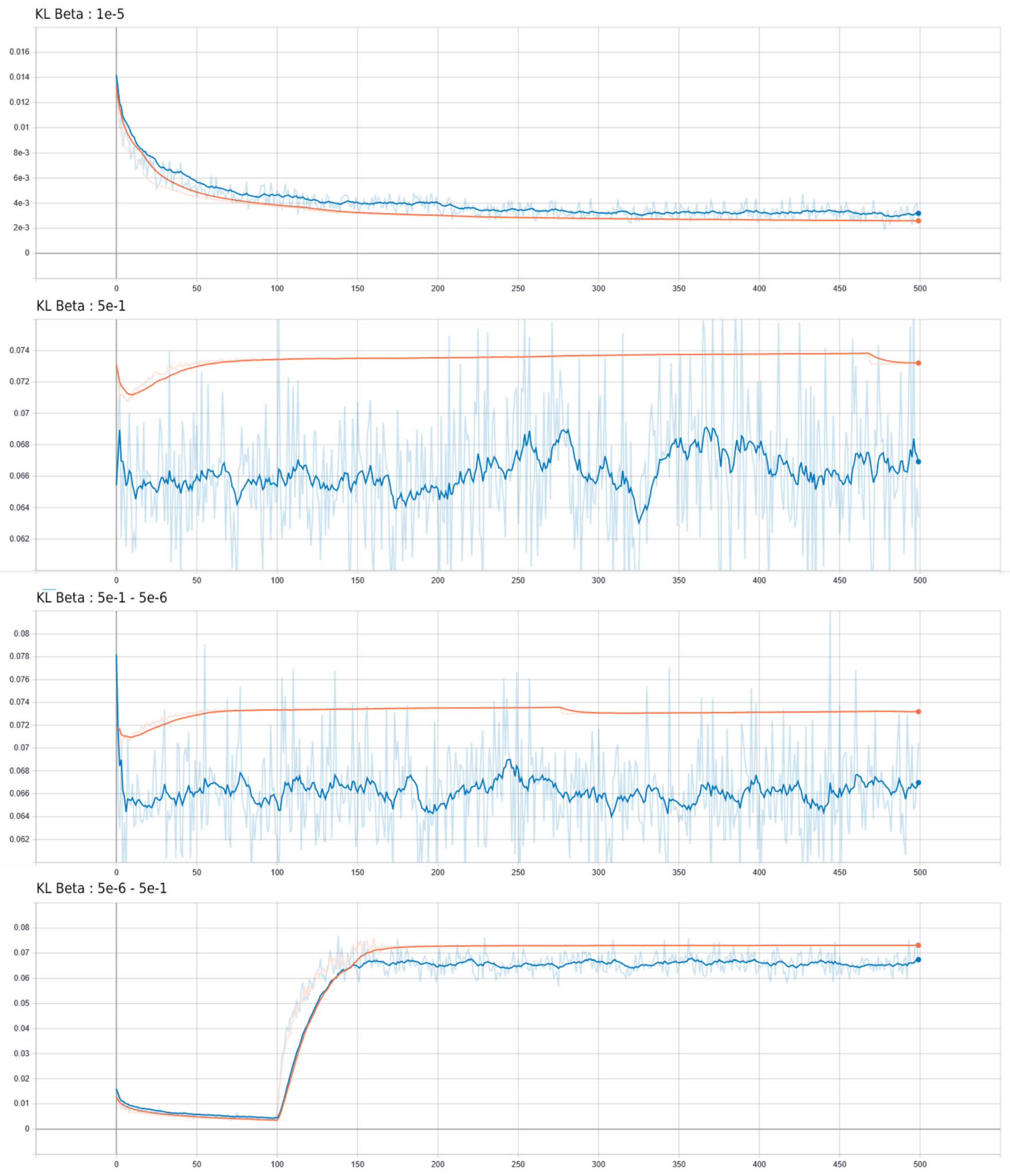
All VAE architectures in LTS use decoder networks that are the reversed replicas of the encoder networks, as in most cases of VAEs. The first VAE model in LTS is a feed-forward network with two dense layers including 2048 neurons, The *dense, dense_1,* and *dense_2* layers in Fig. 2 apply Rectified Linear Units (ReLU) as the neuron activation functions. The latent space of our first VAE model consists of 256 dimensions. This number is still acceptable given that the previous work [6] used a 64-dimensional latent space to cluster a much smaller range of conventional musical instrument timbres. Yet, we explored the possibility to find a deeper network that could generate a latent space with a smaller number of dimensions.

The second VAE network that we present in this paper aimed to decrease the number of latent space dimensions of the first network using a deeper architecture, shown in Fig. 3. We pursued an iterative design procedure where we tried to decrease the number of dimensions of the latent space while maintaining the reconstruction quality and trying to achieve lower final loss values in the training runs. Our experiments aimed for a 8-dimensional latent space, and we first tested increasing the number of dense

[9] Exploration and exploitation are two search strategies in optimization applications [35, Sect. 5.3].



(a) The effect of the KLD multiplier on the reconstructions



(b) The loss values during training where x-axis is the training epoch and y-axis is the loss value

Fig. 4 The effect of the KLD multiplier in the loss function on Variational Autoencoder training

layers. Our experiments found an optimum of 4 dense layers where the number of neurons is changed by twofold for each layer (Fig. 3). In addition to the dense layers, we also tried adding convolutional layers on top of the dense layers. We imagined that the convolutional layers could grasp the relationships between frequency bins of a given CQT vector, such as harmonics. The convolutional layer generates a 2-dimensional vector with size 24x16, which is in relation to 48 bins per octave for 8 octaves while trying to remain as close as to a square. Notice that, each row in the 2D vector generated by the convolutional layer corresponds to half of an octave. We imagined that this could further help the network to capture the harmonic content in the CQT vectors. Our experiments found an optimum of one additional convolutional layer with 32 filters and a kernel size of 3 and a stride size of 2 on top of the dense layers, shown in Fig. 3. The final second network could decrease the number of latent space dimensions while giving a low final loss value. However, the network introduced a consistent floor noise sound in the reconstructions. The introduction of normalization techniques such as batch, instance, or layer normalization could not eliminate the floor noise in the reconstructions. Our findings suggest a balance between the number of latent space dimensions and the floor noise in the reconstructions where increasing the number of latent space dimensions is the solution to eliminate the noise.

Given that audio quality is of great importance in composition tasks, we focused on using and disseminating the first network given in Fig. 2. Additionally, the low computational complexity of the first network can be an advantage when we combine the VAE with deep learning models for time-series sequence generation in our future work, detailed in Sect. 8.

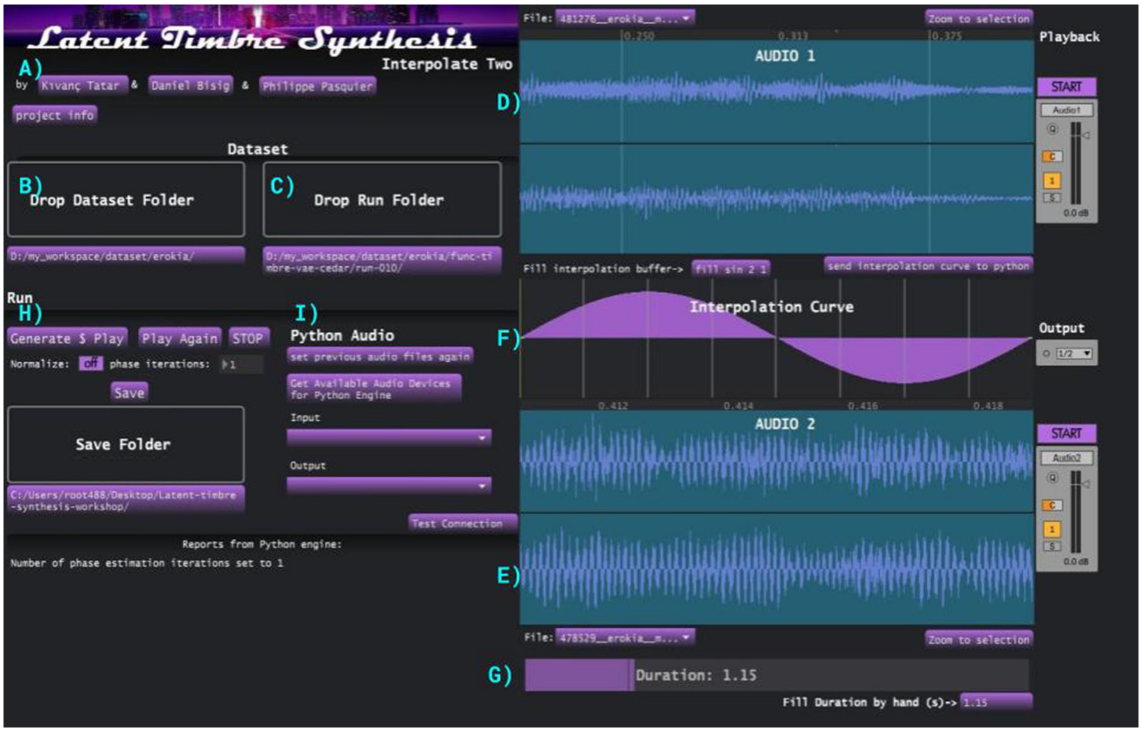
6 User application

The first application of Latent Timbre Synthesis is the *interpolate_two* framework that allows composers to synthesize sounds using interpolation and extrapolation with two sounds. The framework requires a trained model to generate sounds using the latent audio frame space. The user can select the duration of the generated sound, and can choose an excerpt from two audio files. These two excerpts have the same duration. The algorithm uses these two excerpts for synthesis with interpolation and extrapolation in the timbre space. The interpolation amount sets how much of the latent vector is copied from one of the audio excerpts. For example, 30% interpolation is adding 30% of the latent vectors of the first audio and 70% of the latent vectors of the second audio. The percentages above 100 or below 0 corresponds to extrapolations. For instance, 120% is moving 20% away from the second audio in the direction the latent vector that points from audio 1 to audio 2. The algorithm synthesizes the audio by calculating every audio frame using inverse synthesis from the generated magnitude CQT spectrogram. Hence, the user sets interpolation amounts for each latent vector that corresponds to one audio frame. The user can draw an interpolation curve to change the interpolation percentage in time using the LTS framework.

The application, *interpolate_two*, consists of two components: Max GUI and the python engine. The Max GUI handles user interactions while the python engine reacts to OSC messages coming from Max. The python engine runs the deep learning model and audio feature extraction (CQT calculations), as well as inverse synthesis that generates audio from CQT magnitude spectrogram combined with Fast Griffin-Lim phase estimation.

The framework of *interpolate_two* is compatible with all variations of the VAE architectures given in Figs. 2 and 3. The B an C regions in the GUI in Fig. 5 load the datasets and models that are produced within a particular run with the dataset. Using the regions D and E, the user can select two audio files from a dataset to choose excerpts. The file dropdown menu allows the users to select an audio file from the dataset. "Zoom to selection" sets the view to the selection area. Clicking *ctrl* (or *cmd* on MacOS) and then dragging the mouse up and down on the waveform views applies zoom in and out. The waveform in the region F sets apply interpolation (or extrapolation) amounts per frame, where *x*-axis is the time and *y*-axis sets the interpolation percentage for a frame. When both waveform views are zoomed to the selection, the *x*-axis of the interpolation curve corresponds to the *x*-axis of the waveform. The interpolation curve view is a [waveform$\sim$] object. The "Vertical Zoom" parameter in the inspector of this Max object sets the maximum interpolation/extrapolation amount. The default maximum is 1.3; hence, [1.0,1.3] and [− 1.0, − 1.3] are the extrapolation regions. It is possible to extrapolate even more by changing the vertical zoom parameter; however, the higher amounts are likely to give audible distortions. The normalize toggle in region H allows the user to normalize the generated audio to prevent audio distortions or extremely high audio volumes while exploring extrapolation possibilities.

The section H send messages to the python engine to handle output generation. "Generate & Play" initiates the python engine to synthesize a sound using the current interpolation curve and the audio selections. "Play Again" plays the previous generated sound, without going through the deep learning calculation. "STOP" immediately stops the audio coming out of the python engine. Phase iterations sets the number of iterations of the Fast Griffin-Lim algorithm. Higher number of iterations (max. 64) gives


Latent Timbre Synthesis
Interpolate Two
A)
by Kıvanç Tatar & Daniel Bisig & Philippe Pasquier
project info
Dataset
B) Drop Dataset Folder
C) Drop Run Folder
D:/my_workspace/dataset/erokia/
D:/my_workspace/dataset/erokia/func-timbre-vae-cedar/run-010/
Run
H)
Generate $ Play
Play Again
STOP
Normalize: off
phase iterations: 1
Save
Save Folder
C:/Users/root488/Desktop/Latent-timbre-synthesis-workshop/
I)
Python Audio
set previous audio files again
Get Available Audio Devices for Python Engine
Input
Output
Test Connection
File: 481276__erokia__m...
Zoom to selection
AUDIO 1
D)
Playback
START
Fill interpolation buffer-> fill sin 2 1
send interpolation curve to python
F)
Interpolation Curve
Output
AUDIO 2
E)
File: 478529__erokia__m...
Zoom to selection
G)
Duration: 1.15
Fill Duration by hand (s)-> 1.15

Reports from Python engine:
Number of phase estimation iterations set to 1

Fig. 5 The Max GUI of the *interpolate_two* application

better results; however, the calculation takes significantly longer. The phase estimation algorithm is the bottleneck of computational complexity of this framework. Still, the calculation of the audio takes 50% of the audio duration with phase iteration set to 1. That is, calculating a 2-second sound takes around 1-second on a laptop with NVIDIA RTX 2080 Max-Q GPU and 2.20 GHz Intel i7-8750H CPU.

In the following, we describe our study on incorporating the *interpolate_two* and Latent Timbre Synthesis to the composition frameworks of practitioners.

7 Evaluation

7.1 Participant composers

We invited nine composers and musicians to use Latent Timbre Synthesis in order to create a new musical work. During the recruitment process, we paid attention to bring together a group of participants that represented a wide diversity of personal interests, cultural backgrounds, creative approaches, and levels of expertise. The final group of participants consisted of four students and five established musicians, seven men and two women, members from eight different countries on four continents, three working in academic settings and six working in practical application domains. The backgrounds of these composers were diverse, covering Electroacoustic Music Composition, Computer Science, Performance Arts, Western New Music Composition, and Interactive Arts. The technical backgrounds of composers were also varying. Some composers were more familiar with textual coding languages and Unix shells, whereas other composers were familiar with visual programming languages such as Max and Pure Data.

We provided three datasets to composers so that they could immediately start experimenting with the LTS by using the *interpolate_two*. We named these datasets according to their contents, as *Electroacoustic*, *Western-Classical*, and *Sound-FX*. The *Electroacoustic* dataset included an album of a well-known electroacoustic composer, where the recordings used granular synthesis frequently, and we imagined that a diverse variety of grains in these recordings could help the training of the deep learning model in LTS. The total duration of recordings in this dataset is 48.34 minutes, including the additional 5 minutes of silence that we introduced to improve the training results. Using a CQT hop-size of 128 samples, this dataset included 895, 918 frames, i.e. the input vectors for the VAE training. Figure 6a, b visualizes the latent space generated by training an LTS model on this dataset. For this 2-dimensional visualization, we used the Barnes–Hut variation of the t-Distributed Stochastic Neighbouring algorithm [20]. Interestingly, we can observe the latent vectors of a sound object form a path in Fig. 6, which is an emergent outcome of the LTS.

For the second dataset, we used an album of Western Classical Music that had recordings of compositions by the Romantic Era composers. While including this second dataset, we were curious of the sound qualities of an LTS model trained on recordings of conventional musical instruments. The recordings in this dataset added up to a total duration of 73 minutes that resulted in 1,509,851 CQT frames for the training.

The third dataset was a sound pack of sound effects samples that were shared on freesound.org[10] with the Creative Commons Attribution 3.0 Licence. We also generated the examples that we provide in our supplementary materials using this dataset[11]. This dataset included 1140 audio samples that sums up to a total duration of 149.2 minutes of audio, which corresponded to 3, 084, 591 CQT frames for the VAE training. In our supplementary materials, we provide a 2-dimensional visualization of this dataset, where we generated the latent vectors of all audio frames using our trained model. In the visualization, we color-coded the latent vectors of individual audio files.

We provided composers pre-trained LTS models[11] along with these three datasets so that they could start experimenting with LTS using the *interpolate_two*. For all three datasets, we trained an LTS model using the architecture given in Fig. 2. We organized brief workshops to introduce LTS and *interpolate_two* to the composers, and we helped them through the installation of the framework on their computers.

We asked the composers to use LTS as a tool to produce a fixed-media composition of at least 1 minute in duration. They were free to further process the sounds that they generated by using the *interpolate_two*, and use any sound post-processing effects they preferred. We did not give composers any restriction other than the minimum time limit because we aimed to test the LTS within a diversity of composition practices. We notified composers that we could train new LTS models on a set of audio recordings that they provide. We also provided the computational resources to train LTS models on the recordings given by the composers.

7.2 Participant interviews

We invited nine composers to use *interpolate_two* application to create a composition with a minimum duration of 1 minute. Then, we asked these composers to fill in a questionnaire that consisted of the open-ended questions in Appendix C. The purpose of the questionnaire was to gain an understanding of how the composers integrated the *interpolate_two* into their musical workflow on both ideation and practical levels (questions 1–5), how they subjectively evaluated the aesthetic qualities of the sounds generated with the LTS (questions 6, 7, 8, 9, 11), the amount of mastery they achieved in tool usage (question 10), the positive and negative aspects in their experience of using the tool (question 12–14), and their opinion concerning suitable target groups for the LTS tools.

The following section summarizes the composers feedback provided through the interviews[12]

7.3 Qualitative study results

7.3.1 Iterative sound design in musical composition

All composers used the *interpolate_two* application within a diversity of composition frameworks. Previously, the iteration process of listening and composing has been referred as action-perception loop in Electroacoustic Music Composition [40]. The composers of our study carried out three distinct iteration processes within their composition practices. Composers 4, 6, and 7 used *interpolate_two* to generate a sound palette for their composition. These composers generated the sound samples before working on the temporal organization, arrangement, and post-processing of their composition using a Digital Audio Workstation (DAW). Hence, the iterations on sound design and sound generation were separate from the iteration of their composition. In the second iteration process, Composers 8 and 9 moved back forth between the *interpolate_two* and their DAW. As they felt of a need to design more sounds, they moved back to *interpolate_two* to generate more sounds. The third iteration process appeared in the composition process of Composer 2. This composer only used *interpolate_two* to produce the composition, and did not use any DAW for composition tasks. In conclusion, the *interpolate_two* as well as LTS fit within a variety of composition frameworks as a musical tool.

7.3.2 Musical strategies

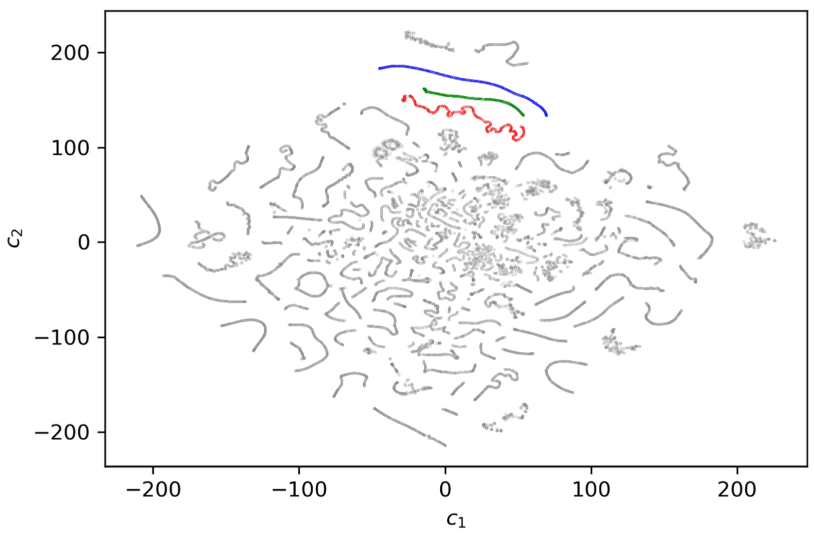
The composers pursued a variety of strategies to utilize LTS within their composition processes. Composer 1 approached the *interpolate_two* application with a specific goal of generating motives in composition through sound

[10] The samples are available to download at the following two links: https://freesound.org/people/Erokia/packs/26656/ and https://freesound.org/people/Erokia/packs/26994/.

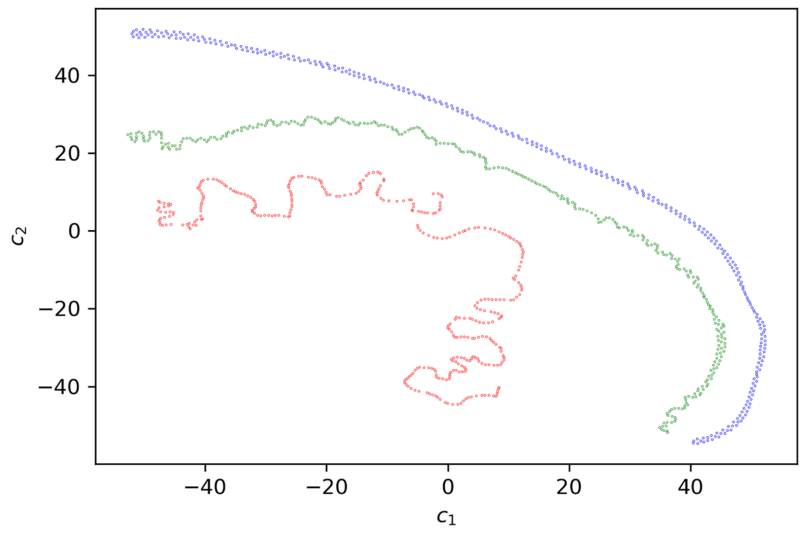
[11] Pre-trained models and example sounds are available at https://kivanctatar.com/latent-timbre-synthesis.

[12] The complete set of answers given by the composers are available at https://medienarchiv.zhdk.ch/entries/40dda1c8-6287-4356-adf4-ecdccec46119.

Fig. 6 Here are the 2-dimensional visualizations of the latent space generated by an LTS model trained on the *Electroacoustic* dataset, using t-Distributed stochastic neighbouring. In Fig. 6a, we used a portion of original recordings in the dataset to improve the clarity of the visualization



(a) The visualization of a portion of the *Electroacoustic* dataset, where each dot represents the latent vector generated by using an audio frame in the original recordings as the input to the encoder



(b) The audio frames located colored by red (top) and blue (bottom) depicts two audio excerpts from the dataset, where as the green (middle) audio frames are the generated latent vectors using a 50% interpolation amount.

design. This composer commented that the *interpolate_two* as an assisted-composition application allowed a “multi-modal approach to synthesis and composition.” Composer 1 found the wide range of sound output possibilities of LTS rather exhausting, whereas Composer 7 and 8 preferred exploring “timbral opportunities” through the “unique capabilities” of LTS.

Exploration and exploitation are two search strategies in Machine Learning. Exploration strategies covers global searches that allow relatively distant discoveries in the search space. In comparison, exploitation strategies take advantage of relatively local findings in the search space to find a local minima. Previous studies covered how exploration and exploitation functions in preset generation tasks in sound design [35]. In this study, we also observed that some composers described exploration and exploitation phases in their experiences using LTS. For example, Composers 4 and 5 used the audio excerpt selection and the interpolation possibilities in *interpolate_two* application to conduct the exploration search. Then, both of these two composers moved to exploitation search by fine-tuning the interpolation curve.

7.3.3 Musical goals and concepts

The composers initial intentions of using *interpolate_two* were also varied. Specifically, the composers came up with the following musical concepts by using our tools: the notion of cruising a sonic space, employing interpolation curves for creating temporal structure either in the form of imperfect sound loops or dramatic musical arcs, contrasting recognizable and non-recognizable sounds, exploring sound artifacts introduced by the tools. Composers 1, 2, 6, and 7 had a premade concept for their compositions in their mind, whereas Composers 4, 5, 8, and 9 developed their concept while using the tool. Composers 1, 2, 6, and 7 tried to anticipate the possibilities of LTS and derived their goals from this anticipation. Composers 4, 5, 8, and 9 preferred to get acquainted with the tool through extended experimentation and let these experiments guide them in their composition processes.

7.3.4 Familiarity

The acquaintance with the tool was another emerging concept in our study. Composer 8 specifically spent "most of the composition process" with getting familiar with the capabilities of the *interpolate_two* application and LTS. In contrast, Composer 6 could not anticipate the results of interpolation curves. Interestingly, Composer 7 experienced a similar unfamiliarity concerning the effects of parameter changes but found this unfamiliarity "engaging." In comparison, Composer 1 initially found the diverse sound output possibilities of LTS rather exhausting. After, Composer 1 decided to explore the affordances of LTS, which allowed this composer to fuse sound design and composition through the interpolation curve interface of *interpolate_two*. Likewise, Composer 9 commented that "Even during the creation of sounds with [Latent Timbre Synthesis], the composer is already composing."

7.3.5 Affordances

Composer 1 also described that the unique affordances of *interpolate_two* allowed following a new compositional approach. Similarly, Composer 5 expressed that the action-perception loop in the *interpolate_two* guided the composition concept, theme, and goals. Composers 4, 5, and 7 emphasized on the unique and diverse sound output possibilities of LTS. Composer 4 commented that these possibilities expanded the composition workspace, and Composer 7 described the process of "new sound generation" with *interpolate_two* as "engaging."

7.3.6 Sound aesthetics

We asked the composers if LTS and *interpolate_two* enabled unique aesthetic possibilities. Composer 1 defined the framework as a "self-contained tool" for "multimodal control." Composers 2, 3, 4, 5, 8, and 9 commented on the uniqueness of interpolation of two timbres using a curve. Similarly, the interpolation curve enabled the Composer 8 to connect easily with musical aspects of the composition such as phrasing and rhythm. Composer 5 specified that "...There's an inherent sense of working with the tool rather than the tool working for me that is very apparent." Composer 6 referred to the sound qualities of LTS as "distinct" and "abstract", whereas Composer 9 characterized the interaction with the sound as "tangible."

Although we provided help and computational resources to train new LTS models for the composers, only two composers returned back to us for training new LTS models using their datasets. Composer 2 prepared a dataset of voice recordings whereas Composer 6 gathered a dataset of national anthems. Composer 6 mentioned the similarity between the training dataset and the LTS generated sounds. Composer 7 noted the signature sound characteristic of LTS throughout all datasets. Composer 8 described that the relationship between the LTS sound outputs of *Electroacoustic* dataset and *Western Classical Music* dataset were loose. In brief, the composers noted the high dependency of LTS sound outputs to the training datasets, while recognizing the subtle sound characteristics of LTS that are shared across all datasets.

7.3.7 Sound quality

Regarding the sound quality of Latent Timbre Synthesis, there were a variety of responses from the composers. Composer 7 found the sound quality of LTS reliable and consistent with the recordings in the training dataset while Composer 4 commented the audio quality as "satisfactory." Composers 8 and 9 noted how the variety in sound possibilities depended on the dataset. Composers 1, 2, 3, and 5 noted the sound quality of LTS as "noisy". Composer 5 defined the noisiness aspect as "interesting and evocative" whereas Composer 6 emphasized how the sound quality depends on the interpolation curve. Composer 6 noted that "Dynamic interpolation curves resulted in richer sounds." Lastly, Composer 8 highlighted that the sound qualities of LTS were in line with Electroacoustic or Electronic Music.

Furthermore, Composer 3 pointed out that the noisiness was the most prominent when the interpolation amount was set to 50%, which is also the same amount that we use for the example in Fig. 6. We can observe in Fig. 6 that the red and blue sounds have distinct latent paths. A linear interpolation of 50% in between these distinct latent paths vaguely resembles both paths. We suspect that this linear interpolation approach is the main cause of the increased noisiness with the interpolation amounts around 50%. We mention future directions to address this issue in Sect. 8.

7.3.8 User interface

Additionally, the composers commented on the user interface (UI) of the *interpolate_two* application, illustrated in Fig. 5. Composer 4 mentioned that this UI was "simple and user friendly"; whereas Composer 3 and 5 highlighted the clarity of the layout. Composer 3 went further to understand the source Max patch of the *interpolate_two*, and found the layout of the source patch confusing. Composer 2 referred to the overall UI as "intuitive." However, Composer 2 was frustrated with the impreciseness of the interpolation curve UI.

7.3.9 Tool deficiencies

We asked the composers what were their frustrations when working with the LTS through the *interpolate_two*. Composer 1 guessed that the infamous "blurry" outputs generated by Variational Autoencoders hampers the full potential of timbre interpolation. Composers 3 and 4 were frustrated that the sound generation was not in real-time. Composers 8 and 9 asked for future versions to include stereo output generation. Composer 5 wished that the training of new LTS models were easier. Currently, the VAE model training on the *Electroacoustic* dataset takes around 6 hours on a high-end GPU. In comparison, the *interpolate_two* application takes approximately 50% of the audio duration to calculate a sound using a trained VAE model on a latest computer, and the application does not necessarily need a GPU. That is, the application takes 1 second to generate a sound with 2 second duration.

7.3.10 Continued use

Lastly, all composers noted that they would continue using LTS and *interpolate_two*. We think that this result indicates the successful incorporation of the Latent Timbre Synthesis within a diversity of composition frameworks through the *interpolate_two* application. All composers of this study used their own computers to work with the *interpolate_two* and the LTS models, which indicated that the computational resources required to run the framework were manageable across various consumer level computers.

8 Conclusions and future work

We introduced the Latent Timbre Synthesis (LTS), a flexible audio synthesis framework that can change the sound output possibilities using a training dataset of audio recordings. We explained the three main parts of the LTS framework, Constant-Q Transform calculation, latent audio frame space generation with deep learning, and inverse synthesis using magnitude CQT spectrograms and Griffin-Lim phase estimation. We focused on Variational Autoencoders to generate the latent space, which later allowed new synthesis possibilities through interpolation and extrapolation of latent vectors of two audio excerpts. We gave the details of *interpolate_two*, an application that enables composers of all levels to incorporate the LTS within their composition practices.

The evaluation study pointed out that the overall experiences of composers interacting with the Latent Timbre Synthesis through *interpolate_two* were positive. The comments of the composers directed us towards possible future research directions. The composers commented on the increased noisiness of generated sounds using the linear interpolation between latent vectors. We aim to address this issue by exploring different interpolation techniques that takes the latent paths into account. The visualizations of the latent audio frame space generated by trained LTS models indicated that sound objects created distinct paths in the latent space generated by VAE models. As future research, we would like to explore how to incorporate these latent paths to the interpolation and extrapolation of timbres, beyond linear interpolation. One possible option would be to experiment with integration of discrete-time derivatives of latent vectors into the interpolation calculation. We could also further improve the sound quality of the LTS by adding an additional training step where a new decoder architecture is trained. In this additional training step, we can exchange the decoder of the Variational Autoencoder with a new decoder architecture that generates the raw audio using the latent vectors generated by the VAE encoder. This would also eliminate the requirement of Griffin-Lim phase estimation, which is the main bottleneck of LTS's computational complexity. In the literature of speech synthesis, the DL architectures for vocoder applications seem promising to improve the decoder of the current architecture in the Latent Timbre Synthesis.

Acknowledgements This research has been supported by the Swiss National Science Foundation, Natural Sciences and Engineering Research Council of Canada, Social Sciences and Humanities Research Council of Canada, and Compute Canada.

Compliance with ethical standards

Conflict of interest There is no potential conflict of interest related to this work within our knowledge.

Appendix A Constant-Q transform

We can calculate the CQT of an audio recording [31], a discrete time domain signal $x(n)$, using the following formula:

$$X^{CQ}(k,n) = \sum_{j=n-\lfloor N_k/2 \rfloor}^{n+\lfloor N_k/2 \rfloor} x(j) a_k^*(j - n + N_k/2) \tag{2}$$

where k represents the CQT frequency bins with a range of [1, K], and $X^{CQ}(k,n)$ is the CQT transform. N_k is the window length of a CQT bin, that is inversely proportional to f_k that we define in Eq. 4 Notice that, $\lfloor \cdot \rfloor$ is the rounding towards negative infinity. a_k^* is the negative conjugate of the basis function $a_k(n)$ and,

$$a_k(n) = \frac{1}{N_k} w\left(\frac{n}{N_k}\right) exp\left[-i2\pi n \frac{f_k}{f_s}\right] \tag{3}$$

where $w(t)$ is the window function, f_k is the center frequency of bin k, and f_s is the sampling rate. CQT requires a fundamental frequency parameter f_1, which is the center frequency of the lowest bin. The center frequencies of remaining bins are calculated using,

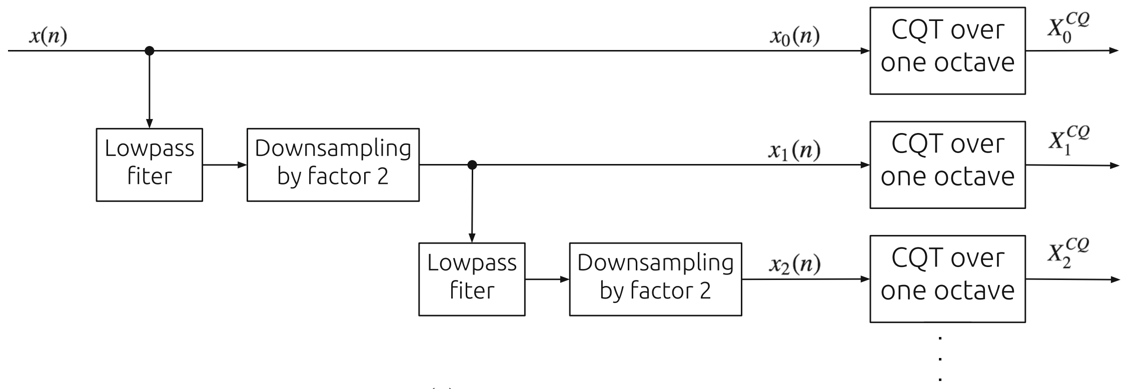
$$f_k = f_1 2^{\frac{k-1}{B}} \tag{4}$$

where B is the number of bins per octave.

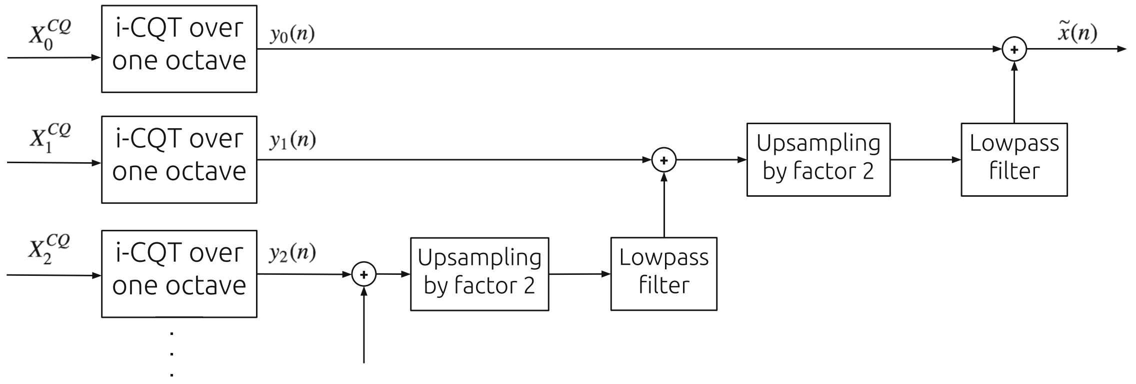
CQT is a wavelet-based transform because the window size is inversely proportional to the f_k while ensuring the same Q-factor for all bins k. We can calculate the Q-factor using,

$$Q = \frac{q f_s}{f_k (2^{\frac{1}{B}} - 1)} \tag{5}$$

where q is scaling factor with the range [0,1] and equals to 1 as the default setting. We direct our readers to the original publication for the specific details of the CQT [31], which also proposed a fast algorithm to compute CQT and inverse CQT (i-CQT), given in Fig. 7.



(a) Constant-Q Transform



(b) Inverse Constant-Q Transform

Fig. 7 A fast algorithm to compute CQT and i-CQT, described in [31] and implemented in Librosa [22]

Appendix B Phase estimation algorithms

Given an audio signal $x(n)$ and its frequency transform $X(i)$,

Algorithm 1 Griffin-Lim Algorithm

```
1: Set: $\angle X_0(i)$
2: Initialize: $X_0(i) = |X(i)| \cdot e^{j\angle X_0(i)}$
3: for $n = 1, 2, \ldots, N$ do
4:     $X_n(i) = T(IT(|X(i)| \cdot e^{j\angle X_{n-1}(i)}))$
5: end for
6: $\hat{x}(n) = IT(X_N(i))$
```

where N is the total number of GLA iterations, T and IT is the frequency transform and inverse frequency transform function respectively; such as Short-Fourier Transform, or Constant-Q Transform in our case. Note that, the space of audio spectrograms is a subset of the complex number space. The iterative process of Griffin-Lim moves the complex spectrogram of the estimated signal $\hat{x}(n)$ towards the complex number space of audio signals in each iteration, as proven in [9].

Algorithm 2 Fast Griffin-Lim Algorithm

```
1: Set: $\angle X_0(i)$
2: Initialize: $X_0(i) = |X(i)| \cdot e^{j\angle X_0(i)}$
3: Initialize: $Y_0(i) = T(IT(|X(i)| \cdot e^{j\angle X_0(i)}))$
4: for $n = 1, 2, \ldots, N$ do
5:     $Y_n(i) = T(IT(|X(i)| \cdot e^{j\angle X_{n-1}(i)}))$
6:     $X_n(i) = Y_n(i) + \alpha(Y_n(i) - Y_{n-1}(i))$
7: end for
8: $\hat{x}(n) = IT(X_N(i))$
```

The Fast Griffin-Lim algorithm (F-GLA) is a revision of the original Griffin-Lim algorithm. A previous study [27] showed that the F-GLA revision significantly improves signal-to-noise ratio (SNR) compared to the GLA, where the setting $\alpha = 1$ (a constant in algorithm 2) resulted in the highest SNR value.

Appendix C Interview questions

1. Describe your compositional process when working with the Timbre Space tools
2. What was the theme and concept of your composition?
3. How did you incorporate the Timbre Space tools into your work?
4. How did working with the Timbre Space tools change your composition workflow? What was unique?
5. What additional tools/technologies apart from the Timbre Space tools were involved in your work?
6. How would you describe the sound qualities of Timbre Space?
7. What were the unique aesthetic possibilities of the Timbre Space tools?
8. What kind of dataset(s) did you train Timbre Space with?
9. If you trained Timbre Space with several datasets, what kind of relationship did you notice between the datasets and the musical results obtained from Timbre Space?
10. Did you feel control, and authorship over the musical material generated?
11. Did you achieve the aesthetic result you intended?
12. What were the positive aspects when working with the tool?
13. What were the frustrations when working with the tool? How can it be Improved?
14. Would you use it again (if the above were addressed)?
15. For whom else or what musical genres/sectors would this tool be particularly useful (if the criticism was addressed)?

References

1. Akten M (2018) Grannma MagNet. https://www.memo.tv/works/grannma-magnet/. Library Catalog: www.memo.tv
2. Briot JP, Pachet F (2020) Deep learning for music generation: challenges and directions. Neural Computing and Applications 32(4):981–993. https://doi.org/10.1007/s00521-018-3813-6
3. Dieleman S Sander Dieleman: Generating music in the raw audio domain. https://www.youtube.com/watch?v=y8mOZSJA7Bc
4. Dieleman S, Oord Avd, Simonyan K (2018) The challenge of realistic music generation: modelling raw audio at scale. In: Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), p. 11. Montreal QC, Canada
5. Engel J, Hantrakul LH, Gu C, Roberts A (2020) Ddsp: Differentiable digital signal processing. In: International Conference on Learning Representations. https://openreview.net/forum?id=B1x1ma4tDr
6. Esling P, Chemla-Romeu-Santos A, Bitton A (2018) Generative timbre spaces: regularizing variational auto-encoders with perceptual metrics. arXiv:1805.08501 [cs, eess]. http://arxiv.org/abs/1805.08501. ArXiv: 1805.08501
7. Gabor D (1947) Acoustical Quanta and the Theory of Hearing. Nature 159(4044):591–594. https://doi.org/10.1038/159591a0
8. Grey JM (1977) Multidimensional perceptual scaling of musical timbres. The Journal of the Acoustical Society of America 61(5):1270–1277. 10.1121/1.381428. https://doi.org/10.1121/1.381428
9. Griffin DW, Lim JS (1984) Signal estimation from modified short-time Fourier transform. IEEE Transactions on Acoustics, Speech, and Signal Processing 32(2):236–243. https://doi.org/10.1109/TASSP.1984.1164317
10. Hantrakul L, Engel J, Roberts A, Gu C (2019) Fast and Flexible Neural Audio Synthesis. In: Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR 2019), p. 7

11. He K, Zhang X, Ren S, Sun J (2016) Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. IEEE, Las Vegas, NV, USA. https://doi.org/10.1109/CVPR.2016.90
12. Iverson P, Krumhansl CL (1993) Isolating the dynamic attributes of musical timbrea. The Journal of the Acoustical Society of America **94**(5), 2595–2603. Publisher: Acoustical Society of America
13. Kingma DP, Welling M (2014) Auto-Encoding Variational Bayes. arXiv:1312.6114 [cs, stat] . http://arxiv.org/abs/1312.6114. ArXiv: 1312.6114
14. Kingma DP, Welling M (2019) An Introduction to Variational Autoencoders. Foundations and Trends in Machine Learning **12**(4), 307–392. http://arxiv.org/abs/1906.02691. ArXiv: 1906.02691
15. Krumhansl CL (1989) Why is musical timbre so hard to understand. Structure and perception of electroacoustic sound and music 9:43–53
16. Kumar K, Kumar R, de Boissiere T, Gestin L, Teoh WZ, Sotelo J, de Brebisson A, Bengio Y, Courville A (2019) MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis. In: Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), p. 12. Vancouver, BC, Canada
17. Lakatos S (2000) A common perceptual space for harmonic and percussive timbres. Perception & psychophysics **62**(7), 1426–1439. Publisher: Springer
18. LeCun Y, Cortes C, Burges C MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/
19. Luigi R (1967) The Art of Noise. A Great Bear Pamphlet
20. Maaten Lvd (2014) Accelerating t-sne using tree-based algorithms. Journal of machine learning research 15(1):3221–3245
21. McAdams S, Winsberg S, Donnadieu S, De Soete G, Krimphoff J (1995) Perceptual scaling of synthesized musical timbres: Common dimensions, specificities, and latent subject classes. Psychological research **58**(3), 177–192. Publisher: Springer
22. McFee B, Raffel C, Liang D, Ellis DP, McVicar M, Battenberg E, Nieto O (2015) librosa: Audio and Music Signal Analysis in Python. In: Proceedings of The 14th Python in Science Conference (SCIPY 2015)
23. Müller M (2015) Fundamentals of Music Processing. Springer International Publishing, Cham . https://doi.org/10.1007/978-3-319-21945-5
24. Nieto O, Bello JP (2016) Systematic Exploration Of Computational Music Structure Research. In: Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR 2016), p. 7. New York, NY, USA
25. Oord Avd, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K (2016) Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499
26. Oord Avd, Li Y, Babuschkin I, Simonyan K, Vinyals O, Kavukcuoglu K, Driessche Gvd, Lockhart E, Cobo LC, Stimberg F, Casagrande N, Grewe D, Noury S, Dieleman S, Elsen E, Kalchbrenner N, Zen H, Graves A, King H, Walters T, Belov D, Hassabis D (2017) Parallel WaveNet: Fast High-Fidelity Speech Synthesis. arXiv:1711.10433 [cs]. http://arxiv.org/abs/1711.10433. ArXiv: 1711.10433
27. Perraudin N, Balazs P, Sondergaard PL (2013) A fast Griffin-Lim algorithm. In: 2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, pp. 1–4. IEEE, New Paltz, NY. 10.1109/WASPAA.2013.6701851. http://ieeexplore.ieee.org/document/6701851/
28. Roads C (2004) Microsound. The MIT Press, Cambridge, Mass
29. Roads C (2015) Composing electronic music: a new aesthetic. Oxford University Press, Oxford
30. Schaeffer P (1964) Traité des objets musicaux, nouv. edn. Seuil
31. Schörkhuber C, Klapuri A (2010) Constant-Q Transform Toolbox For Music Processing. In: Proceedings of the 7th Sound and Music Computing Conference (SMC 2010), p. 8. Barcelona, Spain
32. Smalley D (1997) Spectromorphology: explaining sound-shapes. Organised Sound 2(02):107–126. 10.1017/S1355771897009059. http://journals.cambridge.org/article_S1355771897009059
33. Stockhausen K (1972) Four Criteria of Electronic Music with Examples from Kontakte . https://www.youtube.com/watch?v=7xyGtI7KKIY&list=PLRBdTyZ76lvAFOtZvocPjpRVTL6htJzoP
34. Sønderby CK, Raiko T, Maaløe L, Sønderby SK, Winther O (2016) How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks. In: Proceedings of the 23rd international conference on Machine learning (ICML 2016). ACM Press, Pittsburgh, Pennsylvania
35. Tatar K, Macret M, Pasquier P (2016) Automatic Synthesizer Preset Generation with PresetGen. Journal of New Music Research 45(2):124–144. https://doi.org/10.1080/09298215.2016.1175481
36. Tatar K, Pasquier P (2017) MASOM: A Musical Agent Architecture based on Self Organizing Maps, Affective Computing, and Variable Markov Models. In: Proceedings of the 5th International Workshop on Musical Metacreation (MUME 2017). Atlanta, Georgia, USA
37. Tatar K, Pasquier P (2019) Musical agents: A typology and state of the art towards Musical Metacreation. Journal of New Music Research 48(1):56–105. https://doi.org/10.1080/09298215.2018.1511736
38. Tatar K, Pasquier P, Siu R (2019) Audio-based Musical Artificial Intelligence and Audio-Reactive Visual Agents in Revive. In: Proceedings of the joint International Computer Music Conference and New York City Electroacoustic Music Festival 2019 (ICMC-NYCEMF 2019), p. 8. International Computer Music Association, New York City, NY, USA
39. Technavio: Global Music Synthesizers Market 2019-2023. https://www.technavio.com/report/global-music-synthesizers-market-industry-analysis
40. Vaggione H (2001) Some ontological remarks about music composition processes. Computer Music Journal 25(1):54–61
41. Varese E, Wen-chung C (1966) The liberation of Sound. Perspectives of New Music **5**(1), 11–19 . https://www.jstor.org/stable/832385?origin=JSTOR-pdf&seq=1#page_scan_tab_contents
42. Velasco GA, Holighaus N, Dörfler M, Grill T (2011) Constructing An Invertible Constant-Q Transform With Nonstationary Gabor Frames. In: Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)), p. 7. Paris, France
43. Yu F, Koltun V (2015) Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122