From Words to Sound: Neural Audio Synthesis of Guitar Sounds with Timbral Descriptors

The Sound of AI Community*
valerio@thesoundofai.com

Abstract

Interest in neural audio synthesis has been growing lately both in academia and industry. Deep Learning (DL) synthesisers enable musicians to generate fresh, often completely unconventional sounds. However, most of these applications present a drawback. It is difficult for musicians to generate sounds which reflect the timbral properties they have in mind, because of the nature of the latent spaces of such systems. These spaces generally have large dimensionality and cannot easily be mapped to semantically meaningful timbral properties. Navigation of such timbral spaces is therefore impractical. In this paper, we introduce a DL-powered instrument that generates guitar sounds from vocal commands. The system analyses vocal instructions to extract timbral descriptors which condition the sound generation.

1 Introduction

Research in the field of sound synthesis is providing musicians with a growing pool of commercial and open-source options to synthesise audio. These tools may employ methods such as physical modeling, acoustic modeling, sample based synthesis, etc. (Russ [2012]). In physically modelled synthesis, an algorithm is developed that matches the timbre and amplitude envelope of a real-world instrument, and provides parameters to mimic how a musician's interactions with the instrument can vary, such as the location that a mallet strikes on a drum (Smith [1996]). The interfaces of physically modelled instruments are arguably easy to use because of the higher likelihood that control parameters may be meaningful. Alternatively, common analog or digital synthesisers acoustically model a waveform by using one of a number of methods such as additive, subtractive, frequency modulation, etc. synthesis. The complex timbral patterns which can be generated with these methods are made possible through fine-grained control points in the form of low-frequency oscillators, envelopes, filters, etc. (Russ [2012]).

As the sound generation method departs from strict physically modelled synthesis, arguably the semantic understanding of what timbral qualities a parameter changes diminishes. This is an important point to consider for musicians making the trade-off between ease of use and flexibility of sound design possibilities, and opens up opportunities to develop audio synthesis tools that afford both semantic understanding for ease of use and generation of rich and varied waveforms. A growing area of research with potential to meet both of these specifications is neural audio synthesis, where a dataset of sound examples is used to train deep learning (DL) models, which then synthesise either raw audio or time-frequency domain representations (e.g., Engel et al. [2017, 2019], Tatar et al. [2021]). A common limitation of such methods is not providing an easy-to-use control to the user to navigate timbral properties in a way that is user friendly, and meaningful to the user.

In this paper, we try to address this issue by presenting a neural synthesiser that generates guitar sounds based on semantic descriptors. We package the neural synthesiser in a sampler instrument. To facilitate interaction between the user and the instrument, we implement a voice-controlled interface. Users can request guitar samples with specific timbral qualities by uttering simple vocal commands

*The Sound of AI Community collaboratively carried out this research project following open science methodologies. The Authors section at the end of the paper lists community members who contributed and their roles.

(e.g., "give me a thick distorted guitar sound"). They can then refine the sample adjusting semantically meaningful timbre sliders, which condition the latent space of the neural synthesiser. We decided to limit the synthesis to guitar sounds to keep the project manageable and because of dataset constraints. However, the technology presented here can be extended to the generation of other instruments.

The remainder of the paper is organised as follows. In Section 2, we review current state-of-the-art DL-based audio synthesis methods. Then, we introduce the proposed instrument and describe its components (Section 3). In Section 4, we discuss an expert-based evaluation of the system. Finally, we provide conclusions and illustrate possible strategies to improve the instrument (Section 5).

2 Related works

Neural audio synthesis is usually performed with Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and WaveNet-like autoregressive models. Tatar et al. [2021] propose a VAE to synthesise audio, where the latent space is used to navigate the timbral properties learned from training data. The model is trained on CQT-Spectrograms of audio examples. Researchers used a latent space with 256 dimensions to retain sound quality. Reduction in the number of latent dimensions resulted in poorer reconstruction of the audio with large noise floor. While this approach works for exploring the timbral space between two audio samples, the latent space is too large to navigate conveniently. Also, the model does not provide independent control for pitch. VaPar Synth (Subramani et al. [2020]) addresses this drawback, by using a Conditional VAE which provides parametric control for pitch and spectral envelope on a frame-by-frame basis. Despite the control afforded to model the spectral envelope at each frame, it is not clear what each coefficient of the latent space represents. Engel et al. [2017] present a WaveNet-style autoregressive model for generating raw audio. Unlike the original WaveNet (Van Den Oord et al. [2016]), the proposed architecture does not depend on external conditioning for long-term dependencies. A major limitation of this system is the time complexity to synthesise audio. This may limit WaveNet-inspired models for semi real-time synthesis on many personal computers. GANSynth (Engel et al. [2019]) uses a GAN architecture for audio synthesis. It generates log-magnitude spectrograms and phase representation. This approach produces more coherent waveforms than those generated directly from raw audio. While global conditioning can be applied, there is little likelihood of the latent dimensions corresponding to semantic features. DDSP (Engel et al. [2020]) takes a different approach to audio synthesis. Instead of generating either raw audio or a spectrogram representation, the DDSP decoder generates inputs for an additive synthesiser, a subtractive synthesiser and, optionally, a reverb. The latent space can be used to navigate timbre. However, its dimensionality is large and therefore makes timbral exploration impractical. As noted for other approaches above, DDSP's latent dimensions are not semantically meaningful. This is a recurring issue that hinders the usability of DL-based synthesisers. In the next section, we introduce our solution, which aims to address this problem.

3 The instrument

To meet our goal of a timbre-driven AI-powered guitar synthesiser, we developed a sampler instrument with a DL backend which takes voice commands as input to neurally synthesise guitar audio samples. This end-to-end system comprises four main components: *Speech Recognition* (SR), *Descriptors Recognition* (DR), *Sound Generation* (SG), and *Sampler*.

We developed all the components of the system in Python and packaged them into a single desktop application. The application can be used across operating systems. It provides a user interface (Figure 1) through which the user can interact with the various components and perform with the Sampler. Requirements for launching the system can be found on the project's GitHub repository along with the system's source code.[2] We provide tutorial videos for Windows and MacOS.[3] We also provide a paper companion website where we share example guitar sounds generated with the instrument.[4]

[2]The source code of the system is available at `https://github.com/TheSoundOfAIOSR/project_common/`.

[3]Tutorial for Windows: `https://www.youtube.com/watch?v=wGsGGvbZByE`. Tutorial for MacOS: `https://www.youtube.com/watch?v=pppuC27xRGo`.

[4]Examples of generated guitar sounds can be found at `https://thesoundofaiosr.github.io/Paper_Companion/`.

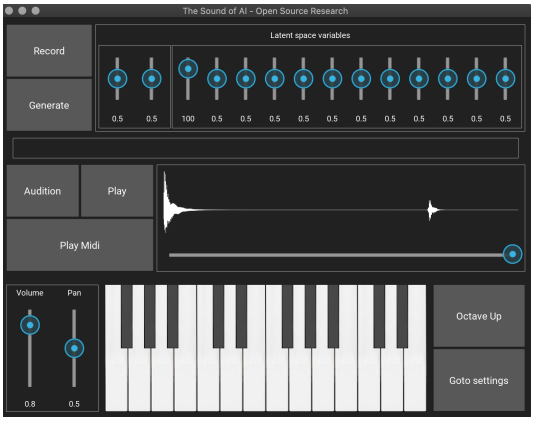

Figure 1: Instrument's GUI.

To generate a guitar sample, the system follows a multi-step process (Figure 2):

1. The user records a vocal command inside of the Sampler interface (e.g., "thin guitar sound").
2. SR analyses the user's voice and extracts the corresponding text (Section 3.1).
3. DR generates a *descriptors embedding* from the text (Section 3.2), which captures the timbral qualities requested by the user.
4. The descriptors embedding is fed to SG and conditions the synthesis of a guitar sample inside a pre-trained latent space (Section 3.3).
5. The generated audio sample is returned to the user and can be played back using the Sampler (Section 3.4).
6. The user can refine the sound adjusting timbre sliders. Sliders correspond to semantically-meaningful latent space parameters of the SG component (e.g., inharmonicity, attack).

The user can directly insert text commands in the Sampler, and shortcut SR. In the remainder of this section, we describe in detail each of the four components which make up the instrument.

3.1 Speech recognition component

The Speech Recognition component allows users to interact with the application through their voice. Rather than modifying sound through control knobs, buttons, and configurations – traditionally used in software music instruments – users can issue simple vocal commands to influence sound generation.

Since effective open-source end-to-end ASR (Automatic Speech Recognition) systems are available, we decided to integrate state-of-the-art ASR technologies rather than building a solution from scratch. The selection criterium for ASR technologies was a combination of accuracy, computational efficiency, and code availability. We selected two ASR models: *wav2vec2* (Baevski et al. [2020]) and *QuartzNet* (Kriman et al. [2020]). wav2vec2 uses representation learning (Bengio et al. [2013]) to learn representations from speech data that are then used to recognise speech. The model employs self-supervised learning to learn speech representations from unlabelled data, and is fine-tuned with a labelled dataset. wav2vec2 achieves state-of-the-art performance with a limited amount of labelled data (Baevski et al. [2020]). Alternatively, we can use QuartzNet. This model achieves state-of-the-art performance (Kriman et al. [2020]), despite its relatively small model parameter size. The two chosen architectures are deployed into a framework designed to choose which model to utilise at runtime. This modular approach enables further extensions, with the option to integrate additional ASR models in the future.

3.2 Descriptors recognition component

Descriptors Recognition is a Natural Language Processing unit that processes the free text incoming from SR and outputs a *descriptors embedding*. Such embedding is a mathematical representation of

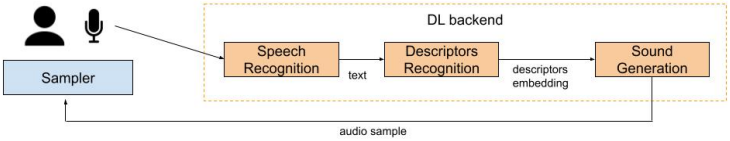

Figure 2: Generation of guitar audio samples.

the timbral qualities featured in the input sentence. For example, a user may ask the application to "generate a bright and strong guitar sound". DR is expected to identify the timbral qualities ("bright" and "strong"), and map them to a subset of 18 standardised timbre descriptors such as "full", "thick", and "hard". The standardised descriptors make up a Guitar Sound Taxonomy we have developed from user surveys and academic literature. The descriptors are organised in 9 pairs of opposites (e.g., clear vs muddy, bright vs dark). These descriptors can be used to condition the Sound Generation component. A detailed account of the taxonomy and how we have derived it can be found in Appendix A.[5]

DR consists of two subcomponents, i.e., a Named Entity Recognition (NER) and a Word to Word Matcher (WWM). To produce a descriptors embedding, DR carries out two steps:

1. NER identifies timbral qualities (i.e., free adjectives that describe sound) present in the input sentence (Section 3.2.1).
2. WWM maps the free sound qualities to the 18 timbral descriptors of the Guitar Sound Taxonomy, thus generating a descriptors embedding (Section 3.2.2).

3.2.1 Named entity recognition

We trained the NER subcomponent using the NER facilities offered by *spaCy* version 3, a Python-based production-ready Natural Language Processing library (Vasiliev [2020]). Specifically, we used the spaCy NER model built on top of the pre-trained RoBERTa base model provided by HuggingFace (Liu et al. [2019]). The goal was to train a model capable of identifying the words in a sentence that indicate timbral qualities of an instrument (e.g., "dark", "distorted"). We trained the NER model on a dataset consisting of 516 sentences labelled with sound qualities. We created the dataset by scraping threads focused on timbral aspects of guitar and other instruments in music-related subreddits. We labelled the sentences using active learning to optimise the labelling process (Shen et al. [2017]). For each sentence, we flagged the words (mainly adjectives) providing timbral information. By the end of the labelling step, we identified 371 distinct timbral qualities. We release the dataset in the public.[6] On the test set, the trained model reached an F-score of 0.92 for sound qualities it had seen during training, and 0.81 for those unknown to the model.

3.2.2 Word to word matcher

WWM maps the free timbral qualities identified by NER to a subset of the 18 standardised descriptors of the Guitar Sound Taxonomy. We wrap this subset into a descriptors embedding. This is an 18-dimensional binary array. Each item in the array represents a standardised descriptor. 0s indicate that the associated descriptors are missing in the input sentence. 1s indicate that they are present.

The mapping is performed by calculating the distance between vector representations of the free sound qualities and the standardised descriptors. We use spaCy's *en_core_web_lg* language model to derive the vector representations (i.e., word embeddings) for all the words involved in the process.

Since the 18 descriptors come in 9 pairs of opposites, we should ensure that at most only one of the opposite descriptors is selected per pair. To fulfill this constrain, the algorithm applies the following two-stage approach:

[5] Appendices can be accessed at `https://drive.google.com/drive/folders/13dnTNT8lN5HKxzOsDcWU9JKvA39ipFrY?usp=sharing`.

[6] The dataset can be downloaded at `https://github.com/TheSoundOfAIOSR/rg_dataset/blob/main/reddit_data_preprocessing/data/curated_data.csv`.

1. Every sound quality is associated with the closest descriptor and its opposite (i.e., a pair). The distance between sound qualities and descriptors is computed using K-Means clustering with cosine distance on the word embeddings.
2. From each pair of opposites with at least one associated sound quality, a single descriptor is chosen. We calculate the difference of the distances between a sound quality and the two opposite descriptors of a pair. Such difference is computed and summed over all qualities associated with that pair. The difference is positive if the summed distance of associated qualities to a descriptor is smaller than the summed distance of the opposite descriptor in the pair. If this value is larger than a specified threshold, then the descriptor is assigned a "1" in the descriptors embedding. The threshold has been empirically set to 10^{-6}. This process is repeated for all 9 pairs to determine all the items of the descriptors embedding.

The descriptors embedding produced by DR is fed to the Sound Generation component.

3.3 Sound generation component

SG syntheses one-shot guitar audio samples with a duration of four seconds. To generate high-fidelity sound, SG uses a novel autoencoder architecture we call Timbre Conditioned Autoencoder (Section 3.3.4), trained with Quasi-Harmonic audio representations of guitar sounds (Section 3.3.1). The autoencoder is conditioned on a set of semantic parameters commonly employed to describe timbre of guitar sounds (e.g., inharmonicity, attack time) that we call *timbre measures* (Section 3.3.3). A detailed mathematical treatment of SG could not be included in this paper for space constraints, however it is presented in Appendix B[5].

3.3.1 Audio representation

Rather than using raw audio or Fourier coefficients to represent audio, we use parametric models which are physically and perceptually motivated. In particular, we consider models which decompose audio into two components (Serra et al. [1997]): i) a deterministic part, usually modelled as a sum of frequency and amplitude-modulated sinusoidal components; ii) a stochastic part, usually modelled as frequency-modulated Gaussian noise.

These audio representations are easy to manipulate before re-synthesis and can be used as a basis to conveniently extract timbre measures (Section 3.3.3). Currently, we ignore the stochastic part and use only the deterministic component.

We employ a Quasi-Harmonic Model (QHM) to represent audio (Pantazis et al. [2008]). The model parameters are $m_k,\ f_k,\ \varphi_k$, respectively the instantaneous magnitudes, frequencies, and phases for each harmonic frequency at each time frame. We perform a prior identification of the QHM parameters, which allows us to formulate the loss directly on these perceptually-relevant parameters (Section 3.3.5), without having to synthesise audio.

The QHM representation is also useful to calculate timbre measures (Section 3.3.3), which would have been difficult to compute directly from raw audio or spectrograms.

For the analysis and synthesis of QHM, we developed the *tsms* library in TensorFlow.[7] tsms provides an algorithm that identifies QHM parameters based on peak picking over the Short-Time Fourier Transform and an iterative refinement strategy.

3.3.2 Transformation of Quasi-Harmonic Model Parameters

The QHM parameters identified with tsms are not in a form that can be easily handled by a neural network, for encoding / decoding purposes. To address the problem, we apply reversible transformations on $m_k,\ f_k,\ \varphi_k$ to obtain a new set of parameters $m_{env},\ m_k^{dist},\ f_0^{shift},\ f_k^{shift},\ \varphi_k^{diff}$, which are normalized zero mean quantities. The transformed QHM parameters are the inputs-outputs of the Timbre Conditioned Autoencoder (Section 3.3.4). They are converted back to the original parameters when synthesising audio. Details about the transformations can be found in Appendix B[5].

[7] Source code for the *tsms* library is available at `https://github.com/fabiodimarco/tf-spectral-modeling-synthesis`.

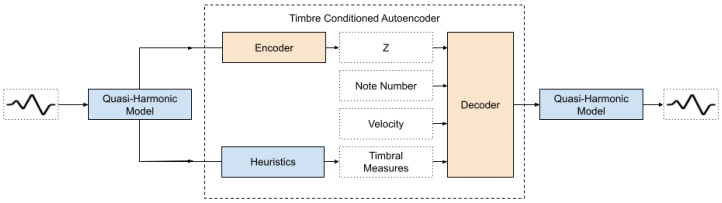

Figure 3: Sound Generation component. Orange elements are trainable.

3.3.3 Timbre measures

Timbre measures computed using $m_k,\ f_k,\ \varphi_k$, quantify timbral characteristics of guitar sounds. They indicate how present a certain timbral feature is in a sound. They can take continuous values in the interval 0 (no presence), 1 (max presence). Timbre measures are used to condition the autoencoder and allow users to navigate the latent space in a meaningful way. We compute the timbre measures on the non-transformed QHM parameters using heuristics. We derived 11 measures inspired by Peeters et al. [2011]: *inharmonicity*, *even_odd*, *sparse_rich*, *attack_rms*, *decay_rms*, *attack_time*, *decay_time*, *bass*, *mid*, *high_mid*, *high*. Details about the measures can be found in Appendix B[5].

3.3.4 Timbre Conditioned Autoencoder

The architecture of the Timbre Conditioned Autoencoder is composed of 3 main computational blocks (Figure 3): Heuristics, Encoder, and Decoder. The Heuristics block is not trainable. It takes $m_k,\ f_k, \varphi_k$ as inputs, and computes the 11 timbre measures introduced in Section 3.3.3. The Encoder block is a trainable CNN architecture which takes as inputs: $m_{env},\ m_k^{dist},\ f_0^{shift},\ f_k^{shift},\ \varphi_k^{diff}$. It outputs a two-dimensional latent space Z. The values of Z are constrained between 0 and 1, by using sigmoid activation. The Decoder block is a trainable CNN architecture which takes as inputs the quantities computed by the previous blocks (i.e., 11 timbral measures, Z) plus the MIDI note number and velocity of the current input sample, for a total of 15 parameters. These are all semantically labelled with the exception of the two-dimensional latent inputs Z. The use of a compact latent space with semantically meaningful dimensions makes it easy for the user to control generation. The outputs of the Decoder are the transformed QHM parameters $m_{env},\ m_k^{dist},\ f_0^{shift},\ f_k^{shift}$ except for φ_k^{diff}, that is not used to synthesise the final audio, since the phase φ_k it is reconstructed via frequency integration.

3.3.5 Model training

We trained the autoencoder on a subset of the NSynth dataset (Engel et al. [2017]), restricting the samples to just guitar sounds. We split the dataset into three sets for training, validation, and testing. We performed offline pre-computation of the QHM parameters. The inputs of the Encoder, and outputs of the Decoder are expected to be the same, i.e., the transformed QHM parameters, except for φ_k^{diff}. To train the model, we use a composed loss computed on the transformed QHM parameters. The components of the loss are $L_{m_{env}},\ L_{m_k^{dist}},\ L_{f_0^{shift}},\ L_{f_k^{shift}}$. Details about the losses can be found in Appendix B[5].

3.3.6 Model inference

Before a sample is generated for the first time, SG receives a descriptors embedding from DR. The embedding is mapped to timbre measures using a rule-based approach. While we provide a default mapping between descriptors embedding and timbre measures, users can create custom mappings which transform words to sound in the way that suits them best. Values for Z are generated randomly, whereas MIDI note number and velocity are specified by the Sampler. Timbre measures, Z values, MIDI note number, and velocity are passed to the Decoder to generate a sample. Once the initial sample is generated, users can modify the values for Z, timbre measures, MIDI note number, and velocity to navigate the latent space of the autoencoder, and generate samples with desired timbral properties. Modifying Z only and leaving all other parameters unchanged has a significant impact on sample generation. We implemented a web-app featuring only SG that can be used to conveniently

generate sounds and explore the semantic latent space and Z.[8] We also provide a paper companion website where we share example guitar sounds generated with the instrument.[9]

3.4 Sampler component

The Sampler component is responsible for managing users' interaction with generated samples, providing an interface for playback using an on-screen keyboard as well as accepting note input from MIDI channels. Following synthesis, the Sampler receives samples from SG across a four-octave note range. Each received one-shot is up-sampled and its amplitude is normalized. Playback is handled by the Csound library (Boulanger [2000]), which provides interfaces for audio effects. A simplified user experience to record voice commands, input text commands, generate new sounds, and explore the latent space using sliders is provided. Sliders are associated to timbre measures, that can be tweaked to adjust sound. Users are also given flexibility to change settings related to MIDI input, microphone input, audio output, buffer size, and latency. To coordinate user interactions with several DL components, we implemented an application state machine. Appendix C[5] provides a detailed account of how the state machine works.

4 Evaluation

We carried out expert-based evaluation of the proposed instrument. We selected five external evaluators with 6-20 years of experience in music and music technology. Evaluators are all academically and/or professionally active musicians, multimedia artists, and audio software programmers.

4.1 Evaluation procedure

We sent evaluators a video tutorial, installation instructions, and documentation to get started with the instrument. We asked them to use the system freely for one hour, during which they should explore its sound design possibilities. After the practical session, evaluators had to provide feedback through a survey which aimed to gather quantitative data on the respondents' opinion about the generated sounds and the instrument. The survey was followed by a 30-minute interview to gain qualitative insights on how evaluators viewed the software as a guitar sample synthesiser, how they interacted with the input and timbre sliders, and how – if at all – they would fit such a tool in a music production workflow as a creative assistant. Evaluators were asked to not use the Speech Recognition component, and directly input text commands in the app instead. This was done to avoid speech transcription errors which would hinder the evaluation of the core components of the system (i.e., DR, SG).

4.2 Evaluation results

A breakdown of the quantitative scores provided in the survey by the evaluators is presented in Table 1.[10] Overall, evaluators indicated that the sounds that the instrument could generate were interesting and unique. They also noted that the sound palette was varied.

When using the text input capabilities, evaluators used descriptors such as "dark", "bright", "thin", "fat", "electric", "acoustic", and "bass", or phrases such as "the most joyful sound". Some users agreed that they were able to generate a sound that matched the input text, while some found it difficult to discern how a specific word influenced a sound as the differences were "...very subtle and not really evident enough if you are not listening for it". But "pushing the app outside its [dictionary] and asking for an instrument other than guitar" led to a "serendipitous result" and encouraged exploration of the instrument.

By interacting with the timbre sliders, evaluators found a "palette of sounds that [is] quite varied", which generally matched the expected change in the sliders, as shown by the 3.8/5 score in the relative query in the survey. Evaluators reported the timbre sliders to provide more expansive sound design

[8] The Sound Generation web-app is accessible at `https://share.streamlit.io/thesoundofaiosr/rg_sound_generation/main/app.py`.

[9] Examples of generated guitar sounds can be found at `https://thesoundofaiosr.github.io/Paper_Companion/`.

[10] Full results of the survey can be accessed at `https://docs.google.com/spreadsheets/d/1cKlabSU49aWZ87Bx-0xxhOjtQ9AQprgWdfhZeJ7Yy5s/edit?usp=sharing`.

Table 1: Results of the evaluation survey. Scores are on a Likert scale in range 1 (worst) to 5 (best).

Query	Score (mean / std dev)
The generated sound matches the input text command	2.80 / 1.48
The heuristic sliders impact the generated sound as I expected	3.80 / 1.09
How would you rate the fidelity of the sound generated?	4.00 / 1.22
How extensive are the sound design possibilities?	4.00 / 1.41
How easy was it to learn how to use this tool?	3.60 / 1.14
How would you rate the user experience of using this tool?	3.20 / 1.30
How likely are you to keep using this product?	2.80 / 1.09

possibilities than the text input. So called "happy accidents" were likely to occur and sounds that would "have [only been] achieved [with] a lot more work [in traditional synthesisers]" were generated. Some parameters of the latent space did not "[respond] quite as expected" in certain conditions. For example, setting a long attack time was reported to sometimes result in a plucky sound with a short attack. This could be caused by the fact that the timbre sliders are not completely independent and their interactions may cause unexpected results. This behaviour may also partially explain the weaker score around the generated sounds matching the input text (2.80).

Some of the drawbacks presented by the evaluators were the impossibility to generate looped and sustained sounds and the not-so-intuitive user experience. This led them to say that they would most likely use this instrument as a sound design tool to generate samples that would then be incorporated into their music production workflows. Another notable drawback was the lack of a preset management system to save and recall the sounds discovered while exploring the instrument. The addition of an effect bank, like a simple reverb and delay, was also highlighted as an obstacle toward a more fruitful use of the software in their music production practice.

In summary, evaluators reported the software to provide unique sound design possibilities that could be used in a music production workflow, but further work to the sound generation and user experience would be needed to consider it a marketable music production tool.

5 Conclusions

In this paper, we set out to address a recurring problem found in DL-powered audio synthesis models: the dimensions of the latent spaces offered by such systems are not easily mapped to any semantically meaningful timbral concepts. This limitation makes the exploration of such timbral spaces problematic. To address this drawback, we built an instrument that is able to generate guitar sounds from vocal commands, describing the timbral characteristics of the expected sound.

The developed technology provides significant contributions to the fields of neural audio synthesis and semantic representation of timbre. DR understands the timbral qualities requested by a user and maps them to standardised descriptors encoded in an innovative Guitar Sound Taxonomy. The Timbre Conditioned Autoencoder generates high-fidelity audio samples without the need of a large model and large latent space. Its generation can be conditioned on timbral descriptors, and the dimensions of its latent space are mapped to semantically meaningful timbre measures that facilitate user exploration. We also introduced compact semantic audio representations which extend QHM via a novel set of perceptually-motivated timbre measures. The measures enable researchers to produce semantic audio representations from unlabelled sound datasets.

Expert-based evaluation confirms that the proposed instrument is capable of producing interesting and unique sounds, by leveraging semantic descriptors. Evaluators used timbre sliders to design unexpected guitar sounds quickly, and to effectively navigate the latent space. However, future research is necessary to turn this prototype instrument into a solution that could be integrated in the day-to-day musician's creative workflow. First, the implementation of an end-to-end DL model that can map descriptors embedding to timbre measures would allow the instrument to generate samples that better match requested timbral qualities. Second, an improved interface with convenient features such as preset saving would enhance the user experience. Finally, the neural synthesiser could be trained on a dataset with more instruments to extend its sound generation capabilities beyond guitar sounds.

References

A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33: 12449–12460, 2020.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

R. Boulanger. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT press, 2000.

J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *International Conference on Machine Learning*, pages 1068–1077. PMLR, 2017.

J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts. GANSynth: Adversarial neural audio synthesis. In *International Conference on Learning Representations*, 2019.

J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts. Ddsp: Differentiable digital signal processing. In *International Conference on Learning Representations*, 2020.

S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6124–6128. IEEE, 2020.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Y. Pantazis, O. Rosec, and Y. Stylianou. On the properties of a time-varying quasi-harmonic model of speech. In *Ninth Annual Conference of the International Speech Communication Association*, 2008.

G. Peeters, B. L. Giordano, P. Susini, N. Misdariis, and S. McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5): 2902–2916, 2011.

M. Russ. *Sound synthesis and sampling*. Routledge, 2012.

X. Serra et al. Musical sound modeling with sinusoids plus noise. *Musical signal processing*, pages 91–122, 1997.

Y. Shen, H. Yun, Z. C. Lipton, Y. Kronrod, and A. Anandkumar. Deep active learning for named entity recognition. *arXiv preprint arXiv:1707.05928*, 2017.

J. O. Smith. Physical modeling synthesis update. *Computer Music Journal*, 20(2):44–56, 1996.

K. Subramani, P. Rao, and A. D'Hooge. Vapar synth-a variational parametric model for audio synthesis. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 796–800. IEEE, 2020.

K. Tatar, D. Bisig, and P. Pasquier. Latent timbre synthesis. *Neural Computing and Applications*, 33 (1):67–84, 2021.

A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125:2, 2016.

Y. Vasiliev. *Natural Language Processing with Python and SpaCy: A Practical Introduction*. No Starch Press, 2020.

Authors

This paper is the culmination of a research project carried out collaboratively by The Sound of AI community in a bottom-up fashion over a period of 1.5 years. Contributors have fulfilled different roles. To credit the work of different authors fairly, we have decided to list the community members' contribution adhering to the Contributor Roles Taxonomy (CRediT).

It is to be noted that more than 150 members participated to the project at different times, but only a minority decided to be listed as an author.

1. **Valerio Velardo**: Conceptualization, Project Administration, Resources, Supervision, Writing – Review & Editing.
2. **Wassim Filali**: Conceptualization, Investigation, Methodology, Project Administration, Resources, Software, Supervision, Validation, Visualization.
3. **Fernando Garcia de la Cruz**: Conceptualization, Investigation, Software, Project Administration, Validation.
4. **Tibor Kiss**: Conceptualization, Methodology, Investigation, Project Administration, Software, Supervision, Validation.
5. **Ruby Annette**: Conceptualization, Data Curation, Methodology, Investigation, Project Administration, Software, Supervision, Validation, Writing – Review & Editing.
6. **Amit Yadav**: Conceptualization, Data Curation, Methodology, Investigation, Project Administration, Software, Supervision, Validation, Writing – Original Draft.
7. **Jesse Hill**: Project Administration, Conceptualization, Investigation, Software, Supervision, Validation, Resources, Methodology, Writing Original Draft.
8. **Lucas Hagemans**: Conceptualization, Data Curation, Investigation, Methodology, Project Administration, Supervision, Writing – Original Draft.
9. **Chris Kang**: Conceptualization, Methodology, Investigation, Project Administration, Software.
10. **Tarun Bisht**: Data Curation, Investigation, Methodology, Project Administration, Software, Writing – Original Draft, Conceptualization.
11. **Fabio Di Marco**: Conceptualization, Data Curation, Methodology, Investigation, Project Administration, Software, Supervision, Validation, Writing – Original Draft.
12. **Leonardo Foletto**: Conceptualization, Investigation, Software, Supervision, Methodology, Resources, Validation, Project Administration, Writing Original Draft.
13. **Gianluca Zuccarelli**: Conceptualization, Investigation, Methodology, Project Administration, Supervision, Validation.
14. **Andrew Wray**: Software, Validation, Methodology.
15. **Yasser Khalafaoui**: Data Curation, Formal Analysis, Investigation, Writing – Original Draft, Supervision, Project Administration.
16. **Jacob Richter**: Project Administration, Conceptualization, Investigation, Methodology, Resources.
17. **Vyron Vasileiadis**: Conceptualization, Investigation.
18. **Vadim Groshev**: Investigation, Software.
19. **Ahmad Omar Ahsan**: Investigation, Methodology, Conceptualization.
20. **Umang Keshri**: Conceptualization, Data Curation, Project Administration.
21. **Antonio Giganti**: Investigation.
22. **Raheel Anjum**: Investigation, Resources.
23. **Tim Lin**: Investigation, Software.
24. **Stephen Lau**: Data Curation, Conceptualization.
25. **Paras Jain**: Resources.
26. **Juan Carlos Piñeros**: Conceptualization, Investigation, Methodology, Resources.

27. **Stefano Imoscopi**: Conceptualization.
28. **Torsha Mondal**: Conceptualization, Data Curation, Resources.
29. **Manoj Kolpe Lingappa**: Software, Resource, Supervision, Validation, Investigation.
30. **Krishna Maneesha Dendukuri**: Investigation.
31. **Mirco Nani**: Conceptualization, Methodology, Software, Supervision, Project Administration, Formal Analysis, Data Curation, Investigation, Writing – Original Draft.
32. **Mubeen Shaik**: Investigation.
33. **Erik Ringaby**: Conceptualization, Supervision, Investigation.
34. **Oliver Ilnicki**: Supervision, Investigating, Software, Methodology, Conceptualization.
35. **Zao Soula**: Investigation.
36. **Widi Persadha**: Investigation, Data Curation.
37. **Umar Farooq Shaik**: Conceptualization, Investigation, Software, Visualization.
38. **Pradeep balaji M**: Investigation, Conceptualization.
39. **Beat Tödtli**: Formal Analysis,Writing – Original Draft, Investigation, Methodology, Data Curation, Validation.
40. **Balaji V**: Conceptualization, Investigation, Software, Validation.
41. **Humberto Pérez-Espinosa**: Investigation.
42. **Victor Ojewale**: Investigation.
43. **Shubham Bindal**: Investigation.
44. **Francesca Ronchini**: Investigation, Software.
45. **Louai El Achkar**: Investigation, Data Curation, Conceptualization.
46. **Reid Yonkers**: Investigation.
47. **Sam Cantor**: Conceptualization, Data Curation, Investigation, Software.
48. **Dhananjay Jindal**: Software, Data Curation, Validation.
49. **Archit Yadav**: Investigation, Formal Analysis, Writing – Original Draft, Validation, Visualization.
50. **Neil Conway**: Investigation.
51. **Elnur Yusifov**: Software, Methodology.
52. **Jonathan Mukiibi**: Resources, Validation.
53. **Soumya Sai Vanka**: Data Curation.
54. **Sam Burtch**: Data Curation, Investigation, Methodology, Conceptualization, Writing – Original Draft, Writing – Review & Editing.