NoiseBandNet: Controllable Time-Varying Neural Synthesis of Sound Effects Using Filterbanks

Adrián Barahona-Ríos and Tom Collins

Abstract—**Controllable neural audio synthesis of sound effects is a challenging task due to the potential scarcity and spectro-temporal variance of the data. Differentiable digital signal processing (DDSP) synthesisers have been successfully employed to model and control musical and harmonic signals using relatively limited data and computational resources. Here we propose NoiseBandNet, an architecture capable of synthesising and controlling sound effects by filtering white noise through a filterbank, thus going further than previous systems that make assumptions about the harmonic nature of sounds. We evaluate our approach via a series of experiments, modelling footsteps, thunderstorm, pottery, knocking, and metal sound effects. Comparing NoiseBandNet audio reconstruction capabilities to four variants of the DDSP-filtered noise synthesiser, NoiseBandNet scores higher in nine out of ten evaluation categories, establishing a flexible DDSP method for generating time-varying, inharmonic sound effects of arbitrary length with both good time and frequency resolution. Finally, we introduce some potential creative uses of NoiseBandNet, by generating variations, performing loudness transfer, and by training it on user-defined control curves.**

Index Terms—**Neural audio synthesis, differentiable digital signal processing, sound effects, procedural audio, game audio.**

I. Introduction

In MEDIA, sound effects can be defined as those sound elements other than music or speech [1]. Typical sound effects are, for instance, footsteps or environmental sounds such as rain. This broad definition implies that sound effects may exhibit, within the same category, wide and narrow spectral bands or static and transient amplitude envelopes [2]. Sound effects are usually produced by sound designers or foley artists by either recording the sounds on demand or sourcing and transforming the assets from pre-recorded sound libraries. However, with the increasing size and complexity of video games and interactive media, creating enough sound assets is time-consuming, especially in scenarios such as virtual reality (VR), where players may freely interact with elements of the virtual environment using haptic controllers [3].

Alternatively to pre-recorded samples, sound effects can also be produced using sound synthesisers – a method which is often called procedural audio [4]. Typically, procedural audio models are handcrafted and built upon digital signal processing (DSP) algorithms running in real-time with parametric controls [4]. Yet the process of building procedural audio models may be challenging for sound designers, and the resulting audio may also lack plausibility when compared to pre-recorded samples [5], [6]. Differentiable digital signal processing (DDSP) [7] commonly refers to the concept of using DSP algorithms alongside deep learning. In the original DDSP paper, Engel et al. [7] synthesise harmonic musical notes controlled by pitch and loudness using a harmonic plus noise synthesiser [8]. Once trained, the resulting synthesiser is able to produce sounds with human-interpretable controls (e.g., pitch and loudness).

In the context of the synthesis of sound effects, and particularly in game audio, human-interpretable continuous controls are desirable, as the synthesiser could adapt its output to, for instance, in-game events (in the case of running in real-time) or to animations (running offline). DDSP-based models also benefit from requiring comparatively less data to train than other data-driven approaches [7]. Additionally, DDSP synthesisers have been demonstrated to be able to run in real-time [9], offering the potential of being integrated into live scenarios such as video games.

Our end goals are to build a general-purpose DDSP synthesiser capable of producing a) sounds with acceptable time and frequency resolution, and b) audio of arbitrary length, just by providing conditioning vectors containing the desired parametric controls. The original DDSP synthesiser [7] relies on the premise that the audio it aims to model is harmonic, which is not the case for most sound effects. Very recently, [10] proposed a method to estimate sinusoidal components using gradient descent, which has been a challenging task when using Fourier-based loss functions [11], opening the possibility of modelling inharmonic sinusoids using DDSP synthesisers. Sound effects, however, may contain noisy or very narrow-band elements that are difficult to model using sinusoidal partials [2], plus the method of [10] has yet to be applied to the context of sound effects. Another option could be to use a time-varying finite impulse response (FIR) filter as in the original DDSP subtractive noise synthesiser, but it suffers from a time and frequency trade-off, where, in order to obtain good frequency resolution, the number of taps in the FIR filter need to be

Manuscript received 13 July 2023; revised 19 December 2023; accepted 24 January 2024. Date of publication 19 February 2024; date of current version 28 February 2024. This work was supported in part by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) under Grant EP/L015846/1 and in part by Sony Interactive Entertainment Europe. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Isabel Barbancho. *(Corresponding author: Adrián Barahona-Ríos.)*

Adrián Barahona-Ríos is with the Department of Computer Science, University of York, YO10 5GH York, U.K. (e-mail: ajbr501@york.ac.uk).

Tom Collins is with the Frost School of Music, University of Miami, Coral Gables, FL 33146 USA, and also with Music Artificial Intelligence Algorithms, Inc., Davis, CA 95616 USA (e-mail: tomthecollins@gmail.com).

Digital Object Identifier 10.1109/TASLP.2024.3364616

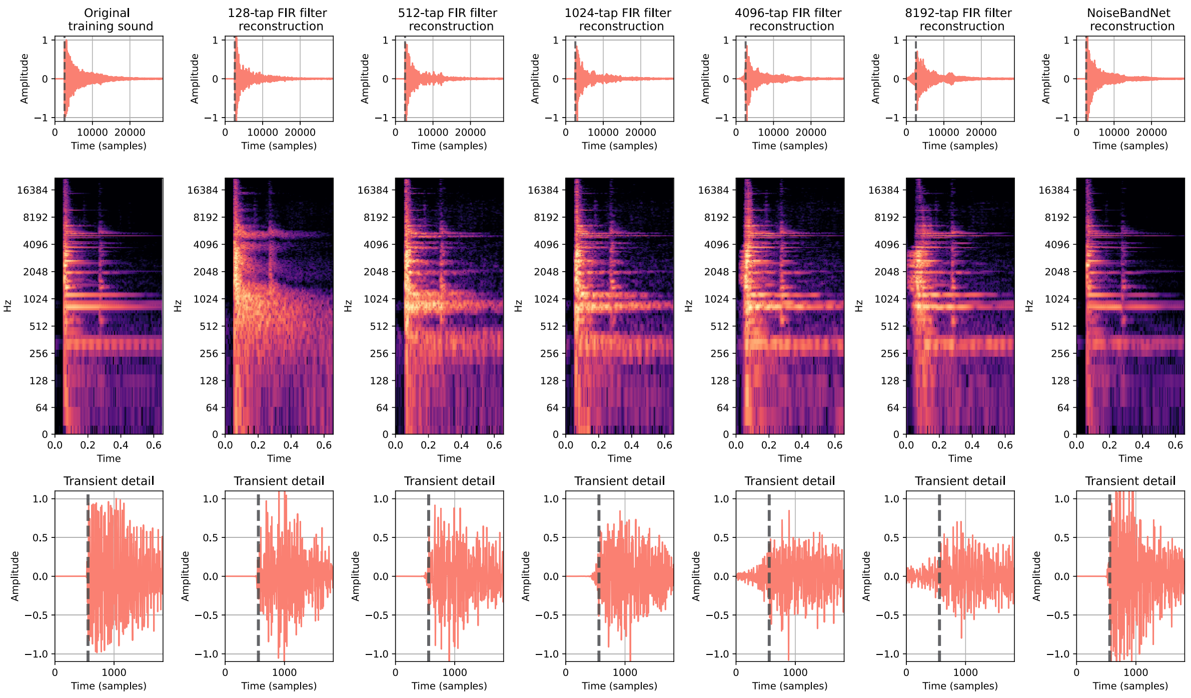

Fig. 1. Reconstruction task comparison between the DDSP time-varying FIR noise synthesiser and NoiseBandNet. The top row shows the waveform of the entire sound, the middle row its log-magnitude spectrogram and at the bottom a detail of the transient. The transient spot is annotated with a vertical dashed line in the first and third rows. The left column shows the original training sample: a short metal impact. The middle columns show the reconstruction of five different configurations of the DDSP time-varying FIR noise synthesiser with 128, 512, 1024, 4096, and 8192 taps, respectively, all of them with a hop size of 32 samples. Observe its time and frequency trade-off: the frequency resolution increases with the number of taps at the same time the time resolution decreases, and vice-versa. The right column shows the NoiseBandNet reconstruction using 2048 filters and a synthesis window size of 32 samples, maintaining both good time and frequency resolution.

relatively high, which in return smears the transients, and vice-versa. This phenomenon is depicted in Fig. 1. As an alternative, and inspired by the work of [2] where they use multi-rate filterbanks and sub-band processing to overcome the time and frequency trade-off of the inverse FFT method (very closely related to the DDSP FIR-noise synthesiser case), we explore the use of filterbanks in this context, leading to a definition of a new architecture called NoiseBandNet. While we do not use sub-band processing in our work, we incorporate some of the ideas from [2] into a differentiable pipeline, linking human-interpretable control parameters to the output audio. We compare NoiseBandNet to the original DDSP noise synthesiser and establish a more suitable method to generate time-varying inharmonic sound effects of unconstrained length using DDSP synthesisers, with both good time and frequency resolution. Thus, our main contributions, framed within the extension of the DDSP architecture, are: the possibility of training and synthesising sounds employing high-level control vectors (including user-defined ones); the proposed synthesis method, which allows for the generation of arbitrary sounds without preconceived characteristics (e.g., not being restricted to harmonic sounds). Code and audio examples can be found at the project website.[1]

[1]https://adrianbarahonarios.com/noisebandnet/

II. Related Work

There are multiple studies on the synthesis of sound effects using DSP techniques. From aeroacoustic [12] or footstep [13] sounds, to guidelines to choose appropriate synthesis methods [4], or efforts to bring models to the wider public [14]. Most current models, however, still lack plausibility when compared to pre-recorded samples [5], [6].

Audio synthesis using deep learning, often called neural audio synthesis, can be seen as an alternative to pure DSP-driven synthesis. While this work has usually focused on speech or music signals, studies on the synthesis of sound effects exist. For instance, in [15], knocking sound effects are synthesised conditioned by emotions, and in [6], footstep sound effects are synthesised conditioned on surface materials. Other approaches focus on more general environmental sound categories [16], [17]. There has also been a growing interest in generating sound effects conditioned on natural language prompts [18], [19]. There is work addressing scarcity of training data when using neural audio synthesis too, especially relevant for sound effects. In [20], unconditional sound variations of arbitrary length are produced just by providing ≈20 seconds of data, and in [5], unconditional variations of short (≈200–750 ms) one-shot sound effects are synthesised by training on a single audio example, which has also been applied to the task of data sonification [21]. More recently, [22] generates novel sound effect variations of arbitrary length conditioned on mel-spectrograms, training on a small dataset.

DDSP architectures can be seen as a middle ground between models trained on large datasets over extended periods of time and architectures that generate data from a few examples, as they exploit inherent biases in DSP components such as filters or oscillators to facilitate the training, synthesis and control tasks. Apart from the original DDSP architecture [7], other studies explore the use of waveshaping [23], wavetable [24], or frequency modulation (FM) [25] synthesis, all of them focusing on the modelling of harmonic or musical sounds. Other approaches, closer to sound effects, focus on the synthesis of rigid-body impacts by predicting the properties of resonant infinite impulse response (IIR) filters based on the object shape [26], the synthesis of harmonic engine sounds [27], or footsteps using the original DDSP time-varying FIR noise synthesiser [28]. In our case, we aim to provide a general-purpose method that, in principle, does not have any pre-conceptions about the target sound to be modelled, and that is able to render both wide and narrow spectral components of time-varying sounds.

Creatively controlling and conditioning deep learning audio models has also been studied previously. In [29], drum sounds are synthesised conditioned on timbral features. Other approaches such as [30] drive the synthesis of environmental sounds using onomatopoeic words, perform real-time timbre transfer [31], use the latent space of a generative adversarial network (GAN) [32] to condition a recurrent neural network (RNN) to synthesise sound textures [33], or generate music by pose sequences [34]. Topics akin to the inverse of these control schemes have also been investigated, such as providing an audio example as input, and using a neural network approach to retrieve synthesiser control parameters [35].

The use of filterbanks in the context of sound texture synthesis was investigated in [36], where target sounds were decomposed into sub-bands using a cochlear filterbank, and a set of statistics were extracted from their amplitude envelopes to build a synthesis model. In contrast to this, multi-rate filterbanks have also been used to model time-varying environmental sounds with narrow spectral components, thus using the filterbank structure itself to spectrally shape white noise [2]. They use sub-band processing to approximate the frequency response of a series of FIR filters that comprise a filterbank, and multiply each of the noise bands resulting from filtering white noise though the filterbank by a time-varying amplitude in order to match the target audio. Our method is related to [2] as we also employ a filterbank to process white noise in order to reconstruct a target sound. We do not employ sub-band processing, however, and do incorporate the filterbank as part of a broader deep learning model in order to link control parameters to the reconstructed audio, allowing for the control of the generated sounds.

III. Method

Our proposed method consists of using a deep learning model similar to the original DDSP architecture, conditioned on high-level audio controls to output M-band time-varying amplitudes, and multiply them by the M bands resulting from processing white noise through a filterbank. From a high-level perspective, we first build a filterbank comprised of adjacent FIR filters with narrow frequency responses that jointly cover an arbitrarily wide-ranging frequency spectrum. Then, in order to ease the computational burden of our approach, we precompute the filtering operation on a white noise instance with all the different filters of the filterbank, “baking” those noise bands. Lastly, we use an architecture similar to the original DDSP paper to predict the time-varying amplitudes of each of the bands for a target dataset, conditioning it on high-level controls, and effectively linking control parameters to the output of the synthesiser. The final output is generated by summing all the bands together in the time-domain.

A. Filterbank Design

Since our method does not make assumptions about the frequency content of the sound to be modelled, and in order to allow for the synthesis of both narrow and broad frequency components, we need to design narrow bandpass filters, with the union of their combined frequency responses covering the whole frequency spectrum $[0, \mathrm{Fs}/2]$, where Fs is the sampling rate, which we set to $\mathrm{Fs} = 44.1\,\mathrm{kHz}$. Thus, two adjacent bandpass filters will share one of their two band edges with each other to cover the totality of the frequency spectrum.

We start by deciding the number of filters M that will comprise the filterbank to obtain a good frequency resolution, which, based on pilot experiments, we set to $M = 2048$ filters. Second, we decide how those filters are going to be distributed across the frequency spectrum. As in [2], we emphasise the lower end of the frequency spectrum by covering those frequencies with more filters than at the higher end. Specifically, we use half of the filters (1024 in our case, [1,...,1024]) to cover the first quarter of the frequency spectrum $[0, \mathrm{Fs}/8]$, distributing their center frequencies linearly in the interval. The other half of the filters (the other 1024 filters, $[1025, \ldots, M]$) cover the remaining interval of the frequency spectrum, $[\mathrm{Fs}/8, \mathrm{Fs}/2]$, with their center frequencies spaced evenly on a logarithmic scale, thus increasing their bandwidth along it (i.e., filters at the higher end of the spectrum have a greater bandwidth than filters at the lower end). While this distribution may not be optimal, in general a more densely populated lower end of the spectrum is desirable, since the human ear is more sensitive in this range than in the higher end [37]. That said, our method is flexible, so could initiated with different filterbank configurations in future.

To implement the filterbank, we use real FIR filters, designing them with the Kaiser window method with a transition width Δ_ω of 20% of the filter bandwidth:

$$\Delta_\omega = \frac{|\omega_1 - \omega_2|}{\mathrm{Fs}} \cdot 0.2 \tag{1}$$

where Fs is the sampling rate, and ω_1 and ω_2 are the left and right band edges respectively. We use a stopband attenuation of $A_s = 50\,\mathrm{dB}$, as in [2]. All the filters are bandpass except for the first one, which is a lowpass filter that covers the $[0, f_{\text{min}}]$ interval, with $f_{\text{min}} = 20\,\mathrm{Hz}$; and the last one, which is a highpass filter that covers the $[\omega_1, \mathrm{Fs}/2]$ interval, where ω_1 is the right band edge of the penultimate bandpass filter.

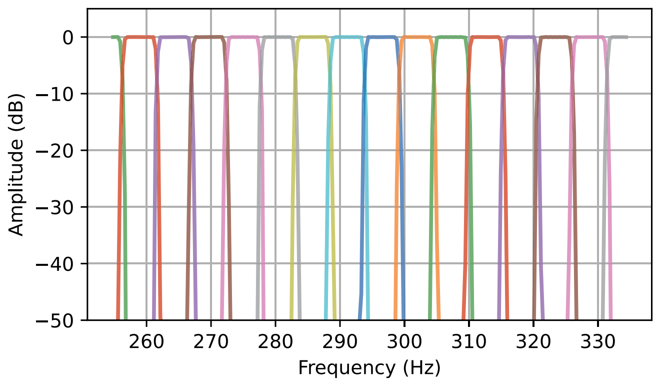

Fig. 2. Detail of the frequency response of some of the filters employed in a 2048-filter filterbank. Each of the filters is represented by a different colour.

An example of the frequency response of some of the filters is depicted in Fig. 2. For reference, using the configuration described above, the longer FIR filter in the filterbank has a total of 120287 taps. The bandwidth of the linearly-distributed (bandpass) filters in the low end is $B_{(1,\ldots,1024)} \approx 5.4\,\text{Hz}$, and the bandwidth of the last (highpass) filter is $B_M \approx 30\,\text{Hz}$. Once we build the filterbank, we zero-pad all the filters' impulse responses to the next power of 2 of the length of the filter with more taps, so they all have the same length. Using the proposed configuration, this results in filters with 131072 taps. The large length of these filters is due to their very narrow nature.

B. Deterministic Loopable Noise Bands

Considering we use many ($M = 2048$) and long (131072-tap) FIR filters in our system, generating the noise bands themselves (i.e., a convolving a noise instance with all the filters) is a computationally expensive operation, which can bottleneck both the training and inference of the model. This is especially true during training where, at each training step, the noise bands may need to be recalculated; or in longer sequences during inference (e.g., generating 120-seconds' worth of audio).

To ease the computational burden of our system, we follow the technique described in [38], where they propose a method to extend stationary sounds such as airplane cabin noise, but applying it to the noise bands resulting from filtering a white noise instance with all the filters of the filterbank. Our aim is to compute these noise bands only once and store ("bake") them, removing the need of recomputing the operation each time the synthesiser generates an output.

More specifically, we use their proposed FFT convolution approach that leads to sounds that can be concatenated along their x-axis thanks to circular convolution. By the convolution theorem, it is known that convolution in the time-domain is equivalent to point-wise frequency-domain multiplication, which can be written as follows for the filtering of a white noise signal with an FIR filter [38]:

$$y = R_{\text{noise}} R_{\text{filter}} e^{j(\theta_{\text{noise}} + \theta_{\text{filter}})}, \tag{2}$$

with R and θ representing the magnitude and phase respectively resulting from the FFT. Since white noise ideally has a flat magnitude response, they set it to unity $R_{\text{noise}} = 1$, and since the phase of the noise signal, θ_{noise}, already randomises the phase of the operation ($\theta_{\text{noise}} + \theta_{\text{filter}}$), they replace it with a random phase, obtaining the final expression [38]:

$$y = R_{\text{filter}} e^{j(\theta_{\text{random}})}, \tag{3}$$

where θ_{random} is formed by uniformly distributed random values drawn from a $[-\pi, \pi]$ interval and having its first and last values (DC and Nyquist frequencies, respectively) set to 0 [38]. Since the FFT exhibits Hermitian symmetry for real-valued data, the values beyond the Nyquist frequency (the negative frequency values) are just the complex conjugate of the positive ones mirrored from the Nyquist frequency, excluding the Nyquist and DC elements. Therefore, θ_{random} is defined as follows:

$$\theta_{\text{random}} = (0, r_1, r_2, \ldots, r_n, 0, -\overline{r}_n, -\overline{r}_{n-1}, \ldots, -\overline{r}_1) \tag{4}$$

where r are the uniformly distributed random values drawn from a $[-\pi, \pi]$ interval, and $-\overline{r}$ their mirrored (i.e., reversed) elements [38].

Finally, by taking the inverse FFT of (3), the "loopable" noise band is created due to the resulting circular convolution operation described above:

$$y = \text{IFFT}\left(R_{\text{filter}} e^{j(\theta_{\text{random}})}\right) \tag{5}$$

Thus, the M noise bands are built as follow:

$$\text{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} \text{IFFT}\left(R_{\text{filter}_1} e^{j(\theta_{\text{random}})}\right) \\ \text{IFFT}\left(R_{\text{filter}_2} e^{j(\theta_{\text{random}})}\right) \\ \vdots \\ \text{IFFT}\left(R_{\text{filter}_M} e^{j(\theta_{\text{random}})}\right) \end{bmatrix} \tag{6}$$

Using our proposed configuration, each of the noise bands have a length of 131072 samples, corresponding to ≈ 3 seconds of audio at 44.1 kHz. We also enforce a deterministic behaviour by setting the same random seed each time a noise band is generated. This is done to 1) maintain coherence each time noise bands are built (i.e., the noise bands used during training and inference will be identical) and 2) being able to share the same noise band instances across multiple models, granting they have the same filterbank configuration. Also, since the amplitude of each the resulting noise bands may be very small, due to the narrow portion of the frequency spectrum they focus on, we find the maximum amplitude value A_{max} across all the noise bands that comprise the filterbank, and divide all the bands by this A_{max} value, effectively scaling their amplitudes up to what we found to be a reasonable level. While this leads to neither homogeneous amplitude distribution across bands, nor a normalised amplitude (i.e., in the range $[-1, 1]$), when all the bands are summed together without further intervention, the scale of the individual noise bands will be handled by the time-varying amplitude predicted by the model (see Section III-C).

Thus, by using the method proposed in [38], we generate deterministic and loopable noise bands that only need to be computed once and can be extended arbitrarily in time by just concatenating them along their x-axis. An example of this is depicted in Fig. 3, where two instances of the same noise band

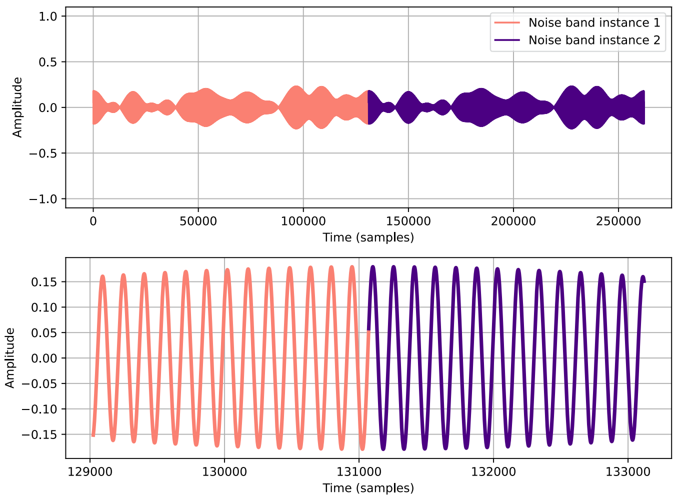

Fig. 3. Loopable noise bands. Two instances of the same noise band are concatenated along their x-axis. The top figure shows the waveform of both noise bands, one after the other, each one with a distinctive colour. The bottom figure shows the detail of the point where the end of the first noise band instance meets the start of the second one. Notice how, thanks to circular convolution, the start and the end of the segments are "joined up."

are concatenated, one after the other. Conceptually, each of those noise bands could be somewhat seen as a wavetable. A wavetable is defined as a block of memory (i.e., a "table") where a discretised signal is stored [39] and, while they are usually employed to store a single period of a waveform, a loopable noise band may be regarded as a period – as it can be looped – of the portion of the frequency spectrum it captures.

C. Architecture

NoiseBandNet, depicted in Fig. 4, is built upon the original DDSP architecture [7], but replacing their harmonic-plus-noise synthesiser with a filterbank structure. As in DDSP, the internal sampling rate of the model is a fraction of the target dataset sampling rate Fs. To obtain a good time resolution, and as in [2], we select a synthesis window size W of 32 samples, granting the model an internal sampling rate of Fs/W, thus producing an amplitude value every W samples. Greater W values will lead to poorer time resolution but less computational burden, and vice-versa.

The inputs to the neural network component of NoiseBandNet are the control parameters, which in the Fig. 4 are loudness and spectral centroid extracted from the training data. These control parameters may be different depending on the control scheme, such as only loudness, or other user-defined controls. Independently of the control scheme, and to synchronise the control parameters to the training data (i.e., to have a $1:1$ mapping between the control parameters and the samples in the target audio), originally the control parameters will have the same length as the dataset waveforms, interpolating them to this length if needed. Before passing the control parameter vectors to the network, we resample them according to Fs/W, the internal sampling rate of our model.

Similar to [7], the control vectors are passed through a time-distributed multi-layer perceptron (MLP) block (one per control parameter vector and in parallel, as depicted in Fig. 4) and a gated recurrent unit (GRU) [40]. The output of the GRU is passed through a series of time-distributed MLP blocks leading to final time-distributed dense layer which outputs the the M-band time-varying amplitudes (each one of them corresponding to each of the noise bands), with a sampling rate of Fs/W. As in [7], to avoid negative amplitude values we scale the resulting amplitudes using a modified sigmoid activation function, in our case:

$$y = 2.0 \cdot \mathrm{sigmoid}(x)^{\ln(10)} + 10^{-18} \tag{7}$$

Then, to bring them to audio rate Fs, we upsample these amplitudes by a factor of W using linear interpolation. We then multiply the amplitudes by the noise bands in the time-domain. To clarify, while it would be possible to multiply the amplitudes by the noise bands directly by downsampling the latter to a Fs/W sampling rate, and upsampling the resulting operation to audio rate Fs afterwards, this could introduce artifacts derived from the resampling operation. Therefore we opt to upsample the amplitudes first, to multiply them with the noise bands in the time-domain. Lastly, we sum all the M noise bands together to produce the final output audio:

$$y = \sum_{i=1}^{M} A_i \cdot \mathrm{band}_i, \tag{8}$$

where A_i and band $_i$ are the i th predicted upsampled amplitude and noise band respectively.

Unless stated otherwise, we model mono audio with a sampling rate Fs of 44.1 kHz.

D. Training and Inference

We train the network on batches of audio chunks of length L_{chunk}. We concatenate all the training waveforms along the time dimension and select random chunks of length L_{chunk} from them. This avoids the network memorising predicted amplitude values to the position of the training examples with respect of time, and so increases its generalisation capabilities when generating longer sequences (especially important when training with small datasets or one-shots). If the training dataset is comprised of a very short ($L_{\text{dataset}} < L_{\text{chunk}}$) training example, we simply repeat it along the x-axis until $L_{\text{dataset}} \geq L_{\text{chunk}}$. As the control parameters have the same length as the audio, we select the same chunk and resample it to Fs/W before passing them to the network. Note, we do not use any data augmentation techniques during training. Instead, we train solely on the audio data provided by the user, in conjunction with their intended control vectors.

Likewise, it is possible that the length of the training chunks L_{chunk} may be smaller than the length of the noise bands L_{bands}. To prevent the network being exposed to noise band portions never seen during training, we "roll" the noise bands along their x-axis at each training step, to a randomised integer shift drawn from a uniformly distributed random value in $[0, L_{\text{bands}}]$, achieving the use of a different, randomised portion of the noise bands at each training step. During training, we compare the output audio against the target audio using a multi-resolution

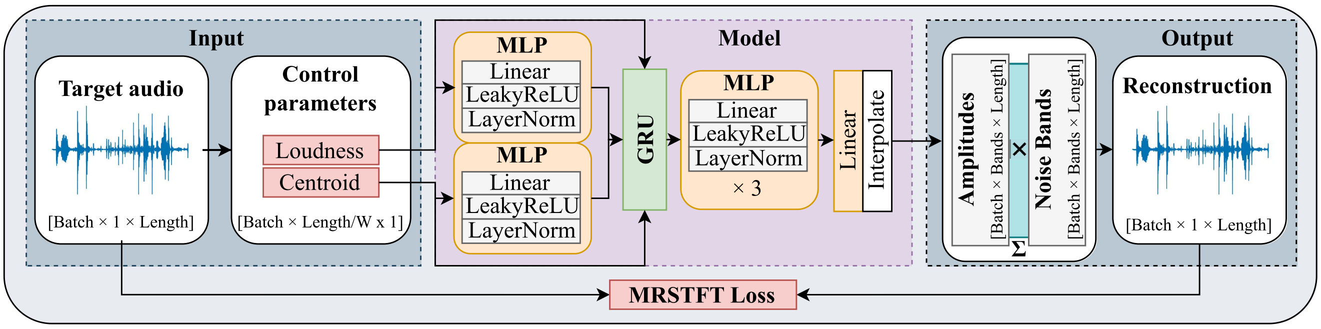

Fig. 4. Overview of the NoiseBandNet architecture and training process. In this case, loudness and spectral centroid features are extracted from the training audio and passed to the network, which predicts an M-band matrix of time-varying amplitudes at an Fs sampling rate divided by a synthesis window size W. Depending on the control scheme, these features or control parameters may be different (e.g., only loudness, or user-provided control parameters). The predicted amplitudes are upsampled using linear interpolation by a factor of W to match the audio length, and multiplied by the M noise bands. The output audio is generated by summing all the bands together. Finally, the model is optimised by comparing the resulting sound against the target audio using a multi-resolution STFT (MRSTFT) loss.

STFT (MRSTFT) loss [41], with the aim of reconstructing the input audio for the given control parameters.

Once trained, the model needs only control parameter vectors of arbitrary length L_{control} to produce an output of $L_{\text{control}} \cdot W$ length in samples. Due to the nature of the architecture, this output is deterministic (i.e., the model produces the same output amplitudes for the same control parameter input). However, in practice, the output audio resulting from multiplying the noise bands by the same predicted amplitudes may be slightly different since, as described above, we randomise the start of the noise bands by a $[0, L_{\text{bands}}]$-shift, and their energy is not constant over their length (refer to Fig. 3, where the amplitude of the band fluctuates over time).

IV. Reconstruction

First, we evaluate NoiseBandNet by comparing its reconstruction capabilities to different configurations of the original DDSP time-varying FIR noise synthesiser [7]. Their synthesiser produces an output by convolving white noise frame-by-frame with an FIR filter predicted by the network, then overlap-adding the frames. As in [7], we do not model the FIR filters' impulse responses directly, but their magnitudes.

A. Experiments

To evaluate the reconstruction capabilities of the systems, we select five sound effect categories relevant to video games, which exhibit both broad and narrow spectral components and a wide range of amplitude envelopes [4], [15]:

- Footsteps (≈4.4 seconds): Footsteps on a metallic staircase.
- Thunderstorm (≈14.0 seconds): Rain and close thunder sounds.
- Pottery (≈95.0 seconds): Breaking and scrapping pottery shards.
- Knocking (≈11.0 seconds): Knocking sound effects with different intensities and emotional intentions.
- Metal (≈19.0 seconds): Hitting and scrapping metal bars.

We source all the training sound effects from the Freesound website [42], except for the knocking sound effects, where we use an excerpt of the dataset provided by [15].

We choose loudness and spectral centroid to evaluate the reconstruction capabilities of the systems, as they are related to the original DDSP loudness and pitch control vectors, but without the constraint of being harmonically-oriented. To extract the loudness and the spectral centroid, we use an FFT size of 128 and 512 respectively, both with 75% overlap. For each sound category, we normalise each of the features to a [0,1] range. Note this is dataset-dependent: we do not normalise the control parameters using their their full range (e.g., $[0, \text{Fs}/2]$ in the spectral centroid case), but using the maximum and minimum values computed across a particular dataset. This prevents feature values being localised to a small portion of the $[0, 1]$ interval for certain sounds (e.g., quieter sound categories would have many loudness values near 0).

We train a NoiseBandNet model for each of the sound categories using a hidden size of 128 for all layers, an $M = 2048$-band filterbank, and a synthesis window W of 32 samples, following the same design described in Section III. We train all models for 10000 epochs using a learning rate of 0.001, batch size of 16, an audio chunk size of 65536 samples, Adam optimiser, and an MRSTFT loss [41] (with FFT sizes for the MRSTFT of $8192, 4096, 2048, 1024, 512, 128, 32$, 75% overlap, and window lengths of the same size as the FFTs), employing the auraloss implementation [43].

Using an NVIDIA Tesla V100, the training process takes ≈45 min for all models, except for the pottery model, which takes ≈180 min. Once trained, the saved model weights have a size of ≈1.8 MB, with each model having a total of 464 K parameters. During inference, the time required to synthesise a single batch signal with an output length of 1322976 samples (around 30 seconds of audio at 44.1 kHz) is of 529.5 ± 2.4 (mean ± sd) milliseconds on a consumer GPU (NVIDIA GTX 1060), and 13.4 ± 0.3 (mean ± sd) seconds on a consumer CPU (AMD Ryzen 5 1600), measured on a 100-run test.

We evaluate NoiseBandNet resynthesis capabilities against four variants of the original DDSP model filtered noise synthesiser [7], with a configuration of FIR filter taps of 256

TABLE I
MRSTFT LOSS (MEAN ± SD) AND FAD RESULTS FOR THE RECONSTRUCTION TASK ON SEVERAL SOUND CATEGORIES COMPARING THE DDSP FIR NOISE SYNTHESISER USING DIFFERENT CONFIGURATIONS AGAINST NOISEBANDNET

	Footsteps		*Thunderstorm*		*Pottery*		*Knocking*		*Metal*		Average values	
	MRSTFT	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***
NoiseBandNet	**1.14±0.01**	**5.41**	**1.24±0.01**	**9.06**	**1.22±0.04**	1.33	**1.10±0.01**	**2.44**	**1.15±0.01**	**4.65**	**1.17**	**4.578**
DDSP$_{256\text{ taps}}$	1.29±0.01	8.45	1.44±0.01	10.08	1.38±0.02	2.17	1.32±0.01	8.68	1.60±0.01	34.64	1.40	12.804
DDSP$_{512\text{ taps}}$	1.26±0.01	9.22	1.41±0.02	10.10	1.37±0.02	**1.22**	1.30±0.01	5.17	1.46±0.01	28.75	1.36	10.89
DDSP$_{1024\text{ taps}}$	1.24±0.01	9.89	1.42±0.01	10.33	1.38±0.03	1.55	1.29±0.01	5.35	1.35±0.01	25.92	1.34	10.61
DDSP$_{4096\text{ taps}}$	1.22±0.02	7.02	1.42±0.02	9.58	1.39±0.04	2.06	1.32±0.02	4.23	1.27±0.01	15.53	1.32	7.69

Lower values of MRSTFT loss and FAD are better (best performers highlighted in bold).

(DDSP$_{256\text{ taps}}$), 512 (DDSP$_{512\text{ taps}}$), 1024 (DDSP$_{1024\text{ taps}}$) and 4096 (DDSP$_{4096\text{ taps}}$). We use a hop size of 32 samples for all of the models. While such hop size is small for some models compared to a more standard 75% overlap, we use this value to 1) compare NoiseBandNet and DDSP using a configuration that is as close as possible for all systems, and 2) demonstrate that a smaller hop size does not necessarily improve the time resolution for the DDSP time-varying FIR noise synthesiser (refer to Fig. 1). We use a hidden size of 128 for all of the models and a single input MLP per input feature, as in NoiseBandNet (see Fig. 4). We employ the DDSP noise synthesiser Pytorch implementation found in [44],[2] do not model reverb, and use the same training configuration and loss function as the NoiseBandNet models.

B. Results

We use two objective metrics to assess all five models' reconstruction fidelities. First, the MRSTFT loss described above, measured from 19 different values at training time as models converged near their final epochs, to compensate for small possible fluctuations occurring during training. Second, the Fréchet Audio Distance (FAD) [45], a quality metric that correlates to human listeners better than spectral distances.[3] MRSTFT loss and FAD results are reported in Table I.

A two-way ANOVA on the MRSTFT loss data with factors for model (five levels, of NoiseBandNet and the four variants of the original DDSP model) and sound effect (five levels of footsteps, thunderstorm, pottery sounds, knocking sound effects and metal sounds) reveals a significant interaction ($F(16, 450) = 193, p < .001$), as well as significant main effects of model ($F(4, 450) = 2128, p < .001$) and sound effect ($F(4, 450) = 1217, p < .001$), suggesting that the type of model drives differences in loss, so does the type of sound effect, and that certain combinations of model and sound effect lead to either particularly low or high loss values.

An analysis of multiple pairwise comparisons (Tukey's Honest Significant Difference method) was conducted to investigate which pairings of groups differ. We found that NoiseBandNet significantly outperforms DDSP$_{256\text{ taps}}$ (mean diff $= 0.234,\ p < .001$), DDSP$_{512\text{ taps}}$ (mean diff $= 0.189,\ p < .001$), DDSP$_{1024\text{ taps}}$ (mean diff $= 0.164,\ p < .001$) and DDSP$_{4096\text{ taps}}$ (mean diff $= 0.150,\ p < .001$). Thus, NoiseBandNet obtains significantly better MRSTFT reconstruction values compared to the variants of the original DDSP noise synthesiser. In terms of the DDSP model variants' performance across the different sound effect categories, they were most effective for footsteps, then knocking, pottery, and thunder, with relatively small differences in performance between variants. This was in contrast to the metal category, where DDSP variants displayed relatively large differences in performance.

The FAD results follow a very similar pattern to those for MRSTFT, but we note the exception that DDSP$_{512\text{ taps}}$ performs better than NoiseBandNet in terms of FAD for the pottery sound effect category. This discrepancy may be caused by the small size of our datasets, which negatively affects the accuracy of the metric [45].

V. Creative Uses

As a second evaluative perspective, in this section we highlight some potential creative uses to which NoiseBandNet can be put. We encourage readers to listen to the audio examples, which can be found at the project website.[4]

A. Amplitude Randomisation

Given that NoiseBandNet uses DSP components at its core to produce audio, we can exploit their inherent biases to further alter the output signal. As outlined in Section III-D, the output amplitudes of the model using the proposed architecture are deterministic. Here, as an example, we present two strategies to generate variations from the predicted time-varying amplitudes. This may be especially relevant to game audio, where it is common to use multiple audio clips to sound design the same in-game interaction in order to avoid repetition [46].

First, we propose to randomise the top k amplitudes k_{amp} within a desired frame length L_{frame} (i.e., we randomise the output amplitudes each L_{frame} amplitude values). To achieve this, first we select the frame length L_{frame} and split the output amplitudes to non-overlapping frames of that length. Note we split these frames before performing the linear interpolation operation that upsample the amplitude values to audio rate. Then, we find the desired top amplitudes k_{amp} on each frame by summing the amplitude values for each of the bands on that frame across the time-axis and selecting the greatest k values. After that, we apply a randomised amplitude modifier in a user-defined range of $[\text{mult}_{\text{min}}, \text{mult}_{\text{max}}]$ by multiplying the amplitude values in that frame by it, scaling them up or down. Since all amplitudes still need to be interpolated to audio rate, the transition between their values is smoothed. We also found that if we compute a different amplitude randomisation for the same

[2]https://github.com/YatingMusic/ddsp-singing-vocoders
[3]https://github.com/gudgud96/frechet-audio-distance
[4]https://adrianbarahonarios.com/noisebandnet/

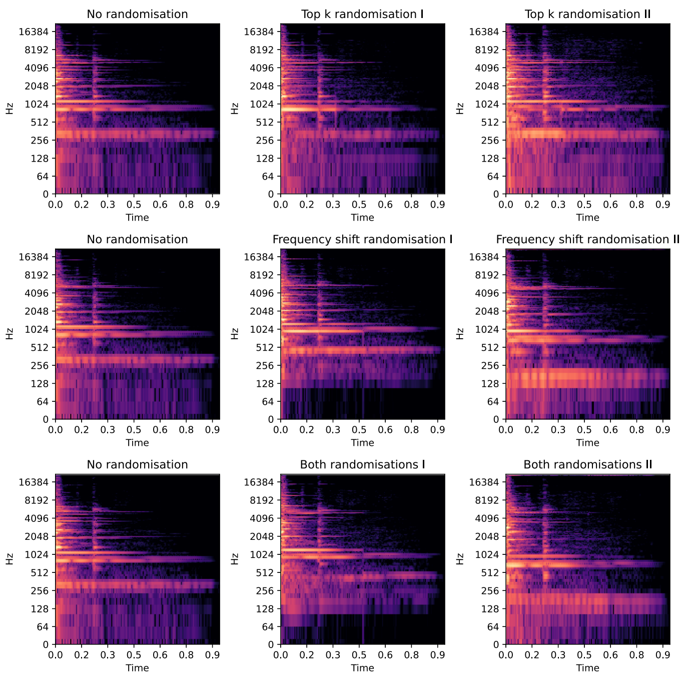

Fig. 5. Log-magnitude spectrograms from the result of the different randomisation schemes. The left column represents a non-randomised (just reconstructed) sound: a metal impact. The second and third columns show two examples of the resulting randomised sound. In the first row we employ the top k randomisation scheme using $L_{\text{frame}} = 430$ (3 frames), $k = 100$ and a randomised multiplier in a $[0.0, 1.0]$ range. The second row depicts the frequency shift randomisation scheme with $L_{\text{frame}} = 645$ (2 frames), $f_{\text{init}} = 30$ and $f_{\text{shift}} = 3$. The third row shows both randomisation combined, using $L_{\text{frame}} = 645$ (2 frames), $k = 100$, a $[0.0, 1.0]$ multiplier, $f_{\text{init}} = 30$ and $f_{\text{shift}} = 3$.

amplitude output, we can generate stereo sequences by panning them left and right as the resulting signal, with the variation introduced by the band shift explained in Section III-D being slightly different for relatively small $[\text{mult}_{\text{min}}, \text{mult}_{\text{max}}]$ values.

Second, we propose another strategy to perform frame-wise pitch-shift on the output amplitudes. Again, we select a desired frame length L_{frame} and split the output amplitudes into non-overlapping frames of before the linear interpolation operation. Within that frame, we "roll" all the amplitude values to a randomised integer value $[-f_{\text{shift}}, f_{\text{shift}}]$ on their band-axis, effectively transposing the bands from one amplitude to another. We take into account the previous f_{shift} values to compute the current shift, implementing a process somewhat similar to a random walk. We also allow for an initial frequency shift f_{init} that rolls all the amplitude values in a randomised $[-f_{\text{init}}, f_{\text{init}}]$ range, effectively transposing the entire sound. Likewise, the subsequent linear interpolation operation to audio rate will provide a relatively smooth transition between shifts.

An example of both schemes is depicted in Fig. 5, showing the top k randomisation in the first row, the frequency shift randomisation in the second row and both randomisation schemes applied together in the third row.

B. *Loudness Transfer*

In [7] they were capable of performing timbre transfer (i.e., transferring the pitch and loudness of an incoming audio to the instrument the model is trained on) using just 13 minutes of expressive solo violin performances. However, unlike harmonic sounds that are constrained by a discretised and well-defined pitch, in Section IV-A we use spectral centroid as an alternative control vector for inharmonic sounds (such as in most sound effects), thus introducing a higher degree of freedom to the control parameters. Considering the deterministic nature of the output amplitudes from the model, and since obtaining enough expressive data to represent all possible loudness and spectral centroid and interactions may be challenging in the context of sound effects, here we employ a control scheme that only relies on one of the features: loudness. Our aim is to transfer the relative loudness envelope of one sound to another, training our network on the latter and using the extracted loudness envelope of the former during inference. This is possible due to loudness being mathematically defined (i.e., it can be extracted programmatically) and normalised to a [0,1] range relative to the training and inference data, as described in Section IV-A, covering the full loudness range regardless of the training data.

To demonstrate the loudness transfer capabilities of the model, we follow the same training procedure as in Section IV-A, but using loudness as the only control parameter. We train three different models on the following short sounds: a metal impact (≈1.0 s), the Wilhelm scream (≈1.2 seconds), and an electric drill sound (≈3.4 seconds). Due to having a single control parameter and therefore a single input MLP, the trained models are slightly smaller than the ones trained on two control parameters. More specifically, they have a total of ≈414 K parameters (as opposed to 464 K) and their weights are of size ≈1.6 MB (as opposed to ≈1.8 MB). For reference, the MRSTFT reconstruction loss for the different models is 1.04 ± 0.01 for the metal impact model, 1.09 ± 0.01 for the Wilhelm scream model and 1.17 ± 0.01 for the drill model using the same objective as in Section IV-A.

We then choose another three sounds to transfer their loudness envelope to all the trained models: a rhythmic beatbox sound, scribbling using a pencil onto paper sounds, and a squeaky toy sound effect. We collect all the training and inference sound effects from the Freesound website [42]. The loudness transfer is performed by simply extracting the loudness from the target sounds (beatbox, scribbling and drill in our examples) computing it using the same procedure described in Section IV-A (including normalising it to a [0,1] range), and using the resulting vector (interpolated accordingly to the internal Fs/W sampling rate) as the input to the trained models (metal impact, Wilhelm scream, and electric drill). We choose a 2^{19} sample length excerpt from the target sounds ($\approx$ 12s at $44.1\,\text{kHz}$) and apply the operation described above.

While both the loudness of the target and trained models sounds are normalised to a [0,1] range, it may happen that most (or the most perceptually relevant) of their values are clustered in a narrower interval, and outliers distort it. To solve this potential issue, and to allow for a finer control over the output of the model, we apply a user-defined scale modifier to the loudness values. In our experiments, we applied the following modifiers to the input loudness values of the sounds depicted in Fig. 6: for metal impact, $+0.1$ on beatbox, -0.1 on scribbling, -0.1 on squeaky

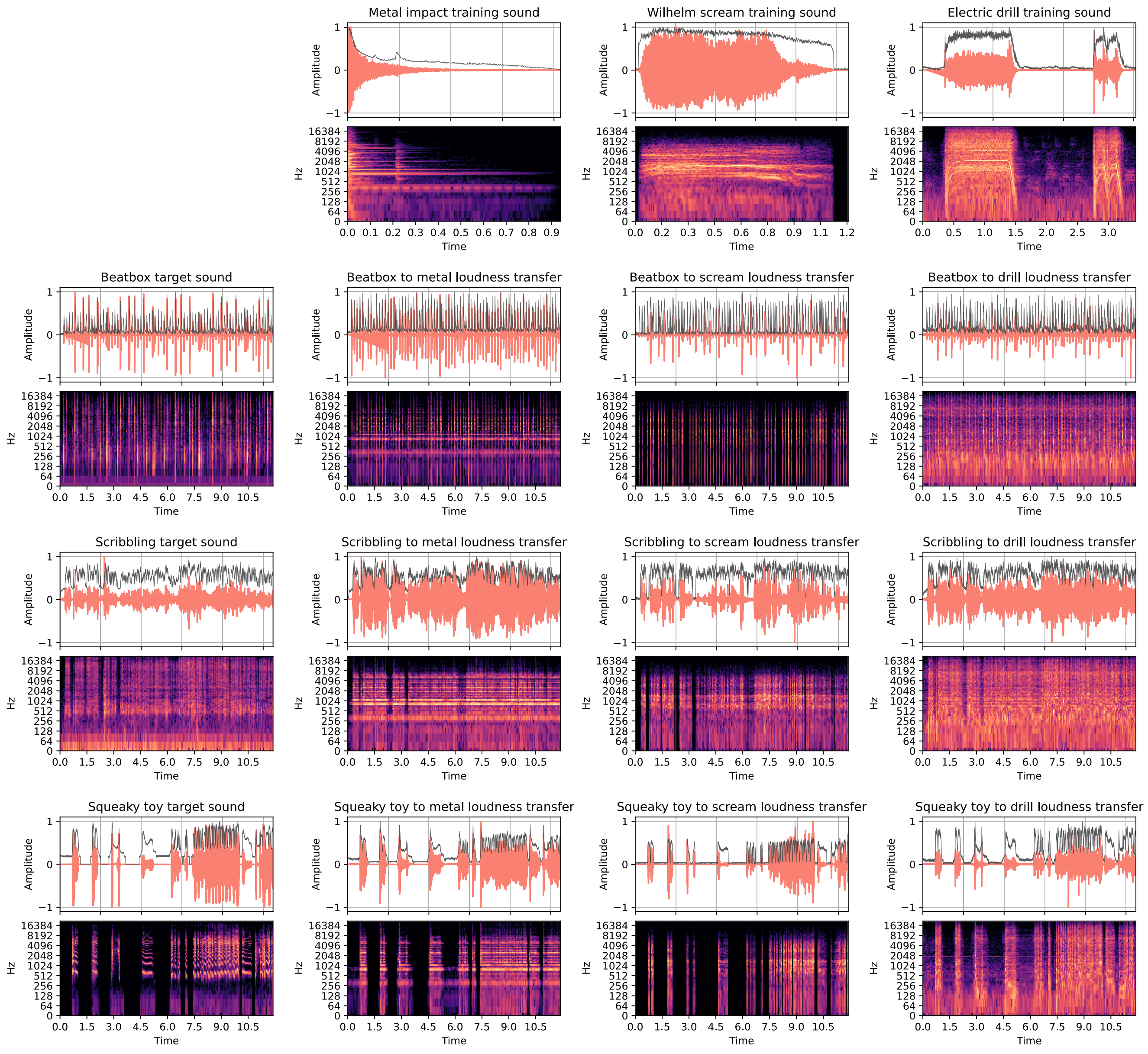

Fig. 6. Waveforms (top) and log-magnitude spectrograms (bottom) pairs resulting from the loudness transfer experiments. The extracted loudness of each sound is represented in black, superimposed on the waveforms in a [0,1] range. The first row depicts the three sounds used to train each of the models: a metal impact, the Wilhelm scream, and an electrical drill sound effect. The first column contains the sounds used for transferring their loudness envelopes. Starting from the second row, the second, third, and fourth columns contain the loudness transfer results for the different sound combinations.

toy; for Wilhelm scream, $+0.3$ on beatbox, $+0.15$ on scribbling, no modification on squeaky toy; for electric drill, $+0.25$ on beatbox, no modification in scribbling, -0.1 on squeaky toy.

The results from the experiments are depicted in Fig. 6. It can be discerned how the target loudness envelope, depicted in the first column, is transferred successfully to the trained models, depicted in the second, third, and fourth columns, starting from the second row. It is also noticeable how the frequency content of the resulting transferred sounds is time-varying, changing over time depending on the input control parameter. To further assess the success of the loudness transfer operation, we compute the Pearson correlation coefficient of the input loudness used to condition the network and the loudness extracted from the generated audio. The results are shown in Table II. There is a strong correlation between each of the pairs (i.e., close to 1),

TABLE II
PEARSON CORRELATION COEFFICIENT RESULTS OF THE LOUDNESS TRANSFER OPERATION FOR THE DIFFERENT SOUNDS CONSIDERED

	Metal impact	Wilhelm scream	Electric drill
Beatbox	0.94	0.91	0.97
Scribbling	0.94	0.91	0.97
Squeaky toy	0.97	0.93	0.98

We compare the input loudness curve used to condition the network, and the loudness extracted from the generated audio. All calculated correlation coefficients have an associated $p < .001$.

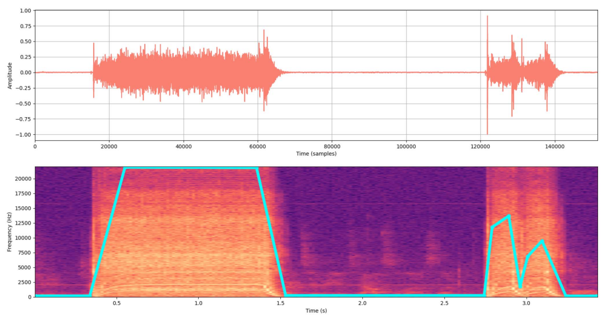

Fig. 7. Graphical user interface used to manually label the data. The image depicts the waveform (top) and the magnitude spectrogram (bottom) of an electric drill sound. The cyan line on top of the spectrogram is the hand-annotated control curve.

indicating a positive relationship between them, thus suggesting that the extracted loudness from the generated audio follows the input loudness closely.

C. Training/Synthesis With User-Defined Control Parameters

Since loudness (or spectral centroid) curves may be challenging to control and interpret from a user-perspective, or may not be adequate for the potential use cases of a particular model, here we explore training on user-provided control parameters, taking inspiration from the Wwise audio middleware Real-Time Parameter Controls (RTPCs).[5] RTPCs can be used to attach in-game parameters (e.g., speed of a car) to sound properties (e.g., pitch of the engine), linking game events to sound control curves. We anticipate a scenario in which a sound designer may wish to draw control curves over audio files during training and, once trained, draw new control curves to guide/shape the output of the model.

To this end, we design a graphical user interface to manually label the data based on potential control curves. The tool is depicted in Fig. 7, containing a waveform of the sound to be modelled at the top and its spectrogram at the bottom. By clicking on top of the spectrogram, a user can manually draw their preferred control curve. Once drawn, this curve is normalised to [0,1], in preparation for use with the model. We choose the same three sounds used in Section V-B and envisage a user with the following in mind regarding their control curves: for metal impact, the curve might control impact force; for the Wilhelm scream, the curve might control scream intensity; for the electric drill, the curve might control drill power. We then train the three models with those hand-drawn control curves as their single input control parameter. For reference, in this case the MRSTFT reconstruction loss for the different models is of 1.05 ± 0.01 for the metal impact model, 1.01 ± 0.01 for the Wilhelm scream model and 1.23 ± 0.01 for the drill model, using the same objective and configuration as in Section IV-A.

To control the synthesiser, we provide a corresponding UI tool for drawing the inference control parameters. It functions exactly as the tool depicted in Fig. 7, but now the user has a blank canvas to draw their desired control curve for driving the synthesiser. We draw three hand-crafted curves per model with a length of 2^{14} each (second, third, and fourth columns of Fig. 8). Since we upsample the output amplitudes to audio rate interpolating them by a factor defined by the synthesis window size $W = 32$, the output signal length is of $2^{14} \cdot 32 = 524288$ samples or $\approx$12 seconds at 44.1 kHz. The results of the experiments are depicted in Fig. 8. The first column contains the original sounds along their user-defined control curves used during training, represented in black on top of the waveforms. The subsequent columns are the synthesised sounds resulting from using the new user-defined inference curves, depicted also in black on top of their waveforms. For each sound category, it can be seen that the resulting audio is broadly consistent with the intended control curve.

VI. Discussion

How high-fidelity sound effects can be generated 1) automatically or 2) with dynamic or creative control where necessary or desired – all without compromising the quality or plausibility of the output audio – is a topic of interest to the fields of sound design and game audio [4], [46], psychoacoustics [36], and extended reality [3]. DDSP methods have shown promise in recent years, at least where assumptions hold regarding the harmonicity of the sounds being modeled [7]. Harmonic sounds represent only a portion of sound effects, however, and so it remained an open problem how to model and then synthesise *arbitrary sounds* with acceptable time and frequency resolution, and of arbitrary length.

The contribution of this paper is to address the modelling and synthesis of arbitrary sounds, tackling both the problem of reconstruction fidelity (see Section IV) and exploring some of the creative uses (see Section V). Our solution is encapsulated in a model called NoiseBandNet, an architecture capable of synthesising continuous sound effects conditioned on high-level parametric controls with consistently good time and frequency resolution. We propose the use of filterbanks to shape white noise, establishing a suitable approach towards modelling non-musical or inharmonic sound effects using DDSP synthesisers. NoiseBandNet is also lightweight and can be trained on very limited data ($\approx$1 s of audio), as shown in our experiments. We also highlight the potential creative uses of the architecture by generating sound variations, performing loudness transfer, and training and synthesising audio with user-defined control parameters.

We evaluated NoiseBandNet against four configurations of the original DDSP filtered noise synthesiser [7], and found that NoiseBandNet significantly outperforms all DDSP variants on the task of resynthesising sounds from different categories, for nine out of ten (sound category, evaluation metric) combinations – the exception being for the metric of FAD on pottery sounds –. In addition to overall better reconstruction capabilities compared to the original DDSP noise synthesiser, our proposed filterbank is not constrained by having its frequency response distributed linearly, such as in the case of a time-varying FIR filter. Thus, both the number of filters and their distribution across the

[5]https://www.audiokinetic.com/en/library/edge/?source=SDK&id=concept_rtpc.html

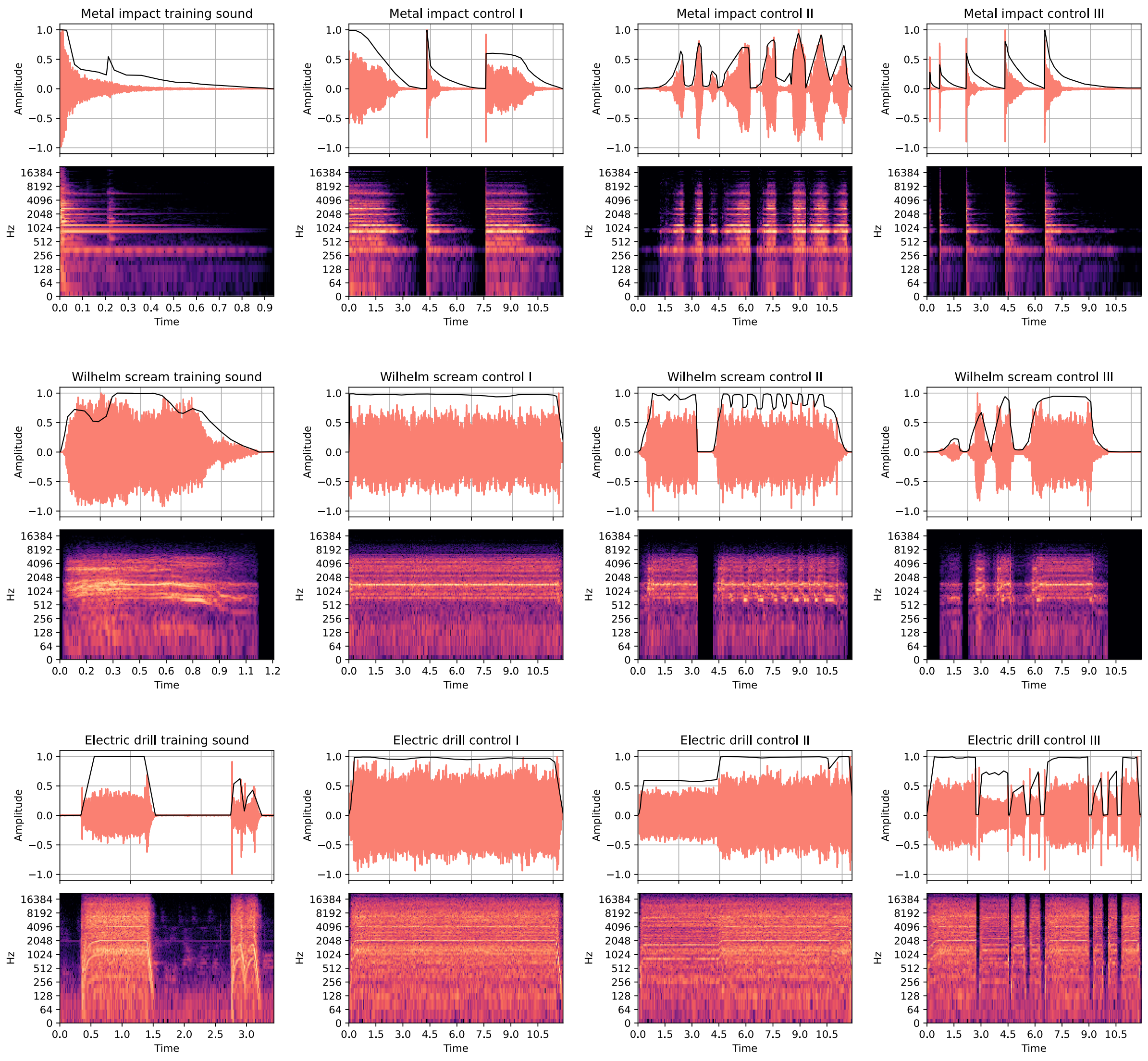

Fig. 8. Waveforms (top) and log-magnitude spectrograms (bottom) pairs resulting from the training on user-defined control experiments. The training sounds are depicted in the first column, with their user-defined training control parameters represented in black superimposed on the waveforms in a [0,1] range. The second, third, and fourth columns contain the sounds generated by using user-defined inference curves for the three models, each one in a different row. The user-defined inference control curves are also represented in black on top of the waveforms in a [0,1] range.

frequency spectrum is flexible and can be altered to accommodate other use cases.

Taking inspiration from current game audio workflows, we also outlined the creative possibilities of NoiseBandNet through a series of experiments, providing examples of amplitude randomisation, automatic loudness transfer and training models using user-defined controls. The code employed to generate those sounds alongside the audio examples described throughout the paper can be found in the accompanying material at the project website.[6]

A. Limitations and Future Work

While we used a filterbank configuration with a higher frequency resolution on the low end, broadly inspired by [2] and which provided satisfactory results on pilot experiments, the design could be further improved by considering auditory perception, for instance increasing the emphasis between 500 and 4000 Hz, where the sensitivity of frequency changes to pure tones is higher [37]. Apart from the effect of the number of filters and their distribution on synthesis quality, we also plan to explore the use of alternative loss functions, such as the differentiable joint time-frequency scattering (JTFS), used recently in the context of audio classification with promising results [47].

Since the proposed noise band structure is not tied to the network architecture itself, for future work we aim to use noise bands with other approaches. For instance, we could replace the architecture with a more lightweight temporal convolution network (TCN) [48], which has been successfully employed to model audio effects [49] and in differentiable FM synthesisers [25]. Another option may be using adversarial training [32] or

[6]https://adrianbarahonarios.com/noisebandnet/

a variational autoencoder (VAE) [50], [51], which opens up the possibility of non-deterministic behaviour. NoiseBandNet could also be applied to harmonic and musical signals, replacing the original DDSP noise synthesiser, or potentially in combination with it when sounds contain inharmonic partials (e.g., training using an approach derived from [10]).

Despite the saved model weights being small in size (≈1.8 MB and ≈1.6 MB for two and one control parameters, respectively), the size of the noise bands is large (≈1 GB on disk for our configuration), due to their the number and length. However, as described in Section III-B, thanks to the deterministic nature of the noise bands when using the same filterbank configuration, a single instance can be used across multiple models, thus only needing to create a single set of them. Nonetheless, to further optimise the model size and alleviate the computational burden involved in multiplying the output amplitudes from the model with the noise bands in the time-domain, we plan to investigate the use of neural audio codecs such as Encodec [52], and multi-rate filterbanks and sub-band processing as in [2]. As we reported in Section IV-A, the offline generation on a consumer GPU is fast (≈529.5 milliseconds to generate 30 seconds of audio), but it is much slower on CPU (≈13.4 seconds to generate the same length). While more research needs to be conducted to address these points, we hypothesise that a combination of architectural changes (such as the use of TCNs), a more efficient auditory-informed filterbank configuration, and the use of neural audio codecs and sub-band processing as described above may result in faster generation, which is especially relevant for real-time synthesis in the context of game audio.

Regarding the automatic extraction of control parameters from the audio, above (Section IV-A) we use loudness and spectral centroid, computed using DSP methods. Other approaches, such as [27], develop highly engineered solutions to extract control parameters from the target audio, such as engine RPM in their case. It is desirable, however, to accommodate a wider range of sounds and use cases. While a first approach could be the use of sound event detection to extract similar sounding clips from longer signals in data-scarce scenarios, such as in [53], achieving the potential granularity required to successfully label continuous data (e.g., drill power in our examples) may be challenging. Another direction, inspired by the recent proliferation of text-to-audio models such as [18], [19], could be the generation short audio clips catered towards being controlled by a model such as NoiseBandNet. For instance, a text-to-audio model could be prompted to generate a drill sound effect with a linearly increasing drill power, and the output audio could be used as the input to NoiseBandNet alongside a linearly increasing control parameter vector going from [0,1] (minimum and maximum drill power values), granting the text-to-audio model successfully renders a sound with those properties.

We acknowledge that while we present three different experiments exploring the creative uses of the architecture, these could be expanded and evaluated in a user study. Future work will comprise carrying out a study with audio experts to evaluate the creative possibilities of the model, and the plausibility of the synthesised sounds. The study will also inform the amount and type of data needed to satisfactorily accomplish a control task, and the feasibility of training with multiple user-defined control parameters (e.g., a weather audio model with both "rain and thunder intensity" control curves). Since NoiseBandNet uses DSP components (time-varying amplitudes applied to filters) that audio experts are familiar with, we also plan to evaluate and expand the randomisation schemes outlined in Section V-A. Ultimately, we aim to understand how the model introduced in this paper could affect the workflows of sound designers and, more generally, audio experts in years to come when using controllable neural audio synthesisers in the context of game audio.

References

[1] C. Hausman, F. Messere, and P. Benoit, *Modern Radio and Audio Production: Programming and Performance*. Boston, MA, USA: Cengage Learn., 2015.

[2] D. Marelli, M. Aramaki, R. Kronland-Martinet, and C. Verron, "Time-frequency synthesis of noisy sounds with narrow spectral components," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 18, no. 8, pp. 1929–1940, Nov. 2010.

[3] "Post-Keynote Panel: Procedural sound synthesis for AR/VR," 2022. Accessed: Oct. 7, 2023. [Online]. Available: https://www.youtube.com/live/9ngwhfF0FhA?feature=share&t=6368/

[4] A. Farnell, *Designing Sound*. Cambridge, MA, USA: MIT Press, 2010.

[5] A. Barahona-Ríos and T. Collins, "SpecSinGAN: Sound effect variation synthesis using single-image GANs," in *Proc. 19th Sound Music Comput. Conf.*, 2022, pp. 302–309.

[6] M. Comunità, H. Phan, and J. D. Reiss, "Neural synthesis of footsteps sound effects with generative adversarial networks," in *Proc. Audio Eng. Soc. Conv. 152*, 2022. [Online]. Available: https://www.aes.org/e-lib/browse.cfm?elib=21696

[7] J. Engel et al., "DDSP: Differentiable digital signal processing," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://iclr.cc/virtual_2020/poster_B1x1ma4tDr.html

[8] X. Serra and J. Smith, "Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition," *Comput. Music J.*, vol. 14, no. 4, pp. 12–24, 1990.

[9] F. Ganis, E. F. Knudesn, S. V. Lyster, R. Otterbein, D. Südholt, and C. Erkut, "Real-time timbre transfer and sound synthesis using DDSP," in *Proc. 18th Sound Music Comput. Conf.*, 2021, pp. 175–182.

[10] B. Hayes, C. Saitis, and G. Fazekas, "Sinusoidal frequency estimation by gradient descent," in *Proc. Int. Conf. Acoust., Speech Signal Process.*, 2023, pp. 1–5.

[11] J. Turian and M. Henry, "I'm sorry for your loss: Spectrally-based audio distances are bad at pitch," in *Proc. "I Can't Believe It's Not Better!" NeurIPS Workshop*, 2020. [Online]. Available: https://openreview.net/forum?id=Z4UwGkTRTes

[12] R. Selfridge, D. Moffat, E. J. Avital, and J. D. Reiss, "Creating real-time aeroacoustic sound effects using physically informed models," *J. Audio Eng. Soc.*, vol. 66, no. 7/8, pp. 594–607, 2018.

[13] R. Nordahl, S. Serafin, and L. Turchet, "Sound synthesis and evaluation of interactive footsteps for virtual reality applications," in *Proc. IEEE Virtual Reality Conf.*, 2010, pp. 147–153.

[14] P. Bahadoran, A. Benito, T. Vassallo, and J. D. Reiss, "Fxive: A web platform for procedural sound synthesis," in *Proc. AES Conv.*, 2018. [Online]. Available: https://www.aes.org/e-lib/browse.cfm?elib=19529

[15] A. Barahona-Ríos and S. Pauletto, "Synthesising knocking sound effects using conditional WaveGAN," in *Proc. 17th Sound Music Comput. Conf.*, 2020, pp. 450–456.

[16] X. Liu, T. Iqbal, J. Zhao, Q. Huang, M. D. Plumbley, and W. Wang, "Conditional sound generation using neural discrete time-frequency representation learning," in *Proc. IEEE 31st Int. Workshop Mach. Learn. Signal Process.*, 2021, pp. 1–6.

[17] S. Pascual, G. Bhattacharya, C. Yeh, J. Pons, and J. Serrà, "Full-band general audio synthesis with score-based diffusion," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2023, pp. 1–5.

[18] F. Kreuk et al., "AudioGen: Textually guided audio generation," in *Proc. 11th Int. Conf. Learn. Representations*, 2023. [Online]. Avaialble: https://openreview.net/forum?id=CYK7RfcOzQ4

[19] H. Liu et al., "AudioLDM: Text-to-audio generation with latent diffusion models," in *Proc. 40th Int. Conf. Mach. Learn.*, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., Jul. 23–29, 2023, vol. 202, pp. 21450–21474.

[20] G. Greshler, T. Shaham, and T. Michaeli, “Catch-a-Waveform: Learning to generate audio from a single short example,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 20916–20928.

[21] S. Pauletto, A. Barahona-Ríos, V. Madaghiele, and Y. Seznec, “Sonifying energy consumption using SpecSinGAN,” in *Proc. 20th Sound Music Comput. Conf.*, 2023, pp. 403–409.

[22] S. Andreu and M. V. Aylagas, “Neural synthesis of sound effects using flow-based deep generative models,” in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2022, pp. 2–9.

[23] B. Hayes, C. Saitis, and G. Fazekas, “Neural waveshaping synthesis,” in *Proc. 22nd Int. Soc. Music Inf. Retrieval Conf.*, 2021, pp. 254–261. [Online]. Available: https://doi.org/10.5281/zenodo.5624613

[24] S. Shan, L. Hantrakul, J. Chen, M. Avent, and D. Trevelyan, “Differentiable wavetable synthesis,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2022, pp. 4598–4602.

[25] F. Caspe, A. McPherson, and M. Sandler, “DDX7: Differentiable FM synthesis of musical instrument sounds,” in *Proc. 23rd Int. Soc. Music Inf. Retrieval Conf.*, Bengaluru, India, 2022, pp. 608–616.

[26] R. Diaz, B. Hayes, C. Saitis, G. Fazekas, and M. Sandler, “Rigid-body sound synthesis with differentiable modal resonators,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2023, pp. 1–5.

[27] A. Lundberg, “Data-driven procedural audio: Procedural engine sounds using neural audio synthesis,” M.S. thesis, KTH School Elect. Eng. Comput. Sci., Stockholm, Sweden, 2020.

[28] D. Serrano, “A neural analysis–synthesis approach to learning procedural audio models,” M.S. thesis, New Jersey Institute of Technology, Department of Computer Science, Newark, NJ, USA, 2022.

[29] J. Nistal, S. Lattner, and G. Richard, “DrumGAN: Synthesis of drum sounds with timbral feature conditioning using generative adversarial networks,” in *Proc. 21st Int. Soc. Music Inf. Retrieval Conf.*, Montreal, Canada, 2020, pp. 590–597.

[30] Y. Okamoto et al., “Onoma-to-wave: Environmental sound synthesis from onomatopoeic words,” *APSIPA Trans. Signal Inf. Process.*, vol. 11, no. 1, 2022, doi: 10.1561/116.00000049.

[31] A. Caillon and P. Esling, “RAVE: A. variational autoencoder for fast and high-quality neural audio synthesis,” 2021, *arXiv:2111.05011*.

[32] I. Goodfellow et al., “Generative adversarial nets,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[33] L. Wyse, P. Kamath, and C. Gupta, “Sound Model Factory: An Integrated System Architecture for Generative Audio Modelling,” in *Proc. Int. Conf. Comput. Intell. Music, Sound, Art Des.*, 2022, pp. 308–322.

[34] D. Bisig and K. Tatar, “Raw music from free movements: Early experiments in using machine learning to create raw audio from dance movements,” in *Proc. 2nd Conf. AI Music Creativity*, 2021, [Online]. Avaialble: https://aimc2021.iem.at/papers/

[35] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, “Flow synthesizer: Universal audio synthesizer control with normalizing flows,” *Appl. Sci.*, vol. 10, no. 1, 2019, Art. no. 302.

[36] J. H. McDermott and E. P. Simoncelli, “Sound texture perception via statistics of the auditory periphery: Evidence from sound synthesis,” *Neuron*, vol. 71, no. 5, pp. 926–940, 2011.

[37] A. J. Oxenham, “How we hear: The perception and neural coding of sound,” *Annu. Rev. Psychol.*, vol. 69, pp. 27–50, 2018.

[38] V. Välimäki, J. Rämö, and F. Esqueda, “Creating endless sounds,” in *Proc. Digit. Audio Effects Conf.*, 2018, pp. 32–39.

[39] D. Creasey, *Audio Processes: Musical Analysis, Modification, Synthesis, and Control*. Abingdon, U.K.: Routledge, 2016.

[40] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *Proc. Int. Conf. Learn. Representations*, 2015.

[41] R. Yamamoto, E. Song, and J.-M. Kim, “Parallel wavegan: A. fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2020, pp. 6199–6203.

[42] F. Font, G. Roma, and X. Serra, “Freesound technical demo,” in *Proc. ACM Int. Conf. Multimedia*, 2013, pp. 411–412.

[43] C. J. Steinmetz and J. D. Reiss, “Auraloss: Audio-focused loss functions in PyTorch,” in *Proc. Digit. Music Res. Netw. One-day Workshop*, 2020. [Online]. Available: https://www.qmul.ac.uk/dmrn/media/dmrn/DMRN-15_proceedings.pdf

[44] D.-Y. Wu et al., “DDSP-Based singing vocoders: A new subtractive-based synthesizer and a comprehensive evaluation,” in *Proc. ISMIR Hybrid Conf.*, 2022, pp. 76–83.

[45] K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi, “Fréchet audio distance: A metric for evaluating music enhancement algorithms,” in *Proc. Interspeech*, 2019, pp. 2350–2354.

[46] G. Zdanowicz and S. Bambrick, *The Game Audio Strategy Guide: A. Practical Course*. Waltham, MA, USA: Focal Press, 2019.

[47] J. Muradeli et al., “Differentiable time-frequency scattering on GPU,” in *Proc. Digit. Audio Effects Conf.*, 2022, pp. 280–287.

[48] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” 2018, *arXiv:1803.01271*.

[49] C. J. Steinmetz and J. D. Reiss, “Efficient neural networks for real–time modeling of analog dynamic range compression,” in *Proc. Audio Eng. Soc. Conv. 152*, 2022. [Online]. Avaialble: https://www.aes.org/e-lib/browse.cfm?elib=21709

[50] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *Proc. 2nd Int. Conf. Learn. Representations*, Banff, AB, Canada, Apr. 14–16, 2014.

[51] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1278–1286.

[52] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, “High fidelity neural audio compression,” *Trans. Mach. Learn. Res.*, 2023. [Online]. Avaialble: https://openreview.net/forum?id=ivCd8z8zR2

[53] Y. Wang, J. Salamon, N. J. Bryan, and J. P. Bello, “Few-shot sound event detection,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2020, pp. 81–85.