STREAMABLE NEURAL AUDIO SYNTHESIS WITH NON-CAUSAL CONVOLUTIONS

Antoine Caillon and Philippe Esling

IRCAM - CNRS UMR 9912
Sorbonne Université, Paris, France
`caillon@ircam.fr | esling@ircam.fr`

ABSTRACT

Deep learning models are mostly used in an offline inference fashion. However, this strongly limits the use of these models inside audio generation setups, as most creative workflows are based on real-time digital signal processing. Although approaches based on recurrent networks can be naturally adapted to this buffer-based computation, the use of convolutions still poses some serious challenges. To tackle this issue, the use of *causal streaming* convolutions have been proposed. However, this requires specific complexified training and can impact the resulting audio quality.

In this paper, we introduce a new method allowing to produce *non-causal streaming* models. This allows to make any convolutional model compatible with real-time buffer-based processing. As our method is based on a post-training reconfiguration of the model, we show that it is able to transform models trained without causal constraints into streaming models. We apply our method on the recent RAVE model as a case study. This model provides high-quality real-time audio synthesis on a wide range of signals and thus is an ideal candidate to evaluate our method. It should be noted that our method is not restricted to RAVE, and can be straightforwardly applied to any convolutional network. We test our approach on multiple music and speech datasets and show that it is faster than overlap-add methods, while having no impact on the generation quality. Finally, we introduce two open-source implementation of our work as Max/MSP and PureData externals, and as a VST audio plugin. This allows to endow traditional digital audio workstations with real-time neural audio synthesis.

1. INTRODUCTION

Neural audio signal processing has set a new state of art in many fields, such as audio source separation [1], text-to-speech [2], timbre transfer [3] and unconditional generation [4]. Recent works on neural audio synthesis such as DDSP [3], melGAN [5] or RAVE [6] have allowed to perform deep audio synthesis faster than realtime. Those methods pave the way towards the integration of neural synthesis and processing inside real-time audio applications.

Amongst these, models based on recurrent layers (DDSP [3] or RNNoise [7]) are built to process time series sequentially. Therefore, they are naturally fit to process live audio streams by caching their recurrent state in-between DSP calls. However, this is not the case for models based on convolutional networks [8] since their reliance on *padding* causes audible phase discontinuities between consecutive audio buffers (e.g clicks), which prevents their use for real-time audio applications. A simple solution to address

this problem would be to rely on the *overlap-add* method, where we process large overlapping audio buffers and cross-fade them to smooth out phase discontinuities. While this method is straightforward compatible with any generative model, processing overlapping buffers leads to redundant computations and degraded quality during transition phases. In addition, this method requires caching buffers that are large enough to fill the receptive field of the model in order to avoid edge effects. This results in a high latency between the input and output of the model during inference. A more specific solution have been proposed through the idea of *streaming* models [9, 10] that use *causal* convolutional layers. These layers replace padding during inference with a cached internal or external state. Although this mechanism allows the use of convolutional models on live audio streams, it usually degrades the model accuracy due to the aforementioned causal constraint.

In this article, we propose a method to make non-causal convolutional neural networks streamable without impacting the audio quality nor introducing computational redundancies. We achieve this by making the model causal *after training*[1], leveraging additional internal delays in order to preserve the original computational graph of the model. Hence, our method can be applied over models that were already trained in a non-causal way. As an application case, we use our method to make the recent RAVE model [6] streamable in real-time. However, our approach can be applied straightforwardly to any convolution-based model. We compare our method with several *overlap-add* alternatives using both quantitative and qualitative metrics. We demonstrate that our method outperforms all other baselines in inference speed, while behaving exactly like the original model in terms of audio quality. Finally, we develop several applications leveraging our method to provide regular digital audio workstations with real-time neural audio processing abilities. All of our experiments, methods and source code are packaged as an open-source Python library available online[2].

Our contributions are:

- A method to make non-causal convolutional layers streamable, widening the range of deep learning models compatible with streaming inference

- The introduction of additional delays inside the converted model aiming at preserving its original computational graph

- An open source Python library to ease the creation of streamable models

- Several applications allowing realtime interactions between the user and streamable models

---

[1]We refer to this as *post-training causal reconfiguration* in opposition to training a causal model, i.e that uses *pre-training causal configuration*.
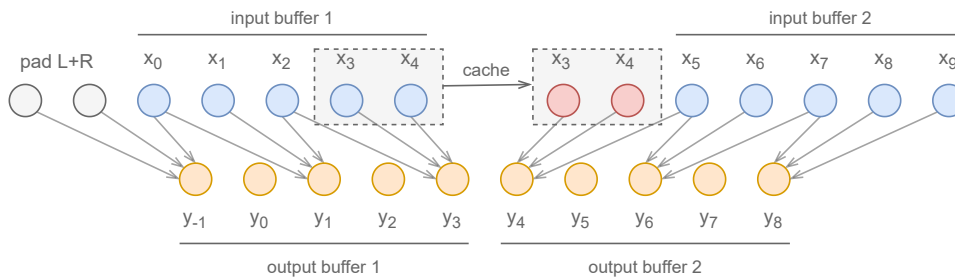
[2]`https://acids-ircam.github.io/cached_conv`

Figure 1: *Convolution applied on two split buffers using cached padding. Input and output buffers* $(\mathbf{x}, \mathbf{y})$ *respectively live in* $\mathbb{R}^{M \times T}$ *and* $\mathbb{R}^{N \times T}$, *however we omit the channel dimensions* (M, N) *in the figure for simplicity. The last* P *frames from input buffer 1 are cached and concatenated with the input buffer 2 (with* P *being the original amount of zero padding) in order to prevent discontinuities between buffers.*

2. STATE OF ART

2.1. Convolutional Neural Networks

We consider a 1-dimensional convolutional layer with a kernel $\omega \in \mathbb{R}^{N \times M \times K}$ applied on an input tensor $x \in \mathbb{R}^{M \times T}$. The resulting tensor y is defined by

$$\mathbf{y}^n[i] = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} \omega^{n,m}[k] \mathbf{x}^m[i+k] \qquad (1)$$

where $\mathbf{y} \in \mathbb{R}^{N \times T-K+1}$. Due to the size of the kernel ω, the temporal size of y is smaller than the input x. When stacking convolutional layers, this can lead to a significant dimensionality reduction that may be unwanted. To tackle this issue, convolutional layers are often used in combination with zero-*padding*. Padding is used to artificially augment the dimensionality of a tensor in order to prevent the loss of dimensionality induced by a convolution with a kernel larger than 1. As an example, in Equation (1), padding the input tensor x with $K - 1$ zeros prior to the convolution results in an output tensor y whose temporal dimensionality is the same as the original input. We call *left-padding* (resp. *right-padding*) the padding of the left-hand side (resp. right-hand side) of the tensor.

Using padding is useful to maintain a tensor dimensionality across layers. However, there are situations where an increase or decrease in temporal dimensionality is required. Convolutional layers with a stride $s > 1$ allow to decrease a tensor dimensionality by a factor s using the same padding strategy as regular convolutional layers. On the other hand, transposed convolutional layers can be used to increase a tensor temporal dimensionality.

2.2. Causal streaming models

Processing audio buffers one after the other using a convolutional neural network is not trivial. Indeed, the use of padding in each layer of the model creates discontinuities in the data when processing two consecutive buffers sequentially. In the context of neural audio synthesis, and more specifically raw waveform modelling, this causes audible phase discontinuities that are not acceptable for real-time audio applications. To address this problem, Rybakov et al. [9] proposed to rely on *causal* Convolutional Neural Networks (CNN) altogether with a *cached padding* mechanism.

Cached padding is implemented by retaining the end of one tensor and using it to left-pad the following one, as shown in Figure 1. This allows to maintain continuity between the computation of two consecutive audio buffers. It is meant to be used as a replacement for left-padding during inference, retaining the original padding increase in dimensionality without creating discontinuities in-between buffers. Although this method provides a solution for the use of CNN in real-time audio generation, it is constrained by the strict necessity of using only causal convolutions, which are not widespread. As most CNN in the literature do not satisfy this assumption, it is uncertain if this method can be widely applied to previous architectures or pre-trained models. Finally, it has been shown that a causal constraint makes the learning process more complex [9], which could impact the final audio quality.

2.3. RAVE

The RAVE model [6] is a variational auto encoder [11] applied directly to the raw audio waveform. It is trained using two separate stages, respectively named *representation learning* and *adversarial fine tuning*. The representation learning stage uses a spectral distance between the input and output of the model as its main training objective. The encoder is regularised with a standard Kullback Leibler divergence between the posterior distribution and an isotropic normal distribution. In order to keep the learned representation as compact as possible, the encoder is only trained during the first stage. During the second stage, the model is trained using elements from generative adversarial networks [12] to improve its synthesized audio quality. A post-training analysis of the latent space is performed as a way to reduce the number of useful latent dimensions. This allows an easier exploration and manipulation of the latent space. Overall, RAVE can be used to perform timbre transfer, latent manipulation and unconditional generation with unprecedented quality while synthesizing 20 to 80 times faster than real-time on a laptop CPU.

RAVE is a feed-forward model, composed of an encoder (a strided convolutional network), and a decoder (a residual transposed convolutional network). The model also implements the noise synthesizer from the DDSP model [3] to increase its synthesis quality when processing noisy signals. It leverages zero-padding to maintain the temporal dimensionality of the tensors across convolutional layers. Therefore, this model in its current state cannot be used to perform streaming inference, and is solely usable on pre-recorded audio files. Nevertheless, its feed-forward architecture and adversarial fine-tuning makes it a perfect candidate for the streaming task as it is both fast and high quality. For more detailed information about this model, we invite the reader to see the original article [6].

3. NON-CAUSAL STREAMING MODELS

The streaming models obtained following the method described in Section 2.2 can readily process live audio streams. However, this requires models that use only causal convolutions, which is not the case for most models proposed in the literature. Indeed, training a model causally can lead to a loss of accuracy or audio quality [9].

Here, we introduce our method that allows to make *non-causal* models streamable. Our proposal is constructed around the idea of performing a post-training causal reconfiguration of the model. This allows to consider convolutional networks trained using any type of padding (potentially non-causal) and turn them into streamable models. One idea to do so would be to extend the cached padding mechanism to right-padding. However, this is not possible by nature, as we are processing live audio streams where the next buffer is not known yet.

Therefore, we propose to reconfigure the model as causal *after training*. This can be achieved by transforming right-padding into an additional left-padding. While this reconfiguration allows the use of a cached padding mechanism, making the model causal after training alters its computational graph. Hence, this might produce unpredictable results if the model includes strided convolutions or has a computational graph with parallel branches (e.g residual connections [13]). In those cases, we propose the introduction of *additional delays* to restore the original behavior of the model. In the following, we detail how we address each of these architectures, in order for our method to be applicable universally on any type of network.

3.1. Aligning strided convolutions

Strided convolutions are often used as a way to reduce the temporal or spatial dimensionality of an input tensor. This is done by skipping some steps in the application of the convoluted kernel, as depicted in Figure 2.
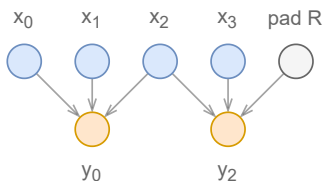

Figure 2: A simplified view of a strided convolution using zero-padding during training.

Transforming right-padding to left-padding shifts the input tensor to the right (i.e adds a lag to the input tensor). This has no consequence for convolutions with stride 1 or transposed convolutions as it only delays the output tensor. However, this lag may have an impact on convolutions with a stride greater than one, where a lag of n samples on the input tensor results in a fractional lag of n/s in the output tensor. We show in Figure 3 how this fractional lag results in a change of behavior of the layer whenever n is not a multiple of s.

Therefore, we introduce an additional delay to the input in order to make its overall lag a multiple of the stride of the convolutional layer, as shown in Figure 4. In the case of a complex convolutional network, it is necessary to keep track of the overall cumulative lag for an input tensor after each convolutional layer.
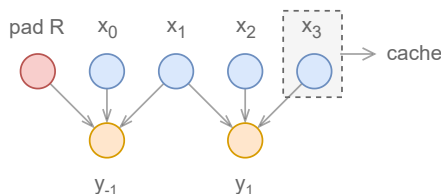

Figure 3: A strided convolution with post-training causal reconfiguration. Due to the input lag, the output of the layer is not the same as during training (see Figure 2 for the regular output).
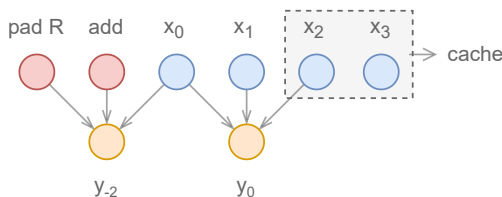

Figure 4: An additional delay (*add*) is applied to the input tensor in order to recover the original behavior of the layer.

Considering that a convolutional layer with stride S and right-pad length R processes an input tensor with cumulative delay D_c, we need to set the additional delay D_a to

$$D_a = S - (R + D_c \mod S) \mod S, \qquad (2)$$

where the cumulative delay D_c is set to 0 at the beginning of the graph, and is updated at each layer according to

$$D_c \leftarrow \frac{D_c + R + D_a}{S}. \qquad (3)$$

3.2. Aligning parallel branches

When designing neural networks, it has been shown that using residual or skip connections facilitate the training of deeper models [13, 14, 15]. Those connections are responsible for the creation of parallel branches in the computational graph of the model, which are later gathered (e.g summed, concatenated, ...) in order to produce an output. However, the delays introduced by a post-training
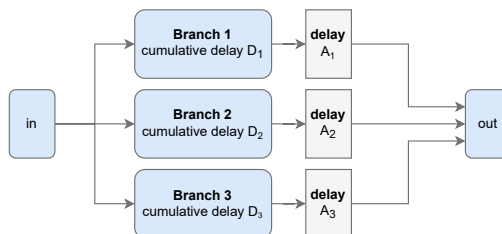

Figure 5: Aligning parallel branches using additional delays.

causal re-configuration of the model might result in a relative misalignment between the outputs of each parallel branch. Therefore models implementing parallel branches must introduce additional delays in order to recover a proper alignment between the different

branches, as shown in Figure 5. In this case, we set the additional delays A_i to

$$A_i = \max_j D_j - D_i, \qquad (4)$$

where D_i is the cumulative delay of the ith branch.

3.3. Overlap-add baseline

For comparison purposes, we use a simple yet effective baseline method to process live audio streams with non-causal convolutional neural networks. We implement the *overlap-add* method by first collecting an audio buffer large enough to account for the receptive field of the model. Then, we apply the unmodified convolutional neural network on this buffer and window the output signal using the Hann window

$$\mathbf{w}[n] = \sin\left(\frac{\pi n}{N}\right)^2,$$

where N is the buffer size.

Finally, we add the resulting tensor to the previous output with a temporal offset of $N/2$. This implements the overlap-add method with a 50% overlapping factor. We compare this method to another having a 25% overlapping ratio, implemented by scaling w accordingly, as depicted in Figure 6. This reduces the computational redundancy of the method and consequently makes it process audio faster. However, using a smaller overlapping window results in harsher transitions between buffers. Hence, we also consider the extreme case of a 0% overlapping factor, where the model is applied on non-overlapping buffers. This last configuration can be seen as an ablation of our method where cached padding and causal constraints are removed.
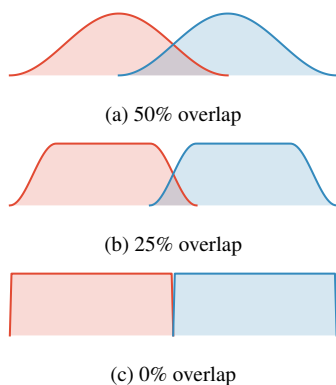

(a) 50% overlap

(b) 25% overlap

(c) 0% overlap

Figure 6: Windows used by the three overlap-add baseline variants implemented.

4. EVALUATION

In order to evaluate our method, we apply it on the RAVE model [6] by replacing the convolutions in the regular model with *cached convolutions*[3]. No other changes have been made to the model. As we are evaluating our method on an audio synthesis task, we

are interested in its computational efficiency compared to baselines, which we cover in Section 4.1. We evaluate the impact of using causal convolutions on the synthesized audio in Section 4.2 and 4.3. Finally, we verify in Section 4.4 that our method does not change the behavior of the model by comparing audio signals synthesized by a non-causal RAVE both in regular and streaming mode.

4.1. Performances

In this section, we evaluate the performances of our proposed streaming method when applied to a non-causal RAVE model compared to different variants of the overlap-add method. The model we use for this purpose is randomly initialized as the value of its parameters do not affect its computational complexity. We keep the default hyperparameters of the original article.

In order to evaluate the inference speed, we rely on the Real-Time Factor (RTF) defined as the ratio between processing time and audio duration when performing and encode-decode pass on an audio signal. A RTF below 1 indicates that the algorithm processes data *faster* than real-time. We also evaluate the amount of memory required during inference on live audio streams, by analyzing the Random Access Memory (RAM) usage[4]. We estimate both memory usage and RTF of the reconstruction process using the various methods applied to 60s long random (white noise) audio signals with varying buffer sizes. We rely on white noise as here the audio output is not relevant to compute the speed of different methods. All results are averaged over 10 trials in order to account for measurement errors.

We show in Figure 7a how our proposed *streaming* and different *overlap-add* methods all have a similar memory usage. The only difference comes from a constant 180kiB of additional RAM needed to store the cached padding of the streaming method.

In terms of processing speed, as we can see in Figure 7b, the overlap method with a 0% overlap ratio is the fastest, while also being the less accurate (see Section 4.4). Although increasing the overlap ratio to 25% or 50% can reduce the corresponding artifacts, it also makes the overlap method increasingly slower than the streaming method. This is due to the computational redundancies involved in this method.

4.2. Processing latency

Introducing delays in the computational graph of a model has an impact on its processing latency. Causal models have the lowest latency, as they only rely on past audio samples when generating an output. Oppositely, non-causal models also leverage future audio samples which, in the context of live audio processing, are not available *yet*. We call the area of the receptive field of the model leveraging future samples its *future receptive field*. While our method allows the use of non-causal models on live audio streams, it also implies a processing latency equivalent to the duration of the future receptive field of the model. In the case of the RAVE model, this latency adds up to ~ 600ms compared to only ~ 50ms when using RAVE trained with a causal constraint.

4.3. Impact of pre-training causal constraint

As discussed in Section 2.2, enforcing a causal constraint on the model prior to its training can complexify the modelling task. We

---

[3]We release the code for cached convolutions here `https://acids-ircam.github.io/cached_conv/`

[4]We use the `mprof` Python package for memory usage estimation.
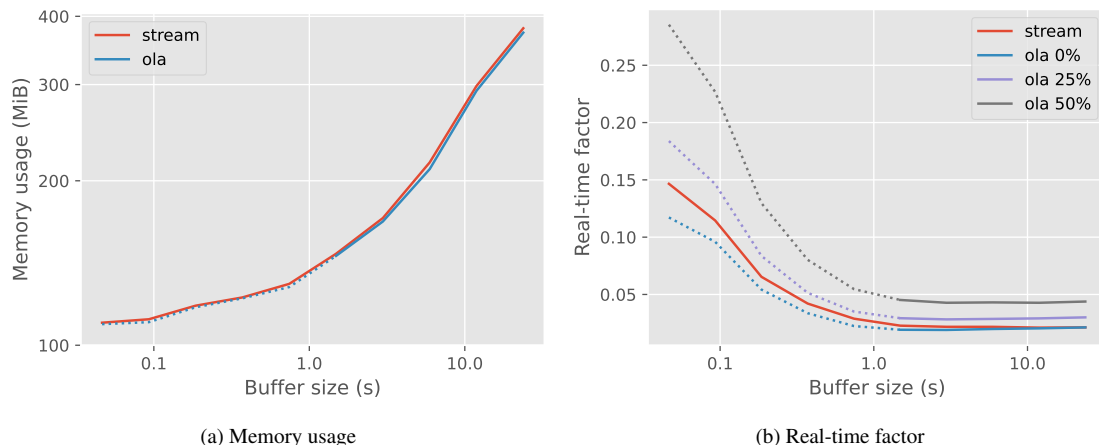
(a) Memory usage

(b) Real-time factor

Figure 7: Memory usage and real-time factor for the *streaming* and *overlap-add* methods on a regular RAVE model with varying buffer size. Memory usage is identical for all overlap-add methods. Dotted lines indicate that the model is applied on buffers smaller than its receptive field.

evaluate the impact of this constraint on the RAVE model trained with the following internal datasets

Darbuka. It has been shown that modelling percussive sounds using a causal model can be difficult [16]. Therefore, we rely on a dataset composed of various solo darbuka performances sampled at 44.1kHz, with a total duration of approximately 3 hours.

Strings. This dataset contains approximately 30 hours of various strings recordings sampled at 44.1kHz that were scraped from different real-life solo violin performances. Compared to the darbuka, it is composed of harmonic signals with smoother attacks.

Speech. The speech dataset is composed of approximately 8 hours of recordings sampled at 44.1kHz. All recordings are produced by a single speaker in a consistent acoustic environment.

All datasets are split into 90%-10% train and validation sets. We use all the augmentation strategies proposed in the original article [6]. We train two variants of the RAVE model for each dataset (pre-training and post-training causal re-configuration). All models are trained for 2M iterations. In order to measure the reconstruction error of audio samples from the validation set as input for a pretrained RAVE model, we use the following spectral distance

$$\mathcal{L}_s(\mathbf{x}, \mathbf{y}) = \| \log(S(\mathbf{x}) + \epsilon) - \log(S(\mathbf{y}) + \epsilon) \|_2, \quad (5)$$

where $S(\mathbf{x})$ is an amplitude Short-Term Fourier Transform with a window of size 2048 and an overlap ratio of 75%. We set $\epsilon = 1$ as proposed by Défossez et al. [17]. We report the means and standard deviations of the resulting spectral distances in Table 1.

Using the pre-training causal configuration results in a small but consistent loss of accuracy as compared to the regular training of models across all datasets. This confirms that using causal networks in the context of neural audio synthesis complexifies the learning process. We hypothesize that this loss of precision is due to the fact that anticipating a transition or articulation in the sound is made more difficult when the model cannot look ahead.

Table 1: *Reconstruction errors for pre-training and post-training causal reconfiguration across different datasets.*

	pre-training	post-training
Darbuka	0.228 ± 0.028	$\mathbf{0.178 \pm 0.038}$
Strings	0.055 ± 0.012	$\mathbf{0.054 \pm 0.011}$
Speech	0.155 ± 0.005	$\mathbf{0.138 \pm 0.005}$

4.4. Fidelity

In contrast to our proposed streaming method, the *overlap-add* approach only yields an approximation of the original model. Hence, we aim to estimate the quality of this approximation by comparing signals coming from the overlap-add method with signals processed offline by a non-causal model. In addition to the spectral distance defined in Section 4.3, we use the following metric

$$\mathcal{L}_w(\mathbf{x}, \mathbf{y}) = \| \mathbf{x} - \mathbf{y} \|_2, \quad (6)$$

where $\mathcal{L}_w$ is the Euclidean distance between two raw waveforms. While the spectral distance is useful to assess how perceptually similar two audio signals are regardless of their phase, the waveform Euclidean distance is highly phase-dependent, and reflects a sample-wise dissimilarity between the raw waveforms. Combined, those two metrics give us insights about how similar signals are both from a perceptual and sample-wise point of view.

We disable the noise synthesizer and set the encoder variance to 0 in order to make the model behave predictably. This is necessary as any randomness involved in the generation process would bias the fidelity measure.

We compare the *overlap-add* methods with several overlapping ratios (0%, 25% and 50%), and also include our *streaming* method to ensure that it is an exact reproduction of the offline method. We compensate the latency present in the synthesized outputs for all methods prior to their evaluation. We use the three pre-trained non-causal RAVE models described in Section 4.3 and compute an average of both the spectral and Euclidean distances over all corresponding validation sets. We report the results for all methods with varying buffer sizes in Figure 8.

(a) Spectral distance
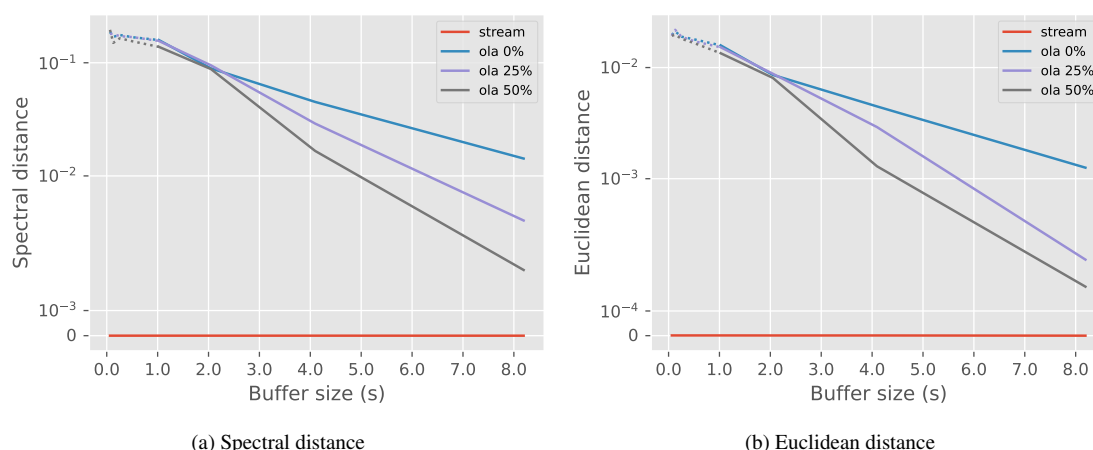

(b) Euclidean distance

Figure 8: Spectral and euclidean distances between different *overlap-add* processing methods (ola) and the offline processing method as a function of the buffer size. Dotted lines indicate that the model is applied on buffers smaller than its receptive field.

As we can see, all variants of overlap-add methods have a decreasing spectral and Euclidean distances to the offline method as the buffer size increases. However, those distances never become null even for buffer sizes larger than 8s, stressing out the artifacts introduced by such methods. Oppositely, our streaming method is exactly identical to the offline method, regardless of the buffer sizes. This confirms that the cached padding and post-training causal reconfiguration of the model allow its use on live audio streams without altering the quality of the output.

5. APPLICATION

While recent works in deep generative modelling have yielded impressive results in audio processing, the high complexity inherent to deep convolutional neural networks have restricted their deployment to technical demonstrations, command line utilities or web based interfaces. The introduction of *streaming models* [9, 10] has paved the way toward the use of deep learning algorithms inside real-world realtime applications, at the expense of a strict causal constraint on the convolutional layers. Our method for converting non-causal models into streaming ones lifts this constraint, allowing to use *any* convolutional neural network in realtime.

Therefore, we introduce alongside this article several applications leveraging our streaming method, proposing user-friendly interfaces and realtime control over pre-trained models. This allows the use of state-of-the-art techniques for various audio processing tasks directly inside regular digital audio workstations. The source code and pre-built binaries for all applications are available online[5].

5.1. Max/MSP and PureData externals

We introduce the *nn~* external for Max/MSP and PureData. This external allows the use of deep learning streaming models to process audio signals inside both applications. It leverages pre-trained models exported as *torchscript* files. By default, the *nn~* external uses the *forward* method of the model. However, it is possible to specify another method by passing an additional argument to the

---

[5] https://acids-ircam.github.io/cached_conv

external during its initialization. The number of inlets and outlets of the external depends on both the model and the method used. For example, the *forward* method of the RAVE model uses one inlet and one outlet, as both the input and output of this method are monophonic audio signals. However, choosing the *encode* method will create one inlet and N outlets, as the input of this method is a monophonic audio signal, while its output is a N-dimensional latent representation. Tensors with a lower sampling rate than audio signals are up-sampled at the audio rate using nearest neighbour interpolation. This method of interfacing N-dimensional tensors as audio signals give the user a lot of flexibility, as each individual dimension can be modified in real-time. To examplify this, we show in Figure 9 an example Max/MSP patch where the first and last dimensions of the latent representation yielded by a RAVE model are respectively biased and replaced by a user defined input.

This implements the high-level manipulation showcased in the original article [6], but also extended by allowing real-time interaction with the generative process. Overall, the *nn~* external can be used to combine deep learning streaming models with the large library of objects already available in both MaxMSP and PureData. Demonstration videos of nn used in combination with RAVE can be found on our supporting website.

5.2. VST audio plugin

As an alternative to the *nn~* external, we propose a VST audio plugin interfacing the RAVE model in order to expand its use to regular digital audio workstations supporting the VST3 plugin format. Our plugin is based on the JUCE framework for both the graphical interface and the audio engine. We depict a screenshot of the plugin in Figure 10.

We generate a latent path, either by using the RAVE encoder on an audio input, or by sampling from the prior of the model. This latent path is then displayed as a circular graph (see Figure 10), where each point corresponds to a single latent dimension. As the latent distribution produced by RAVE is close to a normal distribution, we define the distance d_i of each point from the center of
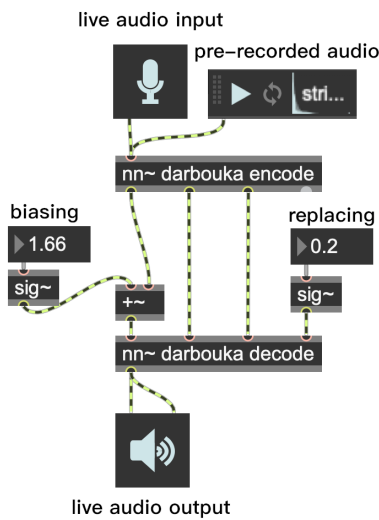
Figure 9: Screenshot of the *nn~* external interfacing a RAVE model trained on a darbuka dataset. Using either a live audio stream or a pre-recorded audio file as an input, the nn~ external allows to modify the latent representation yielded by the encoder in real-time. In this example, the first (resp. last) dimension of the latent space is biased (resp. replaced) by a user defined scalar.

the graph using the following cumulative distribution function

$$d_i = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{z_i}{\sqrt{2}}\right)\right), \tag{7}$$

where erf is the Gauss error function and z_i is the value of the ith dimension of the latent representation. Applying Equation (7) to a random variable x sampled from a normal distribution $\mathcal{N}(0; 1)$ results in a uniformly distributed value between 0 and 1. We give the user the possibility to apply a scale and bias as well as a random noise to each individual dimension. The resulting latent representation is then duplicated and fed to the decoder in order to produce a fake stereo image whose width can be adjusted by the user. We also provide several pre-trained models available in a *model explorer*, where other models will be added over time.

6. CONCLUSION AND FUTURE PERSPECTIVES

In this paper, we introduced a novel method allowing to transform any convolutional network for audio generation into a streamable model compatible with real-time buffer-based digital signal processing. We showed that our method can be applied on already-trained model by introducing a post-training causal reconfiguration. By carefully handling delays, we showed that this method easily extends to complex architectures with parallel branches. By comparing our method on several speech and music datasets, we showed that it provides faster computation and has no impact on the resulting audio quality. Finally, we released several implementations using our method to provide realtime CNN processing inside digital audio workstations. We hope that this work will pave the way towards the broader integration of the extensive possibilities offered by neural audio synthesis inside creative workflows.
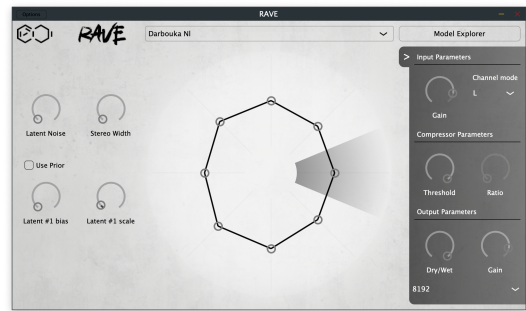

Figure 10: Screenshot of the RAVE VST interfacing a RAVE model trained on a darbuka dataset.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation. *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*, pages 334–340, 6 2018.

[2] Wei Ping, Kainan Peng, and Jitong Chen. ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech. Technical report, 9 2018.

[3] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable Digital Signal Processing. Technical report, 9 2019.

[4] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A Generative Model for Music. *arXiv*, 4 2020.

[5] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville. MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis. *arXiv*, 10 2019.

[6] Antoine Caillon and Philippe Esling. RAVE: A variational autoencoder for fast and high-quality neural audio synthesis. 11 2021.

[7] Jean-Marc Valin. RNNoise , 2017.

[8] Yann Lecun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In Michael Arbib, editor, *The handbook of brain theory and neural networks*. MIT Press, 1995.

[9] Oleg Rybakov, Natasha Kononenko, Niranjan Subrahmanya, Mirko Visontai, and Stella Laurenzo. Streaming keyword

spotting on mobile devices. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2020-October:2277–2281, 5 2020.

[10] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An End-to-End Neural Audio Codec. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 30:495–507, 7 2021.

[11] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 12 2014.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, 6 2014.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:770–778, 12 2015.

[14] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. 9 2016.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9351, pages 234–241. Springer Verlag, 2015.

[16] Sicong Huang, Qiyang Li, Cem Anil, Xuchan Bao, Sageev Oore, and Roger B. Grosse. TimbreTron: A WaveNet(CycleGAN(CQT(Audio))) Pipeline for Musical Timbre Transfer. *7th International Conference on Learning Representations, ICLR 2019*, (2), 11 2018.

[17] Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Léon Bottou, and Francis Bach. SING: Symbol-to-Instrument Neural Generator. *Advances in Neural Information Processing Systems*, 2018-December:9041–9051, 10 2018.