FM Tone Transfer with Envelope Learning

Franco Caspe
f.s.caspe@qmul.ac.uk
Centre for Digital Music,
Queen Mary University of London
United Kingdom

Andrew McPherson
andrew.mcpherson@imperial.ac.uk
Dyson School of Design Engineering,
Imperial College London
United Kingdom

Mark Sandler
mark.sandler@qmul.ac.uk
Centre for Digital Music,
Queen Mary University of London
United Kingdom

ABSTRACT

Tone Transfer is a novel deep-learning technique for interfacing a sound source with a synthesizer, transforming the timbre of audio excerpts while keeping their musical form content. Due to its good audio quality results and continuous controllability, it has been recently applied in several audio processing tools. Nevertheless, it still presents several shortcomings related to poor sound diversity, and limited transient and dynamic rendering, which we believe hinder its possibilities of articulation and phrasing in a real-time performance context.

In this work, we present a discussion on current Tone Transfer architectures for the task of controlling synthetic audio with musical instruments and discuss their challenges in allowing expressive performances. Next, we introduce Envelope Learning, a novel method for designing Tone Transfer architectures that map musical events using a training objective at the synthesis parameter level. Our technique can render note beginnings and endings accurately and for a variety of sounds; these are essential steps for improving musical articulation, phrasing, and sound diversity with Tone Transfer. Finally, we implement a VST plugin for real-time live use and discuss possibilities for improvement.

CCS CONCEPTS

• **Applied computing** → **Sound and music computing**; • **Computing methodologies** → **Neural networks**; • **Human-centered computing** → **Interaction techniques**.

KEYWORDS

Neural Networks, Synthesis Control, Musical Instrument Interaction, Mapping

ACM Reference Format:
Franco Caspe, Andrew McPherson, and Mark Sandler. 2023. FM Tone Transfer with Envelope Learning. In *Audio Mostly 2023 (AM '23), August 30–September 01, 2023, Edinburgh, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3616195.3616196

1 INTRODUCTION

Synthesizers can be very expressive instruments, whether controlled by the ubiquitous keyboard [28], by augmented instruments, or instrument-like interfaces [1, 25, 26], or by whole new sets of gestures enabled by novel controllers [8, 35].

Recent developments in integrating Deep Neural Networks (DNNs) with audio generators have renewed interest in using the unaltered audio of a musical instrument as a control source for a synthesizer. One such example is the DDSP architecture and its derivatives [5, 12, 16, 34], that allows for real-time control of a synthesizer using a set of features extracted from an input audio signal. It has been used to develop various creative timbre transformation applications, which we collectively refer to as Tone Transfer applications. [3, 4, 24, 30].

We situate the scope of our work on audio-based synthesis control for real-time performances, looking at sonic diversity and synthesizer phrasing and articulation. These essential components of musical expression have been thoroughly studied for composition with MIDI for decades [2, 43, 44] but we argue that they open new challenges and possibilities when considering an audio-based control approach. Transients at the beginnings of notes and the transitions between notes play a vital role in defining the continuity and flow of musical phrasing. We argue that a continuous control approach such as Tone Transfer could potentially learn mappings that capture beginnings, endings, and the links between notes during performance, generating musically articulated synthetic sounds.

In this work, we begin by examining the challenges faced by existing Tone Transfer architectures when it comes to effectively supporting aspects of musical expression such as phrasing, articulation, and sonic diversity. We argue that these challenges are primarily linked to the training methods employed and the commonly used synthesis models. Next, we propose Envelope Learning as a method to circumvent these issues. This technique revolves around designing Tone Transfer architectures that focus on matching synthesis parameters instead of audio features. Since our models learn musical events at the level of synthesis control, they can reproduce quick changes in sound, such as the start and end of musical notes, which are essential for musical phrasing.

We train the models to learn different tones by using patches from a well-known FM synthesizer, which provides a diverse range of sounds to work with. Finally, we implement our models on an audio plugin for real-time performances and reflect on its performance and possibilities for improvement. For training and deploying source code, see the online supplement [1]. We expect our models to complement existing Tone Transfer architectures and offer further performance possibilities for live use and sound design.

AM '23, August 30–September 01, 2023, Edinburgh, United Kingdom

ACM ISBN 979-8-4007-0818-3/23/08.
https://doi.org/10.1145/3616195.3616196

[1] https://fcaspe.github.io/fmtransfer

2 BACKGROUND

2.1 Synthesis control with audio signals

Audio-based control in synthesis has a longstanding history, exemplified by pioneering instruments like the Roland guitar [19]. In recent times, a prevalent technique involves using onset detectors and fundamental frequency trackers, enabling control of a synthesizer through MIDI signals [11]. This method facilitates the translation of audio input into synthesized sounds, offering a versatile approach to musical control.

However, generating MIDI triggers through explicit note onset detection introduces a bottleneck in the gestural channel [18] between the instrument and the synthesizer which may hinder an expressive performance. In that regard, timbre characteristics related to musical phrasings, such as variations in dynamics, frequency spectrum, amplitude envelope, and attack transients [27], are compressed into a single scalar velocity value. In MIDI, phrasing is implicitly represented through the timing and velocities of a sequence of note events. Audio-to-MIDI converters typically identify single notes at a time without consideration of longer sequences, so any temporal jitter or inaccuracy in dynamics could result in a disjointed sense of phrasing when the MIDI sequence is replayed by a synthesizer.

In addition to MIDI-based control, alternative strategies have been investigated, involving the extraction of continuous features from the audio signal of a musical instrument to control synthesis processes. In prior work, audio signals from instruments have been utilized as oscillators [29]. This technique employs the audio signal itself as an oscillator for generating synthesized sounds, a special case of an Adaptive Digital Audio Effect [20, 39]. However, this approach restricts the method's versatility as it binds the sonic characteristics of the input directly to the output limiting its range of sonic possibilities.

Continuous control offers the possibility of better supporting musical expression. Interestingly, by closely analyzing how notes from an audio source are intertwined, we may also be able to facilitate longer phrasing arcs on the synthesizer. The problem now resides in navigating the complexity of the mapping design. What are the features we should extract, and how should we associate them to support a variety of synthetic sounds? There are many degrees of freedom, and the strategy becomes much less evident [31].

One possible answer can be found in the work of Levitin et al., where they proposed a valuable framework for analyzing the processes involved in a musical event control [21]. They outline distinct stages of control within a musical event including the beginning, middle, ending, and terminus of the event, and highlight that Digital Musical Instruments (DMIs) often provide greater control over the middle.

In this context, different beginnings and endings can encompass musical articulation and serve as vital contextual links within musical phrases [23]. The audio signal contains this important information within a very short duration and may require direct attention and specific handling to accurately capture and preserve these critical elements. These insights underscore the need for a focused approach to address beginnings and endings explicitly.

2.2 Differentiable Signal Processing

The seminal work of Van Den Oord et. al. [38], spawned a novel approach for data-driven audio generation and control called Neural Audio Synthesis. In this context, Deep Neural Networks (DNN) learn complex synthesizers from audio corpora, that can be used for composition [13], singing voice control [42] timbre transformation [17] and synthesizer parameter estimation [6], to name a few applications.

Engel et. al. [12] proposed a method called Differentiable Signal Processing (DDSP) that combines neural networks and DSP modules, such as synthesizers and audio effects, allowing an error signal to be backpropagated through them. This approach enables joint training of the whole pipeline, effectively biasing the network to learn to control the DSP modules. It allows efficient sound generation with DNN models that can comfortably run in real-time on a CPU [14] and yield impressive results on a variety of differentiable synthesis architectures for musical instrument [5, 16, 34] and singing voice rendering [42, 45].

2.3 Tone Transfer

Tone Transfer [4] is a promising application enabled by DDSP for audio-based control of synthesizers. The supporting architecture, called DDSP Decoder [12], learns to control parameters of a synthesizer, conditioned by a frame-wise fundamental frequency (F_0) and loudness sequences extracted from an input audio signal. These characteristics are instrument-agnostic and relate uniquely to musical form; during inference, the model can support any musical instrument signal that contains a tractable F_0.

A continuously controllable synthesizer such as the DDSP Decoder can potentially deal with the fine-grained characteristics of note beginnings and endings, essential for phrasing. Nevertheless, we note that certain design decisions related to its architecture and training methods may hinder phrasing, articulation, and sound diversity in a performance setting.

One problem is related to the training process, which aims to resynthesize an audio corpus of a particular instrument from a set of F_0 and loudness conditioning sequences, guided by the Multiscale Spectrogram Loss [36, 42]. This involves a trade-off between time and frequency resolution [32], affecting the model's capacity of discerning and synthesizing accurate instrument onsets, which typically happen in the order of tens of milliseconds [41] and are essential to convey distinct articulation and build musical phrases.

Transient rendering is also affected by the synthesizer architectures typically employed in DDSP decoders. In the majority of the cases, a harmonic source such as a harmonic synth [12], a waveshaper [16] or a wavetable [34] is paired with a noise synthesizer in a setting that resembles a Spectral Modelling Synthesizer [33]. This configuration is usually not sufficient for an accurate representation of transients [9, 40].

Regarding sound diversity, we note that the resynthesis objective implicitly ensures a high correlation between the input and output loudness, as indicated in the original paper [12]. Since different musical instruments have different loudness profiles, in many cases performers expect the dynamic characteristics of the generated audio to be different from those of the input. Losing this degree of freedom may make the learned timbre track the dynamics of the

input too closely, producing unnatural sounds and limiting sonic diversity.

Another issue is related to the availability of training data. Single musical instrument datasets are difficult to collect [22], and in many cases, F_0 may not be easy to extract, especially for synthetic sounds. This also limits the amount and type of sounds that can be synthesized with Tone Transfer.

Finally, it is worth noting that sound design practitioners are not familiar with the spectral modeling synthesizers typically employed for Tone Transfer. A well-known architecture with interpretable parameters allows performers to intervene in the synthesis process and manipulate results enhancing the possibilities of pre-trained models [5].

2.4 FM Synthesis

Frequency Modulation (FM) synthesis is a well-known method to generate complex sounds from a compact set of synthesis parameters [7]. One of the best-known implementations is the Yamaha DX7, which utilizes a well-established linear FM synthesis architecture, that has been used in other works for applications such as sound matching and neural audio synthesis [5, 6].

The DX7 generates its distinctive sound using six frequency-modulated sinusoidal oscillators. Programming the synthesizer involves configuring a patch that specifies various parameters for each oscillator. These parameters include the routing, which determines how the oscillators are interconnected (e.g., in a stacked or additive manner), the frequency ratios of the oscillators relative to the played note, as well as the Attack-Decay-Sustain-Release (ADSR) parameters of its Envelope Generators (EGs).

During audio rendering, the oscillator's frequency ratios and routing remain fixed. Instead, the sound dynamics are primarily controlled by the ADSR envelopes. These envelopes modulate the output levels of each oscillator, influencing either their volume or modulation index, depending on their interconnection. Sound design on the DX7 involves configuring the routing, frequency ratios, and ADSR parameters of the EGs.

3 METHOD

Existing Tone Transfer architectures have shown the ability to learn relationships between control inputs and synthesizer features. However, we have observed certain limitations in terms of transient generation and sonic diversity that could restrict the performative possibilities of the models.

We propose an alternative design method for a model that learns relationships from a dataset of synthesis control signals extracted from synthesizer patches and designed following the musical event control model described by Levitin et al. [21]. The model learns to render note beginnings, middles, and endings directly from a continuous control source. We use an FM synthesizer based on the Yamaha DX7 for which there is a wide variety of sounds available on the web [37].

We divide our approach into three stages, shown in Figure 1, namely a dataset generation step that creates event-aligned sequences from synthesizer patches, a training step we call Envelope Learning that learns a mapping function g_ϕ between these sequences, and an inference step where we deploy trained models into a Tone Transfer pipeline and use them to control an FM oscillator block with audio signals.

Our current research shares similarities with our previous work, where we utilize a neural network to control oscillator amplitudes of an FM synthesizer based on a sequence of audio signal features [5]. However, in contrast to our earlier approach, we introduce a new design strategy that (1) avoids the reconstruction objective and MSS loss, allowing decoupled dynamics (2) learns an input-to-output mapping at the level of short frames of signal, allowing for accurate transients, and avoiding the use of differentiable synthesis components, and (3) does not require an audio corpus for training, and instead can learn from a patch collection of the FM synthesizer.

3.1 Dataset Generation

In this step, we create a dataset of M training tuples (a^i, f^i, ol^i), $i = 1, ...M$, with $a = a_1, ...a_K \in \mathbb{R}$ and $f = f_1, ...f_K \in \mathbb{R}$ being sequences of length K modeling amplitude and fundamental frequency of a monophonic audio input respectively. $ol = ol_1, ...ol_K \in \mathbb{R}^6$ represent the linear output level envelopes of the six FM oscillators, that we extract from a synthesizer patch. For training, we use (a, f) as input sequences to our model, and ol as supervision.

In order to generate the input sequences (a^i, f^i) we take into consideration the model proposed by Levitin et al., [21]. To simplify the dataset generation process, we only consider separate notes as musical events. For our Tone Transfer use case, an explicit note beginning is determined by a sudden change in the input amplitude contour a and a valid F_0 detected in f. During the middle, the amplitude and fundamental frequency are sustained over time. Finally, the ending of a note is characterized by a decay trajectory in amplitude, while the F_0 remains valid until the terminus. Considering this, we can model our amplitudes a with a trapezoid generator, that is, a step generator plus a decay ramp. The fundamental frequency contour of a note can be represented with a step generator.

To obtain ol^i, we use a Python implementation of the Yamaha DX7 ADSR Envelope Generators (EGs) adapted from a well-known emulator [15]. These EGs can be programmed with a synthesizer patch p and actuated through MIDI to obtain the amplitude envelope sequences of the six oscillators.

The dataset generation starts with a designer selecting a synthesizer patch p they want to enable for Tone Transfer. We program the ADSR parameters of the EGs with p and generate a set of MIDI notes of random duration and with random velocity and note values.

For each note, we obtain the oscillator envelope sequences ol^i, and create the aligned input sequences (a^i, f^i) following a simple set of rules. When a "NOTE ON" message is received, we generate a step response with an amplitude proportional to the velocity and note value for a and f respectively. After the "NOTE OFF" event is received, a linear decay ramp is rendered in a until the last oscillator envelope in ol reaches zero. During this time f remains valid and then is set to zero.

Next, the input sequences are normalized between $[0, 1]$, establishing a linear range of a and f corresponding to MIDI velocity and MIDI notes values respectively. The oscillator envelope sequences fall in the range of $[0, 2]$; they are also normalized to a range within

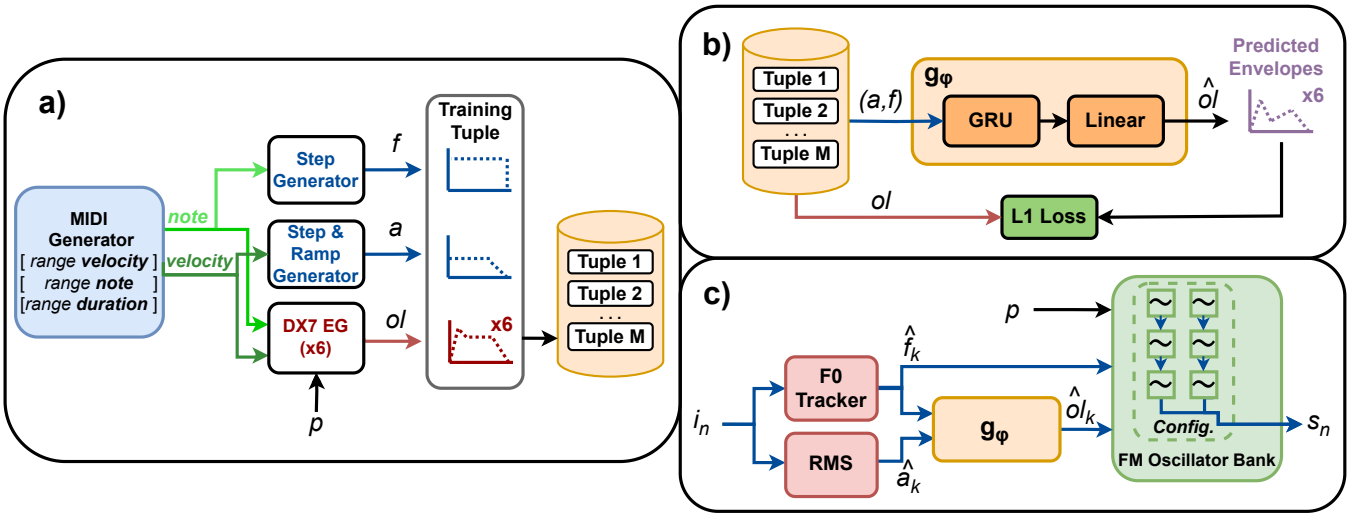

Figure 1: Design steps for our Tone Transfer system. a) we create a synthetic dataset of aligned sequences (a, f, ol). a and f model the frame-wise amplitude and F_0 trajectories of a monophonic audio signal, while ol are the oscillator output levels of an FM synthesizer programmed with a patch p. b) We train a Recurrent Neural Network model g_ϕ to learn the correspondences between the features a, f and the controls ol reflected in the dataset. c) We deploy the RNN into a Tone Transfer pipeline. In this context, g_ϕ processes frame-wise input features from real audio i_n, and controls the envelopes of an FM oscillator bank configured according to p.

$[0, 1]$. Finally, all sequences are padded with zeroes before and after so that each training tuple features the same length.

The end result is a dataset that aligns two characteristics of musical events (a and f) to synthesizer controls (ol) that can render a specific timbre obtained from the patch. Since the input a is proportional to the velocity, note beginnings are characterized by different amplitude discontinuities which in turn, are aligned with the oscillator envelopes ol that render different onsets. Middles are mostly aligned with the decay and sustain parts of ol, and the decaying sections of note endings are synchronized with the release sections of the envelopes. Figure 2 shows a plot of the first six training instances of a dataset generated from the "E. PIANO 1" patch, a well-known DX7 electric piano patch, illustrating the synchrony between inputs and oscillator envelopes.

It is important to recognize that musical instrument notes can often display ambiguous behavior, and it is not always the case that a decrease in amplitude indicates the end of a note or a sustained amplitude indicates the middle. In a causal setting for real-time use, we cannot be sure that a note is ending even if there is a decay amplitude trajectory in the input. Although the input trajectories used in this setting may not fully represent a real-world scenario, they are valuable in demonstrating the proof of concept and analyzing opportunities for improvement.

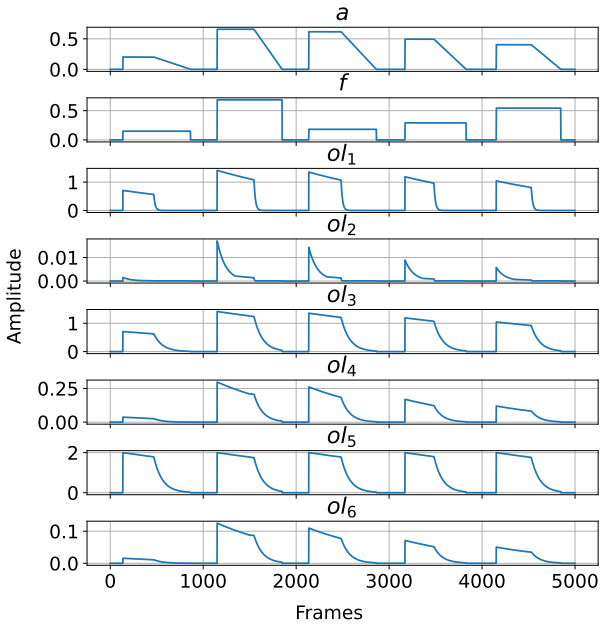

Figure 2: Five training tuples of a dataset extracted from the "E. PIANO 1" patch. Showing its corresponding input sequences a and f, and the synchronized envelopes ol.

3.2 Envelope Learning

To implement our system, we need a neural network model that can learn the temporal relationships between the inputs (a, f) and the oscillator envelopes ol, rendering the attack and decay sections of the control sequences after a discontinuity in the input is detected, and generating note ends accordingly when a decay trajectory is detected in the input.

To this end, and following the design of other Tone Transfer architectures, we employ a model that features a stateful Gated Recurrent Unit (GRU) and a linear layer as output layer. The GRU is a causal model that works frame-by-frame, is conditioned by the a and f sequences, and learns the relationships between current and past inputs, producing a hidden state that is projected with the linear layer into six controls for the oscillators. We denote the neural net as the parameterized function g_ϕ as shown in Eqn. 1, where k denotes the frame index.

$$\hat{ol}_k = g_\phi(a_k, f_k) \tag{1}$$

Since we do not employ audio during our training process, we train the network by conditioning it with a and f, and using the oscillator envelopes ol from the dataset as supervision. We use the L1 Loss between the oscillator envelope predictions and ground truth as the minimization objective: $L = ||ol - \hat{ol}||_1$. We call this process Envelope Learning.

The L1 loss aims to match every single frame that is generated by the network directly with the ground truth. This is unlike the DDSP-based methods that learn to control envelopes *indirectly* by employing a resynthesis objective, a spectrogram audio loss, and noise synthesizers. This results in limited transient resolution, as explained in the previous section. Learning in a direct fashion allows us to explicitly address and reproduce transients during training.

3.3 Inference

Model inference takes place within the Tone Transfer pipeline, which takes an input audio signal i_n, and yields a synthesized output s_n, with n denoting the audio sample index. Similar to other Tone Transfer approaches, we divide this pipeline into three stages:

(1) Feature Extraction, which obtains aligned features from input audio $\hat{a_k}$ and $\hat{f_k}$ related to input amplitude and fundamental frequency f_{0k} respectively, with k denoting frame index. These are extracted from input audio across an analysis window of length W. $A(.)$ may denote a signal amplitude or power estimator algorithm, $F(.)$ an F_0 tracker, and $G(.)$ a normalization function that maps F_0 into the range $[0, 1]$.

$$\begin{aligned} \hat{a_k} &= A(i_{Wk}, ...i_{W(k+1)}) \\ f_{0k} &= F(i_{Wk}, ...i_{W(k+1)}) \\ \hat{f_k} &= G(f_{0k}) \end{aligned} \tag{2}$$

(2) Control Prediction, we use our neural network g_ϕ to infer a set of frame-wise FM synthesis controls, the oscillator output levels $\hat{ol}_k$, from the conditioning signals $\hat{a_k}$ and $\hat{f_k}$.

$$\hat{ol}_k = g_\phi(\hat{a_k}, \hat{f_k}) \tag{3}$$

(3) An FM oscillator bank $S_p(.)$ renders a window of N audio samples from output levels $\hat{ol}_k$, fundamental frequency f_{0k}. We configure the bank with the oscillator routing and frequency ratios of the patch p used to train g_ϕ, although this can be changed during inference.

$$s_{Nk}, ...s_{N(k+1)} = S_p(\hat{ol}_k, f_{0k}) \tag{4}$$

4 IMPLEMENTATION

4.1 Training

We select a set of common DX7 patches and create a training dataset for each one of them. Next, we train one neural net model per patch following our Envelope Learning method.

For each patch, we generate 1000 random MIDI notes with velocities between 1 and 127, and note values between 0 and 127. We set a random duration for each note between 600 and 732 frames. Next, we generate the aligned input and oscillator envelope sequences a, f, and ol, as described in Section 3.1. Finally, we pad them with zeroes to reach a final size of 1000 frames per instance, so that the active notes occupy about two-thirds of the total length. We split the dataset with a ratio of $[0.80, 0.1, 0.1]$ for training, validation, and testing respectively.

Our neural net features a GRU with a hidden size of 128. We empirically choose this value as we note that the training loss does not improve with bigger models, and to keep the computing requirements low. We train one model per dataset, for a total of 120000 steps, using the Adam optimizer with a learning rate of 1e-3, a learning rate decay of 0.98 for every 10000 steps, and a batch size of 32 instances. We use Pytorch as a training framework. The process takes about four hours per model using a single NVIDIA GeForce RTX 2080 Ti GPU.

To assess the effectivity of the training process, we compare on the test set the absolute distance between the ground truth oscillator envelopes and the predictions $||ol - \hat{ol}||_1$ for each trained model.

Furthermore, we set out to assess the capabilities of each of the trained networks for synthesizing audio with the learned timbre. Firstly, we render audio using both the ground truth ol and predicted envelopes ol, using an FM oscillator block configured with the oscillator routing and ratios extracted from the patches used to train the models. We employ the fundamental frequency f_0 extracted from the normalized MIDI note values present in the sequences f of the test set.

Next, we compute the signal-to-noise ratio (SNR) as a power quotient of our reference and an error computed from the sample-by-sample difference between both signals. We compute the SNR in decibels (dB), as shown in Eqn. 5.

$$SNR = 10 \cdot log_{10}\left(\frac{||S_p(ol, f_0)^2||_1}{||(S_p(ol, f_0) - S_p(\hat{ol}, f_0))^2||_1}\right) \tag{5}$$

We use this metric to assess the reconstruction quality of note beginnings and endings. To account for note beginnings, we aggregate the first 100 milliseconds of each note in the test set for both rendered audios and then compute the SNR on these signals obtaining SNR_{onset}. We use 100 ms to account for the different onset times that the models present. Furthermore, we identify the ending sections of each note by looking at the decaying ramp in a, which is aligned with ol in our dataset. We aggregate the audio samples of each note and compute SNR_{end}. We aggregate the rest of the audio section of each note, between the note onset and the start of the decaying ramp, and compute SNR_{mid} at the note middle.

Table 1 shows the results for the metrics. The low L_1 loss indicates that our model is able to minimize the training objective and predict the oscillator envelope sequences from the conditioning

signals. This translates into an adequate reproduction of note beginnings, middles and endings; the SNR metrics show that even in the worst case, the models can render the note sections with not more than about 1% of power error.

Although these results do not represent our models' performance capabilities when deployed in a Tone Transfer pipeline, they show that our training objective allows the networks to learn the envelope contours of the oscillators for different timbres, and can accurately render the beginning, middles, and endings of notes when conditioned with the continuous input sequences a and f.

Model	Envelope L_1	SNR_{onset}	SNR_{mid}	SNR_{end}
Brass	7.77e-4	36.4	29.3	33.7
Strings	1.05e-3	29.4	34.4	34.9
E. Piano	3.06e-3	27.5	30.2	27.9
Marimba	8.07e-4	36.5	39.7	35.8
Voice	7.37e-4	19.5	33.3	30.0
Sitar	2.23e-3	22.2	25.3	27.2
PolySynth	1.36e-3	30.6	37.3	30.7

Table 1: Envelope absolute error and audio SNR at beginnings and endings of the test set notes for all trained models.

4.2 Deployment

We implement the Tone Transfer pipeline on a real-time audio plugin using JUCE and Libtorch, Pytorch's C++ API. Our prototype can load new neural net models and FM configurations, supporting all the learned timbres. It runs in real-time and performs inference and synthesis at a frame rate of 690 Hz, to render audio at a sample rate of 44.1kHz, similar to our Yamaha DX7 reference implementation [15].

Within the pipeline, we extract frame-wise fundamental frequency f_{0k} using the YIN algorithm [10], using an analysis window of 1024 samples, which yields a minimum detectable frequency of about 90 Hz. We compute the conditioning signal f_k by converting the fundamental frequency values from Hz to MIDI note value and then applying normalization between $[0, 1]$, as shown in Eqn. 6. Furthermore, when a valid fundamental frequency is not detected, the extractor returns zero.

$$f_k = 12\frac{log_2(f_{0k}/220) + 57.01}{127} \tag{6}$$

Next, we supply the continuous amplitude input for our system a from a decibel-scale RMS detector. We employ a compute block over a sliding window W, clamping the minimum value to -70dB, and normalizing between 0 to 1.

$$a_k = 1 + \frac{1}{70} \cdot max(-70, log_{10}(\sum_W \frac{i_n^2}{W})) \tag{7}$$

Since our datasets (and therefore, our trained models) present a linear amplitude range in a, our system tries to match normalized RMS in decibels to envelope variations associated with MIDI velocity.

Next, our model predicts the current envelope values for the FM synth, which are interpolated from frame to sample rate and used for the synthesis process. The fundamental frequency f_{0k} is also linearly interpolated and used to drive the oscillators at the synthesis step.

Furthermore, we reset the model's hidden state to all zeroes when both conditioning sequences are zero. This ensures that the model starts from a known state to process a new incoming note. The plugin runs on a MacBook Pro 2021 with a USB audio interface running at 44.1 kHz and a hop size of 64 samples, yielding a pipeline delay of 3 ms including buffering.

5 DISCUSSION

Our model offers the capability to generate a wide range of timbres on an FM synthesizer by learning the dynamic trajectories of oscillator envelopes reflected in the dataset. Our approach effectively replaces the traditional envelope generator of the DX7 with a recurrent neural network (RNN) that provides continuous controllability instead of MIDI.

In this context, the dataset generation approach serves as the bridge between explicit note beginnings and endings, which are event-based, and the continuous control framework. When trained with our Envelope Learning method, the network is able to learn and reproduce the rapid beginnings and endings of notes, even without explicit information about note boundaries.

Previous Tone Transfer architectures learn to control synthesizers *indirectly* by minimizing an audio loss of a resynthesis task, using spectrogram losses that act upon long windows of audio. These effectively look at the middle of musical events and present a limited temporal resolution for beginnings and endings. In contrast, our approach overcomes this limitation by learning a direct correspondence between inputs and synthesis parameters at a control level. This allows for precise rendering of the transient characteristics of the learned timbre, provided we have a representation of that timbre available in the form of a synthesizer patch. We argue these are the first steps for achieving expressive and nuanced synthesizer articulation with Tone Transfer algorithms.

We suggest that these results are encouraging to explore the Envelope Learning technique building further input-output associations. One possibility would be to align additional input features such as spectral features to other sound characteristics like attack and decay rates. Another would be to introduce multiple notes per training instance to explicitly model phrasing in context, modeling note events of specific musical instruments in the dataset for a more nuanced control. Other alternatives include exploring further conditioning choices to assess responsivity in terms of dynamics, modifying the trapezoidal amplitude note model in a to better account for particular instruments and input detectors, or training using patch data from other synthesizer architectures.

5.1 Reflections on performances

As a proof of concept, we record two musicians using our audio plugin in real-time, playing guitar and sax [2]. We select these two source instruments since they provide very different volume dynamics and articulations to drive our plugin. We record three models, trained with electric piano, strings, and brass patches respectively.

We informally observe that our Tone Transfer approach can effectively render timbre from the learned patches, including note

[2]The video is available on the supplementary website

beginnings. This is reflected particularly well in the example of the guitar controlling the electric piano, which shows a bright attack on the beginning of those notes that are not *legato*. Next, in the guitar example that plays a string tone, the synthesizer features a slow attack, even though the guitar is plucked, showing that our model does not project input loudness to the output, as DDSP does.

Note endings are much more difficult to assess since their generation depends on a decaying amplitude envelope presented by the audio input. On the guitar, the decay envelope may be too fast to render the learned note ending before the fundamental frequency cannot be tracked anymore.

For the case of the saxophone as a control source, note beginnings and endings are not that clear, but this is to be expected as it presents a much different amplitude contour, including amplitude modulations that were not accounted for during dataset generation. These may force the model to re-render note characteristics of beginnings or endings, which can be observed when the saxophone controls the electric piano.

On the other hand, we note a lack of dynamic range in the synthesis output. We argue that this is due to the fact that the patches were originally designed to be played with a keyboard with MIDI notes and velocity controls. For the electric piano patch, for instance, we observe that a low-velocity value still produces a signal with high amplitude but less brightness. Redesigning the patches to obtain higher variations in output amplitude and retraining the models may improve the results.

6 CONCLUSIONS

Transforming the audio of an instrument to a synthetic sound is a challenging task, as it involves a one-to-many relationship. Each instrument has its unique timbral palette, dynamic contour, and articulation possibilities, which can vary significantly even among instruments of the same type. On the other hand, the sound produced by a synthesizer can be highly versatile; and only a subset of the source instrument's characteristics may be desired in the output.

We can argue that there is no definitive "gold standard" that can provide a baseline mapping between an instrument's audio and a synthetic sound: tradeoffs are necessary to find viable solutions.

In this work, we first analyzed current Tone Transfer architectures and identified a tradeoff in their rendering capabilities: these models learn new timbres from audio corpora and can project the input loudness to the output, at the expense of a good resolution of note beginnings and endings which are essential for musical articulation and phrasing.

In light of the analyzed shortcomings, we presented Envelope Learning, a design method where a model learns a set of input-to-synthesis parameters correspondences and accurately replicates note beginnings and endings. The tradeoff, in this case, is on the note middles: we use a simplified musical note model for our dataset generation that does not consider variations in amplitude or pitch during an event. This works well during testing in the training environment but may result in unexpected transitions and reduced dynamic range when used in a Tone Transfer setting, especially with sustaining instruments. We leave for future work an assessment of the performance possibilities of our algorithm and an exploration of techniques to overcome current limitations.

Finally, we implemented a Tone Transfer pipeline in an audio plugin for real-time performance, taking a step towards improving sound diversity and phrasing capabilities for audio-based control of synthesizers. Our system bridges the sonic diversity gap of previous approaches, learning new sounds from a vast number of DX7 patches for which their timbre can now be continuously controlled with musical instruments. We hope that our work motivates further research in model design with the goal of improving phrasing and articulation in real-time neural synthesizers controlled by musical instruments.

ACKNOWLEDGMENTS

This work is supported by the UKRI through the Centre for Doctoral Training in Artificial Intelligence and Music (EP/S022694/1) and a UKRI Frontier Research grant (EP/X023478/1).

REFERENCES

[1] Frédéric Bevilacqua, Florence Baschet, and Serge Lemouton. 2012. The Augmented String Quartet: Experiments and Gesture Following. *Journal of New Music Research* 41, 1 (March 2012), 103–119. https://doi.org/10.1080/09298215.2011.647823

[2] Roberto Bresin and Giovanni Umberto Battel. 2000. Articulation Strategies in Expressive Piano Performance Analysis of Legato, Staccato, and Repeated Notes in Performances of the Andante Movement of Mozart's Sonata in G Major (K 545). *Journal of New Music Research* 29, 3 (Sept. 2000), 211–224. https://doi.org/10.1076/jnmr.29.3.211.3092

[3] ByteDance. 2023. Mawf. https://mawf.io/.

[4] Michelle Carney, Chong Li, Edwin Toh, Nida Zada, Ping Yu, and Jesse Engel. 2021. Tone Transfer: In-Browser Interactive Neural Audio Synthesis. In *Joint Proceedings of the ACM IUI 2021 Workshops*. ACM, College Station, United States,, 6 pages.

[5] Franco Caspe, Andrew McPherson, and Mark Sandler. 2022. DDX7: Differentiable FM Synthesis of Musical Instrument Sounds. In *Proceedings of the 23nd ISMIR Conference*. International Society of Music Information Retrieval (ISMIR), Bengaluru, India, 9 pages.

[6] Zui Chen, Yansen Jing, Shengcheng Yuan, Yifei Xu, Jian Wu, and Hang Zhao. 2022. Sound2Synth: Interpreting Sound via FM Synthesizer Parameters Estimation. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence, Vienna, Austria, 8 pages.

[7] John M. Chowning. 1973. The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. *JAES* 21, 7 (Sept. 1973), 526–534.

[8] Palle Dahlstedt. 2017. Physical Interactions with Digital Strings - A Hybrid Approach to a Digital Keyboard Instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression,*. NIME, Aalborg University, Copenhagen, 115–120.

[9] L. Daudet. 2001. Transients Modelling by Pruned Wavelet Trees. In *Proceedings of the 2001 International Computer Music Conference*. Michigan Publishing, Havana, Cuba, 4 pages.

[10] Alain de Cheveigné and Hideki Kawahara. 2002. YIN, a Fundamental Frequency Estimator for Speech and Music. *The Journal of the Acoustical Society of America* 111, 4 (April 2002), 1917–1930. https://doi.org/10.1121/1.1458024

[11] Olivier Derrien. 2014. A Very Low Latency Pitch Tracker for Audio to Midi Conversion. In *17th International Conference on Digital Audio Effects (DAFx-14)*. DAFX, Erlangen, Germany, 6 pages.

[12] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. 2020. DDSP: Differentiable Digital Signal Processing. In *8th International Conference on Learning Representations*. ICLR, Addis Ababa, Ethiopia, 19 pages.

[13] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. 2017. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, Sydney, NSW, Australia, 1068–1077.

[14] Francesco Ganis, Erik Frej Knudesn, Søren V. K. Lyster, Robin Otterbein, David Südholt, and Cumhur Erkut. 2021. Real-Time Timbre Transfer and Sound Synthesis Using DDSP. In *Proceedings of the 18th Sound and Music Computing Conference*, Vol. abs/2103.07220. Sound and Music Computing, Virtual, 11.

[15] Pascal Gauthier. 2023. Dexed. https://asb2m10.github.io/dexed/.

[16] Ben Hayes, Charalampos Saitis, and György Fazekas. 2021. Neural Waveshaping Synthesis. In *Proceedings of the 22nd ISMIR Conference*. International Society of Music Information Retrieval (ISMIR), Online, 8 pages.

[17] Sicong Huang, Qiyang Li, Cem Anil, Xuchan Bao, Sageev Oore, and Roger B. Grosse. 2019. TimbreTron: A WaveNet(CycleGAN(CQT(Audio))) Pipeline for Musical Timbre Transfer. In *International Conference on Learning Representations*. ICLR, New Orlearns, USA, 17 pages.

[18] Robert Jack, Tony Stockman, and Andrew Mcpherson. 2017. Rich Gesture, Reduced Control: The Influence of Constrained Mappings on Performance Technique. In *Proceedings of the 4th International Conference on Movement Computing*. ACM, London, United Kingdom, 1–8. https://doi.org/10.1145/3077981.3078039

[19] Otso Lähdeoja. 2008. An Approach to Instrument Augmentation : The Electric Guitar. In *Proceedings of the 2008 Conference on New Interfaces for Musical Expression (NIME08)*. NIME, Genoa, Italy, 4 pages.

[20] Victor Lazzarini, Joseph Timoney, and Thomas Lysaght. 2007. Adaptive FM Synthesis. In *DAFX-07 the 10th Int. Conference on Digital Audio Effects*. DAFX, Bordeaux, France, 6 pages.

[21] Daniel J. Levitin, Stephen McAdams, and Robert L. Adams. 2002. Control Parameters for Musical Instruments: A Foundation for New Mappings of Gesture to Sound. *Org. Sound* 7, 2 (Aug. 2002), 171–189. https://doi.org/10.1017/S135577180200208X

[22] Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. 2019. URMP: Creating A Multi-track Classical Musical Performance Dataset for Multimodal Music Analysis: Challenges, Insights, and Applications. *IEEE Trans. Multimedia* 21, 2 (Feb. 2019), 522–535. https://doi.org/10.1109/TMM.2018.2856090

[23] Eric Lindemann. 2007. Music Synthesis with Reconstructive Phrase Modeling. *IEEE Signal Processing Magazine* 24, 2 (March 2007), 80–91. https://doi.org/10.1109/MSP.2007.323267

[24] Google Magenta. 2023. DDSP-VST. https://magenta.tensorflow.org/ddsp-vst.

[25] Andrew McPherson. 2012. TouchKeys: Capacitive Multi-touch Sensing on a Physical Keyboard. In *NIME 2012*. NIME, Ann Arbor - Michigan, 4. https://doi.org/10.1007/978-3-319-47214-0_27

[26] Giulio Moro and Andrew P. McPherson. 2020. Performer Experience on a Continuous Keyboard Instrument. *Computer Music Journal* 44, 2-3 (July 2020), 69–91. https://doi.org/10.1162/comj_a_00565

[27] Kirk N. Olsen, Roger T. Dean, and Yvonne Leung. 2016. What Constitutes a Phrase in Sound-Based Music? A Mixed-Methods Investigation of Perception and Acoustics. *PLoS ONE* 11, 12 (Dec. 2016), e0167643. https://doi.org/10.1371/journal.pone.0167643

[28] Trevor Pinch and Frank Trocco. 1998. The Social Construction of the Early Electronic Music Synthesizer. *Icon* 4 (1998), 9–31. jstor:23785956

[29] Cornelius Pöpel and Roger Dannenberg. 2005. Audio Signal Driven Sound Synthesis. In *Proceedings of the International Computer Music Conference*. Michigan Publishing, Barcelona, Spain, 5 pages.

[30] Qosmo. 2023. Neutone by Qosmo. https://neutone.space/.

[31] Nicolas Rasamimanana. 2012. Towards a Conceptual Framework for Exploring and Modelling Expressive Musical Gestures. *Journal of New Music Research* 41, 1 (March 2012), 3–12. https://doi.org/10.1080/09298215.2011.642391

[32] Benjamin Ricaud, Guillaume Stempfel, Bruno Torrésani, Christoph Wiesmeyr, Hélène Lachambre, and Darian Onchis. 2014. An Optimally Concentrated Gabor Transform for Localized Time-Frequency Components. *Adv Comput Math* 40, 3 (June 2014), 683–702. https://doi.org/10.1007/s10444-013-9337-9

[33] Xavier Serra and Julius Smith. 1990. Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition. *Computer Music Journal* 14, 4 (1990), 12–24. https://doi.org/10.2307/3680788 jstor:3680788

[34] Siyuan Shan, Lamtharn Hantrakul, Jitong Chen, Matt Avent, and David Trevelyan. 2022. Differentiable Wavetable Synthesis. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Singapore, 4598–4602. https://doi.org/10.1109/ICASSP43922.2022.9746940

[35] Koray Tahiroğlu, Miranda Kastemaa, and Oskar Koli. 2021. AI-terity 2.0: An Autonomous NIME Featuring GANSpaceSynth Deep Learning Model. In *NIME 2021*. PubPub, Shanghai, China, 20. https://doi.org/10.21428/92fbeb44.3d0e9e12

[36] Joseph Turian and Max Henry. 2020. I'm Sorry for Your Loss: Spectrally-Based Audio Distances Are Bad at Pitch. In *Accepted Papers at ICBINB 2020*. Curran Associates, Virtual, 16 pages.

[37] Joseph Turian, Jordie Shier, George Tzanetakis, Kirk McNally, and Max Henry. 2021. One Billion Audio Sounds from GPU-Enabled Modular Synthesis. In *2021 24th International Conference on Digital Audio Effects (DAFx)*. IEEE, Vienna, Austria, 222–229. https://doi.org/10.23919/DAFx51585.2021.9768246

[38] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. In *Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*. International Speech Communication Association (ISCA), Sunnyvale, USA, 15 pages.

[39] V. Verfaille, U. Zolzer, and D. Arfib. 2006. Adaptive Digital Audio Effects (a-DAFx): A New Class of Sound Transformations. *IEEE Transactions on Audio, Speech, and Language Processing* 14, 5 (Sept. 2006), 1817–1831. https://doi.org/10.1109/TSA.2005.858531

[40] Tony S. Verma and Teresa H. Y. Meng. 2000. Extending Spectral Modeling Synthesis with Transient Modeling Synthesis. *Computer Music Journal* 24, 2 (June 2000), 47–59. https://doi.org/10.1162/014892600559317

[41] Joos Vos and Rudolf Rasch. 1981. The Perceptual Onset of Musical Tones. *Perception & Psychophysics* 29, 4 (July 1981), 323–335. https://doi.org/10.3758/BF03207341

[42] Xin Wang and Junichi Yamagishi. 2019. Neural Harmonic-plus-Noise Waveform Model with Trainable Maximum Voice Frequency for Text-to-Speech Synthesis. In *Proceedings of the 10th ISCA Speech Synthesis Workshop*. International Speech Communication Association, Vienna, Austria, 6 pages. https://doi.org/10.21437/SSW.2019-1

[43] D. Wessel, D. Bristow, and Z. Settel. 1987. Control of Phrasing and Articulation in Synthesis. In *Proceedings of the 1987 International Computer Music Conference*. Michigan Publishing, Champaign/Urbana, Illinois, USA, 9 pages.

[44] Yusong Wu, Ethan Manilow, Yi Deng, Rigel Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Cheng-Zhi Anna Huang, and Jesse Engel. 2022. MIDI-DDSP: Detailed Control of Musical Performance via Hierarchical Modeling. In *International Conference on Learning Representations*. ICLR, Virtual, 27 pages.

[45] Da-Yi Wu1, Wen-Yi Hsiao, Fu-Rong Yang, Oscar Friedman, Warren Jackson, Scott Bruzenak, Yi-Wen Liu, and Yi-Hsuan Yang. 2022. DDSP-based Singing Vocoders: A New Subtractive-based Synthesizer and A Comprehensive Evaluation. In *Proceedings of the 23nd ISMIR Conference*. International Society of Music Information Retrieval (ISMIR), Bengaluru, India, 8 pages.