SYNTHESIZER SOUND MATCHING USING AUDIO SPECTROGRAM TRANSFORMERS

Fred Bruford[1], *Frederik Blang*[2], *and Shahan Nercessian*[3]

Native Instruments
[1]London, United Kingdom [2]Berlin, Germany [3]Boston, MA, USA
{fred.bruford|frederik.blang|shahan.nercessian}@native-instruments.com

ABSTRACT

Systems for synthesizer sound matching, which automatically set the parameters of a synthesizer to emulate an input sound, have the potential to make the process of synthesizer programming faster and easier for novice and experienced musicians alike, whilst also affording new means of interaction with synthesizers. Considering the enormous variety of synthesizers in the marketplace, and the complexity of many of them, general-purpose sound matching systems that function with minimal knowledge or prior assumptions about the underlying synthesis architecture are particularly desirable. With this in mind, we introduce a synthesizer sound matching model based on the Audio Spectrogram Transformer. We demonstrate the viability of this model by training on a large synthetic dataset of randomly generated samples from the popular *Massive* synthesizer. We show that this model can reconstruct parameters of samples generated from a set of 16 parameters, highlighting its improved fidelity relative to multi-layer perceptron and convolutional neural network baselines. We also provide audio examples demonstrating the out-of-domain model performance in emulating vocal imitations, and sounds from other synthesizers and musical instruments.

1. INTRODUCTION AND RELATED WORK

Systems for controlling synthesizers based on an input sound, often referred to as *synthesizer sound matching*, *parameter estimation* or *automatic synthesizer programming*, have attracted research interest since at least the 1990s [1, 2] due to their practical music production applications. Tools that manipulate the parameters of a synthesizer automatically to recreate an input sound could make programming synthesizers more accessible for musicians with a limited understanding of sound synthesis, but also speed up synthesizer programming for experienced musicians alike. Beyond its use as a workflow improvement, synthesizer sound matching could open up new creative possibilities, enabling the use of sound as a control interface to synthesizers, and aiding the creation of new and unique synthesizer patches beyond factory preset libraries. As some rudimentary examples, synthesizer sound matching could open up the possibility of controlling synthesizers through query-by-vocalization, or enable users to recreate sounds from sampled tracks using their own synthesizer.

Systems for synthesizer sound matching fall on a scale according to the level of knowledge they assume of the underlying synthesis architecture. Many contemporary solutions [3, 4, 5] require a fully differentiable implementation of the synthesizer to be a part of the model, assuming that leveraging knowledge of the sound synthesis process should make synthesizer sound matching easier. However, such architectures are inflexible in that they will only work for the synthesizers that they are explicitly designed for, and new architectures will have to be developed for each individual synthesizer. This also becomes intractable as the complexity of the synthesizers being modelled increases; it may be possible to implement simpler synthesizers with differentiable operations, but to do so for the highly complex synthesizers that find widespread use in contemporary music production practice, utilizing banks of wavetables, effects, and complex modulation and routing options, could be prohibitively labor-intensive.

As a result, many approaches to synthesizer sound matching attempt to bypass the use of differentiable processors altogether to create more scalable solutions for the problem. *InverSynth* [6] considers convolutional neural network (CNN) architectures inferring synthesizer configurations from either input spectrograms or even raw audio, whereby models optimize an objective function defined over inferred and ground truth synthesizer parameters. Acknowledging that parameter losses do not directly correlate to perception, *FlowSynth* leverages a variational auto-encoder with normalizing flows in order to jointly organize and map auditory and parameter spaces [7]. Extensions have also considered the joint modeling of several virtual instruments concurrently, encouraging formulation of the method as a multi-task learning problem to further promote system generalization [8].

In recent years, Transformers [9] have been finding increased usage in classification and regression tasks, beyond their popular use as generative models. This has led to their increased uptake within the domain of music information retrieval (MIR) specifically; recently architectures leveraging transformers have been found to outperform more traditional convolutional or recurrent architectures in archetypal MIR tasks such as music tagging [10, 11] and piano music transcription [12]. However, this comes at the cost of increased computational complexity and data requirements, something that can be a problem for music-focused tasks where labeled data can be scarce.

In this paper, we propose a synthesizer sound matching method using an Audio Spectrogram Transformer (AST) [10] backbone. To the best of our understanding, this is the first work to consider the use of an AST for the synthesizer sound matching problem. In doing so, we advance work towards a general-purpose synthesizer sound matching architecture that requires minimal assumptions about the underlying synthesis architecture. ASTs offer a natural solution to this problem due to their strong performance across supervised learning tasks in the music domain. Furthermore, through generation of randomly sampled datasets of paired parameter values and respective audio samples rendered through a synthesizer, we can leverage arbitrarily large, fully-labeled datasets that afford the successful training of transformers, a rarity in the con-

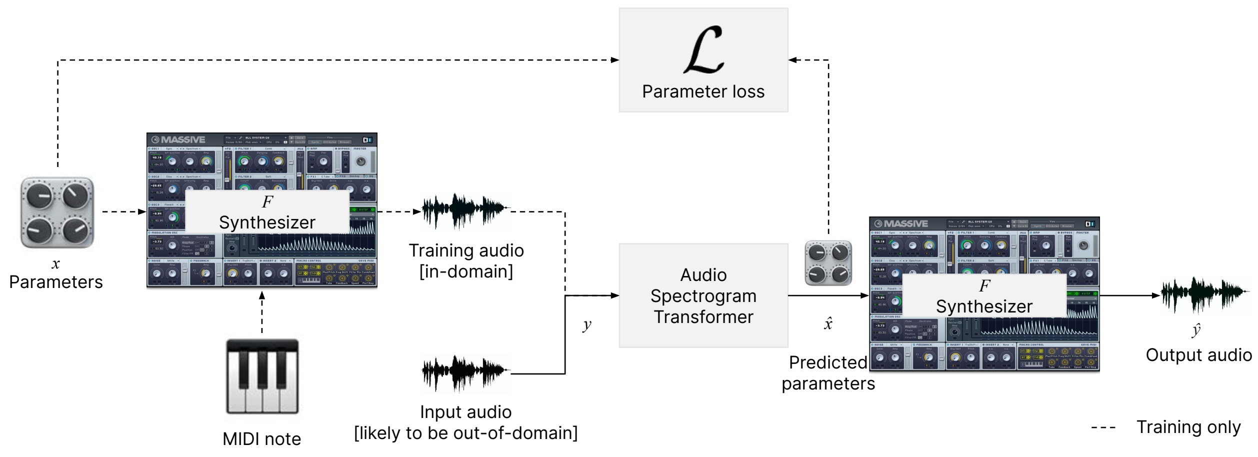

Figure 1: Proposed synthesizer sound matching system block diagram.

text of MIR research. For demonstrative purposes, we evaluate our proposed model on *Massive* by Native Instruments, a popular and widely used synthesizer, known for its expressivity and complexity. We carry out an automated evaluation of our synthesizer sound matching model for parameter prediction quality and audio reconstruction quality, comparing our approach against both multi-layer perceptron (MLP) and CNN baselines. Lastly, we provide examples of sound matching model performance for both in- and out-of-domain audio inputs, available at `https://synth-ast-dafx.netlify.app`.

2. PROPOSED METHOD

Figure 1 outlines our proposed approach. We consider the problem of synthesizer sound matching as one of *parameter inference*, whereby a predictive model (such as an AST), taking an audio sample y as an input, attempts to predict the underlying synthesis parameters x that were used to render that sample. These predicted parameters $\hat{x}$ can then be used to render a new sound $\hat{y} = F(\hat{x})$ that matches the input audio sample by means of a given synthesizer F. This inference process is shown on the right side of Figure 1. To achieve this, a typical approach is to train deep learning models on paired samples of parameter sets x and rendered one-shot samples $y = F(x)$. The training process is shown in the left side of Figure 1. While such a model will be able to recreate synthesizer sounds from the synthesizer being modeled by design, we rely on these models having strong *out-of-domain* performance to be able to generate parameters approximating an arbitrary audio example y originating from any possible sound source. This is to say that models should be robust to the fact that input audio at inference time is unlikely to be constrained to the subspace of sounds created via the signal model $y = F(x)$.

2.1. The Audio Spectrogram Transformer

Our proposed model is derived from the Audio Spectrogram Transformer (AST) [10], itself based primarily on the Vision Transformer (ViT) model [13]. AST is an architecture for audio classification based fundamentally on a vanilla Transformer encoder, combined with a patch-based Mel spectrogram pre-processor.

Transformers typically take as input a 1D vector of token embeddings. To convert a single 2D Mel spectrogram to a 1D sequence of token embeddings, AST splits the spectrogram into a series of *patches* across both axes of the spectrogram in a manner similar to that used in ViT for image data. We use 16×16 patches with a stride of 6. Each patch is flattened to pass into a linear embedding layer of size 768. Due to their use of residual connections between layers, transformers require that all its internal layers are of the same size, so the internal dimensionality of AST is also 768.

As in ViT, AST uses a learnt positional embedding which is summed with the patch embedding to retain the 2D position of each patch, before passing through to an encoder-only Transformer. It uses 12 encoder layers, with 12 attention heads each, and a hidden size of 768, copying the hyperparameters used in the smallest ViT model, ViT-Base [13].

2.2. AST for Synthesizer Sound Matching

We make a number of modifications to AST to make it suitable for the problem of synthesizer sound matching. Firstly, to enable it to work for our regression formulation rather than classification, the final output layer is modified to use an mean-squared error (MSE) loss between output parameter values and target parameter values, rather than the binary cross-entropy loss used in AST. As an additional improvement, we insert a small 3-layer MLP of width 768 between the output of the Transformer encoder and the output layer. We found this to increase the performance of the model, with a minimal effect on training time.

Secondly, AST applies an adaptation to the learnt positional encoding that allow it to model sequences of variable length by interpolating between embeddings. Allowing for variable length inputs is less important for prototyping the synthesizer sound matching, as we assume all inputs sounds are one-shot samples, meaning we can neglect this adaptation and treat the input spectrograms as fixed-size, the same as images are treated in ViT.

We also reduce the size of our Mel spectrogram to 64 bins rather than 128. We found this to have minimal impact on downstream performance, while halving the number of patches required to capture a given spectrogram, thus significantly speeding up the training time. Finally, both ViT and AST pre-train on ImageNet [14] before fine-tuning on downstream classification tasks, but due to our ability to create datasets of arbitrary size (as discussed in Section 3.1) we do not carry this out.

Table 1: *Massive parameters used for 16 parameter dataset*

Parameter name	Description
osc1_position	Wavetable start position for oscillator 1
osc1_amp	Output gain of oscillator 1
osc2_pitch	Detune pitch of oscillator 2
osc2_position	Wavetable start position for oscillator 2
osc2_amp	Output gain of oscillator 2
noise_amp	Noise generator output gain
filter1_cut_prm_1	Lowpass filter cutoff
filter1_res_prm_3	Lowpass filter resonance
filter2_cut_prm_1	Highpass filter cutoff
filter2_res_prm_3	Highpass filter resonance
envelope4_att_tme	Output amplitude attack
envelope4_dec_tme	Output amplitude decay
envelope1_att_tme	Lowpass filter cutoff attack
envelope1_att_lev	Lowpass filter cutoff envelope level
envelope1_dec_tme	Lowpass filter cutoff decay
master_fx2_dry_wet	Reverb dry/wet

3. EXPERIMENTAL RESULTS

3.1. Dataset

Our dataset for the synthesizer sound matching task consists of paired examples of parameter settings and the respective rendered one-shot samples for our chosen synthesizer, *Massive*. Rather than deriving samples from factory presets (as in [7] or [8]), we create a synthetic dataset by randomly sampling parameter values and generating corresponding one-shots, an approach also used in [6]. This allows us to sample the parameter space of *Massive* at arbitrary levels of complexity by varying only desired parameters, and generate datasets of an arbitrary size, allowing for the scale required for training the Transformer architecture.

We automate the generation of this dataset using the *Pedalboard* Python library [15], feeding *Massive* with random parameter values and rendering the output audio for a note of pitch C4 and maximum velocity. Each generated one-shot sample is 4 seconds long, consisting of a 1-second MIDI note on event at the beginning of the sample. Any samples generated with a loudness below a threshold of -60dB were removed to ensure there were no silent or near-silent samples. One disadvantage of using random datasets generated in this way is that the resulting samples may be less practical or realistic than typical synthesizer sounds that a user may want to create. To alleviate this, rather than sampling parameter values from a uniform distribution, we sample them from within the distribution of parameters extracted across all the *Massive* factory presets, therefore sampling more of the ‘usable’ subspace of parameter values that is found in practice.

We created a dataset of 1 million paired samples using a set of 16 parameters. The parameters were selected by hand to lead to sonically diverse samples from a scaled-back parameter set in order to provide an easier proof-of-concept. Random samples are created based on a ‘base’ preset using two oscillators (one with fixed pitch), two filters, two envelopes and a reverb. We select only continuous parameters at this stage to simplify the required architecture, avoiding any categorical parameters or binary switches. The parameters used are shown in Table 1.

We extract and store 64-bin Mel spectrograms from each rendered sample, resulting in a total dataset size of 22GB. Running multiple instances of *Massive* in parallel via *Pedalboard*, we are able to generate this dataset in approximately 24 hours on a dedicated *Mac mini* (2018).

3.2. Evaluation Method

As in [7], we evaluate our models on the test set for both parameter reconstruction and audio reconstruction accuracy using two respective measures: mean-squared error (MSE) between predicted and input parameters, and *Spectral Convergence* (SC) [16] between input audio and output audio as rendered through *Massive* from the input and predicted parameter sets. The latter allows us to directly measure the audio quality of the synthesizer sound matching, which may not be reflected by the parameter reconstruction score alone. In order to further underscore the effectiveness of our methods, we compare our AST model against two established baselines. All SC and MSE scores are aggregated over 10,000 examples taken from the test set.

Aside from quantitative evaluation metrics, to subjectively evaluate both in-domain and out-of-domain sound matching model performance we provide audio examples of reconstructed sounds at our accompanying website. In-domain samples are *Massive* presets drawn from the test set, while out-of-domain samples include vocal synthesizer imitations, sounds from other synthesizers, and musical instrument one-shots. As the AST outputs a set of parameters which can be rendered at any MIDI pitch, reconstructed sounds are rendered at the pitch deduced from input audio.

3.2.1. Baselines

Our two baseline models are implemented based on those used in [7], and are each trained on the same dataset for 50 epochs.

MLP 5-layer MLP with width of 2048, including batch normalization and a dropout of $p = 0.3$, with ReLU activations.

CNN 5-layer, 128-channel 2D CNN with kernel size 7, stride 2, batch normalization and dropout of $p = 0.3$. This is followed by a 5-layer MLP using the same hyperparameters as the standalone MLP.

Our 1 million sample dataset was split into training, validation and test sets in a 80/10/10 split. All models were trained for 50 epochs using the Adam optimizer, though the CNN and MLP training converged after roughly 5 epochs. The learning rate of the CNN and MLP training was $3e-4$ and the AST $5e-5$. Evaluation was carried out on checkpoints with minimum validation loss.

3.3. Results

As can be seen in Table 2, our AST model significantly outperforms both baselines in both parameter prediction accuracy (MSE) and audio reconstruction accuracy (SC), demonstrating its strength at predicting parameters for in-domain data, with a relatively small difference between our two baselines. The strong performance of the AST over the baselines in terms of audio reconstruction accuracy shows that the ability to reconstruct parameter sets does result in well reconstructed one-shots.

Table 2: *Results for parameter reconstruction mean-squared error (MSE) and audio reconstruction spectral convergence (SC).*

	MLP	**CNN**	**AST**
MSE	0.077	0.094	**0.031**
SC	4.608	5.372	**0.616**

Subjectively, the audio demos also reflect the strong performance of the AST-based sound matching model. For in-domain sounds, the AST model is broadly able to reconstruct the timbre and envelope of the input sound, albeit limited in expressivity and accuracy by the constrained set of 16 parameters it operates on. The AST is also able to reconstruct timbre and envelope for out-of-domain input sounds, suggesting this approach can be used to approximate arbitrary audio examples effectively, despite using only training data extracted from the synthesizer. One failure mode of the model seems to be in the modelling of oscillator pitch. This effect can be heard in Sample 8 and Sample 11 of the in-domain examples. In both of these, one oscillator is detuned, while the rest of the sample is reconstructed accurately. Oscillator pitch is one parameter where the discrepancy between parameter difference and sonic difference can be very high; a small difference in the tuning of one oscillator can introduce beating or dissonance, significantly affecting how a sample sounds. These kinds of issues could be a limitation of using a parameter-only loss function rather than incorporating an audio-based loss function, or a limitation of our data rendering and training strategy, where we do not vary the MIDI pitch of sounds in our training dataset, and do not train our model explicitly to predict the overall pitch of an input sound alongside other parameters.

4. CONCLUSION

In this paper, we provide some early evidence that our AST-based architecture, trained on a large synthetic dataset of randomly sampled one-shots from a software-based synthesizer, can yield strong performance in a synthesizer sound matching task. This could indicate the viability of such an architecture for general-purpose synthesizer sound matching, where sound matching systems could be created for further synthesizers with minimal architectural modifications, and without requiring them to be implemented differentiably. The ability of our AST-based architecture to generalize effectively also suggests arbitrary input sounds can be approximated effectively even when the training data is derived solely from the synthesizer itself.

As further work, we hope to scale our models to larger and more complex sets of synthesizer parameters and audio data. For more comprehensive modeling of synthesizers, we aim to investigate multi-output models that concurrently model continuous and categorical parameters, thus capturing the different kinds of parameters typically seen in synthesizers. We also aim to investigate specific architectural modifications for modeling routing and modulation in the context of synthesizer parameter matching. Finally, we intend to experiment further with our dataset rendering strategy to enable our models to match the pitch of a sound more effectively, including rendering samples at multiple pitches, and learning input pitch explicitly as a parameter.

5. ACKNOWLEDGMENTS

Thanks to the entire Audio Research team at Native Instruments for their support and feedback throughout the duration of this work. In particular, thanks to Johannes Imort for his assistance in setting up the demo website and Max Reifsteck for his assistance in dataset generation. Finally, thanks to the anonymous reviewers for their feedback on this work.

6. REFERENCES

[1] A. Horner, J. Beauchamp, and L. Haken, "Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis," *Comp. Mus. Jour.*, vol. 17, no. 4, pp. 17–29, 1993.

[2] K. Wehn et al., "Using ideas from natural selection to evolve synthesized sounds," in *Proc. of Dig. Aud. Eff. Work.*, 1998.

[3] N. Masuda and D. Saito, "Improving semi-supervised differentiable synthesizer sound matching for practical applications," *IEEE/ACM Trans. on Aud., Speech, and Lang. Proc.*, vol. 31, pp. 863–875, 2023.

[4] F. Caspe, A. McPherson, and M. Sandler, "DDX7: Differentiable FM synthesis of musical instrument sounds," in *Proc. Conf. of the Int. Soc. for Mus. Inf. Ret. (ISMIR)*, 2023.

[5] N. Uzrad et al., "Diffmoog: a differentiable modular synthesizer for sound matching," *arXiv preprint arXiv:2401.12570*, 2024.

[6] O. Barkan, D. Tsiris, O. Katz, and N. Koenigstein, "InverSynth: Deep Estimation of Synthesizer Parameter Configurations From Audio Signals," *IEEE/ACM Trans. on Aud., Speech, and Lang. Proc.*, vol. 27, no. 12, pp. 2385–2396, 2019.

[7] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, "Flow synthesizer: Universal audio synthesizer control with normalizing flows," *Applied Sciences*, vol. 10, no. 1, May 2020.

[8] D. Faronbi, I. Roman, and J.P. Bello, "Exploring approaches to multi-task automatic synthesizer programming," in *Proc. IEEE Int. Conf. on Ac., Speech and Sig. Proc. (ICASSP)*, 2023, pp. 1–5.

[9] A. Vaswani et al., "Attention is all you need," in *Proc. Conf. on Adv. in Neur. Inf. Proc. Sys. (NeurIPS)*, 2017, vol. 30.

[10] Y. Gong, Y.A. Chung, and J. Glass, "AST: Audio Spectrogram Transformer," in *Proc. Interspeech*, 2021, pp. 571–575.

[11] K. Koutini, J. Schlüter, H. Eghbal-zadeh, and G. Widmer, "Efficient Training of Audio Transformers with Patchout," in *Proc. Interspeech 2022*, 2022, pp. 2753–2757.

[12] C. Hawthorne, I. Simon, R. Swavely, E. Manilow, and J. Engel, "Sequence-to-Sequence Piano Transcription with Transformers," in *Proc. Conf. of the Int. Soc. for Mus. Inf. Ret. (ISMIR)*, 2021, pp. 246–253.

[13] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ," in *Proc. Inter. Conf. on Learn. Rep.*, 2021.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] P. Sobot, "Pedalboard," Available at https://doi.org/10.5281/zenodo.7817838, accessed May 20, 2024.

[16] S.O. Arık, H. Jun, and G. Diamos, "Fast spectrogram inversion using multi-head convolutional neural networks," *IEEE Sig. Proc. Let.*, vol. 26, no. 1, pp. 94–98, 2019.