

# ONE BILLION AUDIO SOUNDS FROM GPU-ENABLED MODULAR SYNTHESIS

*Joseph Turian \**

Spooky Audio

Berlin, Germany

lastname@gmail.com

*Jordie Shier \* , George Tzanetakis , Kirk McNally*

Computer Science and Music Technology

University of Victoria

Victoria, Canada

jordieshier@gmail.com

*Max Henry*

Music Technology Area

McGill University

Montreal, Canada

max.henry@mail.mcgill.ca

## ABSTRACT

We release synth1B1, a multi-modal audio corpus consisting of 1 billion 4-second synthesized sounds, paired with the synthesis parameters used to generate them. The dataset is 100x larger than any audio dataset in the literature. We also introduce torchsynth, an open source modular synthesizer that generates the synth1B1 samples on-the-fly at 16200x faster than real-time (714MHz) on a single GPU. Finally, we release two new audio datasets: FM synth timbre and subtractive synth pitch. Using these datasets, we demonstrate new rank-based evaluation criteria for existing audio representations. Finally, we propose a novel approach to synthesizer hyperparameter optimization.

## 1. INTRODUCTION

Machine learning (ML) progress has been driven by training regimes that leverage large corpora. The past decade has seen great progress in natural language processing (NLP) and vision tasks using large-scale training. As early as 2007, Google [1] achieved state-of-the-art machine translation results using simple trillion-token  $n$ -gram language models. Recent work like GPT3 [2] suggests that it is preferable to do less than one epoch of training on a large corpus, rather than multiple epochs over the same examples. Even tasks with little training data can be attacked using self-supervised training on a larger, related corpus followed by a transfer-learning task-specific fine-tuning step.

Audio ML research tends to lack many large-scale corpora, instead involving multiple epoch training on comparably small corpora compared to vision or NLP. Table 1 summarizes various large-scale and/or synthesizer audio corpora. Using AudioSet [9] requires scraping 5000 hours of YouTube videos, many of which become unavailable over time (thus impeding experimental control). FSD50K [4], a free corpus, was recently released to mitigate these issues, but contains only 108 hours of audio. To our best knowledge, the largest audio set used in published research is Jukebox [13], which scraped 1.2M songs and their corresponding lyrics. Assuming average song length is 4:20, we estimate their corpus is ≈90K hours.

In this paper we introduce synth1B1, a multi-modal audio corpus consisting of 1 billion 4-second synthesized sounds, which is 100x larger than any audio dataset in the literature. The dataset is

Name	Type	#hours	Multi-modal
Diva [3]	synth	12	parameters
FSDK50 [4]	broad	108	tags
NSynth [5]	notes	333	tags
Amp-Space [6]	guitar fx	525	parameters
LibriSpeech [7]	speech	1000	text
DAMP-VPB [8]	songs	1796	lyrics
AudioSet [9]	broad	4971	video+tags
YFCC100M [10]	broad	8081	video
Libri-Light [11]	speech	60000	weak labels
MSD [12]	songs	72222	tags+metadata
Jukebox [13]	songs	86667	lyrics+metadata
synth1B1	synth	1111111	parameters

Table 1: Large-scale and/or synthesizer audio corpora.

multi-modal in that each sound is paired with the corresponding synthesizer parameters used to generate it. To create such a large dataset, we built torchsynth,<sup>1</sup> a GPU-based open source modular synthesizer that builds the synth1B1 samples on-the-fly at 16200x faster than real-time. In order to compare the scope of sounds covered by synth1B1, we also generate two other datasets based on existing synthesizers and their human-made presets, and release them to the public with this paper: FM synth timbre<sup>2</sup> and subtractive synth pitch.<sup>3</sup> Using these datasets, we demonstrate new rank-based, synthesizer-motivated evaluation criteria for existing audio representations. Finally, we propose a novel approach to synthesizer hyperparameter optimization, and discuss how perceptually-correlated auditory distance measures could enable new applications in synthesizer design.

### 1.1. Background and Motivation

#### 1.1.1. Pre-training and learned representations

Tobin *et al.* [14] argue that learning over synthesized data enables transfer learning to the real-world. Multi-modal training offers additional benefits. Multi-model learning of audio—with video in [15] and semantic tags in [16]—has led to strong audio representations. Contrastive audio learning approaches like [17] can be used in multi-modal settings, for example by learning the correspondence between a synthesized sound and its underlying parameters. However, training such models is limited by small corpora and/or

<sup>1</sup><https://github.com/torchsynth/torchsynth>

<sup>2</sup><https://zenodo.org/record/4677102>

<sup>3</sup><https://zenodo.org/record/4677097>

the relatively slow synthesis speed of traditional CPU-based synths (Niizumi, p.c.).

### 1.1.2. Software Synthesizers

Programming audio synthesizers is challenging and requires a technical understanding of sound design to fully realize their expressive power. Many commercial synthesizers have well over 100 parameters that interact in complex, non-linear ways. One of the most commercially successful audio synthesizers, the Yamaha DX7, is notoriously challenging to program. Allegedly nine out of ten DX7s coming into workshops for servicing still have their factory presets intact [18].

Since the early 90s, researchers have leveraged advances in ML to develop a deeper understanding of the synthesizer parameter space and to build more intuitive methods for interaction [19]. Recently, deep learning has been used for programming synthesizers. Esling *et al.* [3] trained an auto-encoder network to program the U-He Diva using 11K synthesized sounds with known preset values. Yee-King *et al.* [20] used a recurrent network to automatically find parameters for Dexed, an open-source software emulation of the DX7.

### 1.1.3. Neural Synthesis

In contrast to traditional synthesis, neural synthesizers generate audio using large-scale machine learning architectures with millions of parameters [5]. Differentiable digital signal processing [21] bridged the gap between traditional DSP synthesizers with the expressiveness of neural networks, exploring a harmonic model-based approach, using a more compact architecture with 100K parameters. One benefit of synthesized audio is that the underlying factors of variation (*i.e.* the parameters) are known. We combine the ideas of traditional DSP and neural synthesis, yielding a greater level of simplicity and speed by building a GPU-optimal modular synthesizer. Our default voice has 78 latent parameters, which model traditional synthesizer parameters.

## 2. MAIN CONTRIBUTIONS

Our synth1B1 corpus and torchsynth software provide a fast, open approach for researchers to do large-scale audio ML pre-training and develop a deeper understanding of the complex relationship between the synthesizer parameter space and resulting audio. A variety of existing research problems can use synth1B1, including:

- Inverse synthesis, *i.e.* mapping from audio to underlying synthesis parameters. [20, 3]
- Inferring macro-parameters of synthesizers that are more perceptually relevant. [3, 22]
- Audio-to-MIDI. [23]
- Perceptual research, such as crafting perceptually motivated auditory representations and inferring timbre dimensions. [24]

Researchers can also use the synth1B1 corpus to take advantage of innovations from adjacent ML fields, namely: large-scale multi-modal, self-supervised, and/or contrastive learning, and transfer-learning through fine-tuning on the downstream task of interest, particularly tasks with few labeled examples.

## 2.1. synth1B1

synth1B1 is a corpus consisting of one million hours of audio: one billion 4-second synthesized sounds. The corpus is multi-modal in that each sound includes its corresponding synthesis parameters. We use deterministic random number generation to ensure replicability, even of noise oscillators. By default, one tenth of the examples are designated as the test set.

Data augmentation has been used on small-scale corpora to increase the amount of labeled training data. As discussed in §1, large-scale one-epoch training is preferable, which is possible using synth1B1's million-audio-hours.

Besides sheer size, another benefit of synth1B1 is that it is multi-modal: instances consist of both audio *and* the underlying parameters used to generate this audio. The use of traditional synthesis methods allows researchers to explore the complex interaction between synthesizer parameter settings and the resulting audio in a thorough and comprehensive way. Large-scale contrastive learning typically requires data augmentation (*e.g.* image or spectrogram deformations) to construct positive contrastive-pairs [25, 26]. However, this sort of faux-contrastive-pair creation is not necessary when the underlying latent parameters are known in a corresponding modality.

## 2.2. torchsynth

synth1B1 is generated *on the fly* by torchsynth 1.0. torchsynth is an open-source modular synthesizer and is GPU-enabled. torchsynth renders audio at 16200x real-time on a single V100 GPU. Audio rendered on the GPU can be used in downstream GPU learning tasks without the need for expensive CPU-to-GPU move operations, not to mention disk reads. Since it is faster to render synth1B1 *in-situ* than to download it, torchsynth includes a replicable script for generating synth1B1. To accommodate researchers with smaller GPUs, the default batchsize is 128, which requires between 1.9 and 2.4 GB of GPU memory, depending upon the GPU. If a train/test split is desired, 10% of the samples are marked as test. Because researchers with larger GPUs seek higher-throughput with batchsize 1024,  $9 \cdot 1024$  samples are designated as train, the next 1024 samples as test, *etc.* The default sampling rate is 44.1kHz. However, sounds can be rendered at any desired sample rate. Detailed instructions are contained in the torchsynth documentation for the precise protocol for replicably generating synth1B1 and sub-samples thereof.

## 2.3. Questions in Synthesizer Design, and New Pitch and Timbre Datasets and Benchmarks

To generate a synthesized dataset, one needs to sample the synthesis parameter space. Typically this is achieved through naïvely sampling parameters uniformly and rendering the resulting audio. Due to the complexity of the parameter space and potential interaction between parameters, such an approach would likely lead to a large number of redundant and/or undesirable sounds (*e.g.*, nearly silent renders, or those having an extreme fundamental frequency). The complexity of this parameter space leads to several open challenges in synthesizer design, specifically focusing on the task of designing and sampling parameters, including:

- How do you measure the apparent diversity of a synthesizer's sounds? How do you maximize it?
- Is there a parameter sampling strategy that results in audio sounds resemble a human-designed preset?

The main research barrier to solving these tasks computationally is the lack of an objective auditory distance, i.e. a perceptually-relevant audio dissimilarity measure [27]. A properly weighted dissimilarity measure could be used, for example, to tune our hyperparameter space to generate sounds that were maximally perceptually different, when torchsynth parameters are randomly sampled. We devise two auditory-distance evaluation methodologies, and concurrently release two datasets, each representing 22.5 and 3.4 hours of audio respectively, generated by the following open-source synthesizers: a DX7 clone and Surge, as DOI 10/f7dg and DOI 10/f652, respectively. Importantly, these datasets represent hand-crafted synthesizer sounds—i.e. presets designed by humans, not just a computer randomly flipping knobs—which we use in two ways: a) New benchmarks for evaluating audio representations. b) Evaluating the similarity of different sound corpora.

### 3. TORCHSYNTH DESIGN

#### 3.1. Synth Modules

torchsynth’s design is inspired by hardware modular synthesizers which contain individual units. Each module has a specific function and parameters, and they can be connected together in various configurations to construct a synthesizer. There are three types of modules in torchsynth: audio modules, control modules, and parameter modules. Audio modules operate at audio sampling rate (default 44.1kHz) and output audio signals. Examples include voltage-controlled oscillators (VCOs) and voltage-controlled amplifiers (VCAs). Control modules output control signals that modulate the parameters of another module. For speed, these modules operate at a reduced control rate (default 441Hz). Examples of control modules include ADSR envelope generators and low frequency oscillators (LFOs). Parameter modules simply output parameters. An example is the monophonic “keyboard” module that has no input, and outputs the note midi f0 value and duration.

To take advantage of the parallel processing power of a GPU, all modules render audio in batches. Larger batches enable higher throughput on GPUs. Figure 1 shows torchsynth’s throughput at various batch sizes on a single GPU. GPU memory consumption  $\approx 1216 + (8.19 \cdot \text{batch\_size})$  MB, including the torchsynth model. The default batch size 128 requires  $\approx 2.3$ GB of GPU memory. A batch of 4 of randomly generated ADSR envelopes is shown in Figure 2.

#### 3.2. Synth Architectures

The default configuration in torchsynth is the Voice, which is the architecture used in synth1B1. The Voice is made of the following modules: a Monophonic Keyboard, two LFOs, six ADSR envelopes (each LFO module includes two dedicated ADSRs: one for rate modulation and another for amplitude modulation), one Sine VCO, one SquareSaw VCO, one Noise generator, VCAs, a Modulation Mixer and an Audio Mixer. Modulation signals generated from control modules (ADSR and LFO) are upsampled to the audio sample rate before being passed to audio rate modules. Figure 3 shows the configuration and routing of the modules comprised by Voice. While the Voice is the default architecture of torchsynth 1.0, any number of synth architectures can be configured using the available modules. A 4-operator frequency modulation (FM) [28] synthesizer inspired by Ableton Live’s Operator instrument is currently in development.

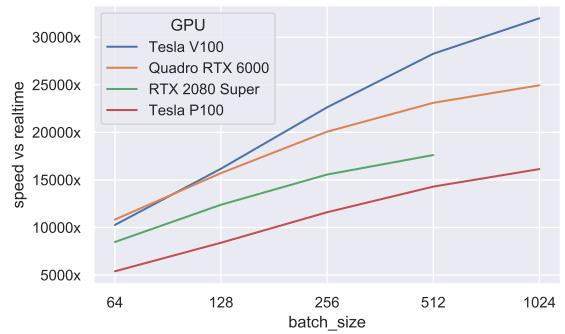


Figure 1: torchsynth throughput at various batch sizes.

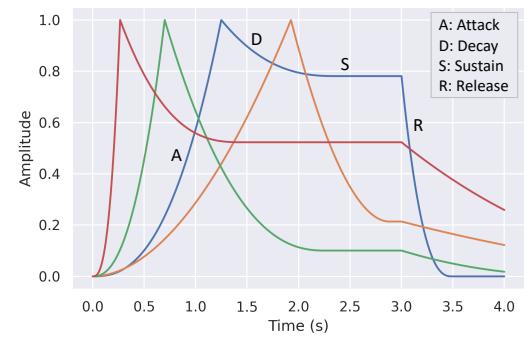


Figure 2: Batch of four randomly generated ADSR envelopes. Each section for one of the envelopes is labelled.

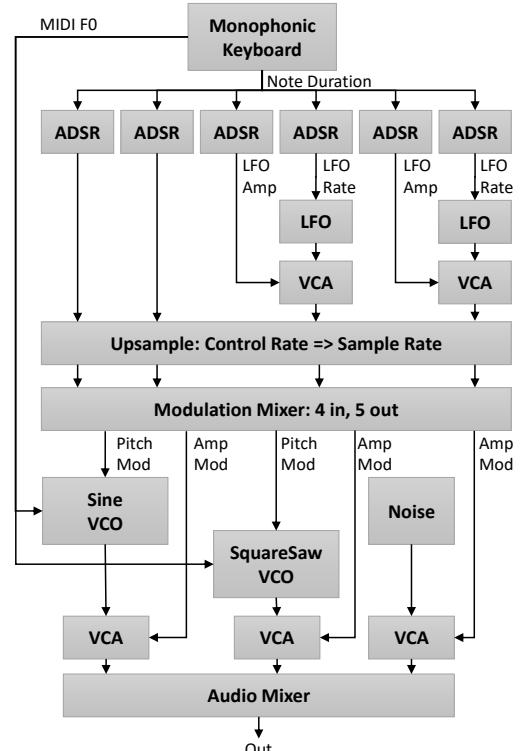


Figure 3: Module configuration for the Voice in torchsynth

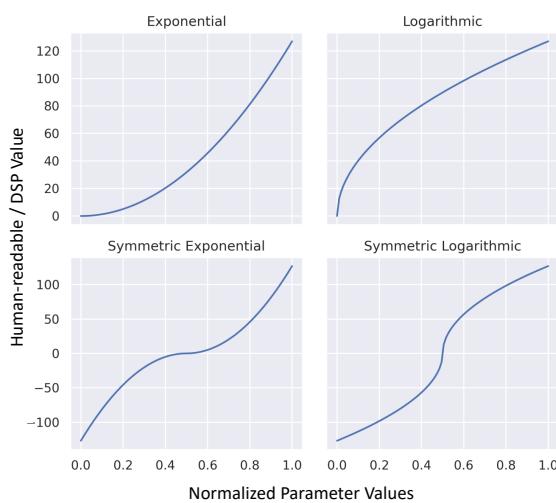


Figure 4: Examples of parameter curves used to convert to and from normalized parameter values and the human-readable values used in the DSP algorithms. The top two curves are non-symmetric curves, mapping to values in the range [0, 127]. The bottom two curves are symmetric, mapping to values in the range [-127, 127].

### 3.3. Parameters

Module parameters can be expressed in human-readable form with predetermined minimum and maximum values, such as  $0 \leq \text{midi } f_0 \leq 127$ . These values are used directly by the DSP algorithms of each module. Internally, parameters are stored in a corresponding normalized range  $[0, 1]$ . synth1B1 parameters are sampled uniformly from the normalized range; however, there is potentially a non-linear mapping between the internal range and the human-readable range. Besides the fixed min and max human-readable values, each parameter has two hyperparameters, “curve,” and “symmetry,” that determine how the internal  $[0, 1]$  values are transformed to the human-readable values. The curve can specify a particular logarithmic, linear, or exponential sampling approach. Symmetric curves, which alternately emphasize the center or edges of the distribution, are used for parameters that are centered at zero, and take on a range of both positive and negative values (such as oscillator tuning offset). An example set of non-linear curves is shown in Figure 4.

In our nomenclature, a particular choice of hyperparameter settings, which corresponds to a random sample space of markedly different sonic character, is called a *nebula*. The initial Voice nebula was designed by the authors based upon intuition and prior experience with synthesizers. We experiment with tuning the hyperparameters of Voice to generate different nebulae in §6.

## 4. EVALUATION OF AUDITORY DISTANCES

We seek (1) to quantify the diversity of sounds that can be generated with torchsynth within a particular nebula; and a similar problem, (2) to quantify to what extent a certain nebula can mimic the variability of sounds in another dataset. In order to do so, we first need a reliable measure of dissimilarity between pairs of sounds, also known as an auditory distance.

Auditory distances typically involve computing some multidimensional representation of a sound, then computing a distance over the representation space. Not all auditory distances are equally informative, depending on what is being measured; the L2 distance between two spectrograms, for example, carries little relative pitch information [29]. In our case, we are looking to quantify the perceptual diversity of our dataset.

### 4.1. Additional Datasets

To find a suitable distance, we devised two experiments using two new datasets. Sounds in each of the following datasets are RMS-level normalized using the normalize package.

#### 4.1.1. DX7 Timbre Dataset

Given 31K human-designed presets for the DX7, we generated 4-second samples on a fixed midi pitch (69 = A440) with a note-on duration of 3 seconds (using this DX7 clone). For each preset, we varied only the velocity, from 1–127. This dataset is built on the assumption that velocity effects a meaningful, monotonic variation in timbre when it is explicitly programmed into a DX7 patch. Not all DX7 patches are velocity sensitive, and some are more sensitive than others. In our generation process, sounds that were completely identical—*i.e.* each sample matched with error 0—were removed. 8K presets had only one unique sound. The median was 51 unique sound per preset, mean 41.9, std dev 27.4.

#### 4.1.2. Surge Pitch Dataset

To explore a second dimension of variability, in this case *pitch*, we used the Surge synthesizer Python API and the 2.1K standard Surge presets. The open-source Surge synthesizer is a versatile subtractive synthesizer with a variety of oscillator algorithms. Here we held the velocity constant at a value of 64, and varied midi pitch values from 21–108, the range of a grand piano. Only a small percentage of presets (like drums and sound effects) had no meaningful pitch variation, and thus no perceptual ordering as pitch increases. Therefore, a small fraction of presets are unclassifiable, imposing a uniform upper bound in accuracy across the board for all auditory distances.

### 4.2. Experiments

#### 4.2.1. Distance Experiment 1: Timbral and Pitch Ordering Within a Preset

In this experiment, we measure the ability of an auditory distance to order sounds by “timbre,” or by pitch, in the DX7 and Surge datasets, respectively. In effect, the experiment is two evaluations in parallel, run on two separate datasets.

We sample a random preset with at least 3 unique sounds. For each sound  $s$ , we pick a random sound  $s_l$  from this preset with a lower rank (using the DX7 set, this would be a sound having the same pitch but a lower velocity; for the Surge dataset this is a sound having the same velocity but lower pitch); and a random sound  $s_h$  with higher rank.

For each of  $s$ ,  $s_l$  and  $s_h$ , we compute the distance  $d(\cdot, \hat{s})$  between this sound and all other sounds  $\hat{s}$  in the dataset. While  $s$  is the sample of interest, distance measures are strictly non-negative. Therefore, we seek a concurrent metric to determine whether the compared sound  $\hat{s}$  is “above” or “below”  $s$ . If the sound  $\hat{s}$  is closer

Representation	model choice	normed	Spearman with a preset		DCG across presets	
			Surge pitch	DX7 velocity	mean	Surge
OpenL3 [30]	env, mel256, 6144		<b>0.821</b>	<b>0.746</b>	0.896	0.880
OpenL3 [30]	env, mel256, 6144	✓	<b>0.821</b>	<b>0.747</b>	0.895	0.809
OpenL3 [30]	music, mel256, 6144	✓	0.817	0.732	<b>0.903</b>	0.820
OpenL3 [30]	music, mel256, 6144		0.813	0.722	<b>0.903</b>	0.892
Coala [16]	dual_ae_c	✓	0.813	0.729	0.896	0.555
Coala [16]	dual_e_c	✓	0.811	0.737	0.884	0.569
NSynth Wavenet [5]		✓	0.810	0.717	<b>0.903</b>	0.582
OpenL3 [30]	music, linear, 6144		0.808	0.722	0.895	0.874
OpenL3 [30]	music, mel256, 512		0.804	0.710	0.899	<b>0.904</b>
OpenL3 [30]	music, mel256, 512	✓	0.801	0.705	0.897	0.585
NSynth Wavenet [5]			0.789	0.675	<b>0.903</b>	0.835
Coala [16]	dual_ae_c		0.776	0.658	0.893	0.748
Coala [16]	dual_e_c		0.750	0.630	0.871	0.681
Multi-scale spectrogram [21, 31]	linear+log, [4096 ... 64]		0.792	0.690	0.894	0.543
Multi-scale spectrogram [21, 31]	log, [4096 ... 64]		0.786	0.689	0.884	0.542
Multi-scale spectrogram [21, 31]	linear, [4096 ... 64]		0.658	0.410	<b>0.905</b>	0.447
Coala [16]	cnn	✓	0.555	0.303	0.806	0.485
Coala [16]	cnn		0.552	0.297	0.807	0.714
						0.614
						0.815

Table 2: Performance of representations on experiments defined in § 4.2.1 and 4.2.2. Best scores, and scores within 0.002 of the best, are bold-faced.  $\ell_1$  distance was used because it outperformed  $\ell_2$ . We sort by mean spearman within a preset.

to  $s_l$ , we determine the sign of the distance to  $s$  to be negative. If  $\hat{s}$  is closer to  $s_h$ , we determine the sign of the distance to  $s$  to be positive. As a result, we have a signed distance metric comparing the sound  $s$  to every other sound in the dataset.

This set of distances is then correlated to the ground-truth index of pitch, or velocity (depending on the dataset). The correlation, here a Spearman rank correlation, reflects the extent to which the signed distance can properly order the dataset by variability in pitch or velocity. One limitation of this methodology for inducing a forced ranking from simple distance is that if, say,  $s = 80$ ,  $s_l = 31$ ,  $s_h = 81$ , and  $\hat{s} = 79$ , we might judge  $\hat{s}$  as closer to  $s_h$  and thus above  $s$ . We controlled for this by using the same choice for every auditory distance of  $s_h$  and  $s_l$  given  $s$ .

Formally, we estimate:

$$\mathbb{E}_{S \in P, s \in S, s_l, s_h \sim S, s_l < s < s_h} \left[ \rho_{\hat{s} \in S} \left( \text{rank}(\hat{s}), d(s, \hat{s}) \cdot \text{sgn}(d(s_h, \hat{s}) < d(s_l, \hat{s})) \right) \right] \quad (1)$$

$P$  is the set of presets,  $S$  sounds in that preset, and  $\rho$  is spearman.

#### 4.2.2. Distance Experiment 2: Determine a Synthesis Preset

In this constrained environment, a good distance measure should have a relatively low distance between sounds generated by the same preset. For each trial, we sample 200 different presets. We sample 2 unique sounds from each preset. For each sound, we compute its distance against the 399 other sounds, and then compute the discounted cumulative gain (DCG) [32] of the sound from the same preset, with binary relevance. The DCG is computed for all 400 sounds in the trial. We perform 600 trials.

In the pitch dataset, to control for the helical nature of pitch perception [33], the second sound was always an interval of six semitones (AKA a tritone, *diabolus in musica*) from the first note.

This ensures that pitches avoid similar partials due to overlapping harmonics that could be easily matched.

### 4.3. Evaluation Results

We audition the following distances: a) The multi-scale spectrogram distance has been used in a variety of applications, particularly in speech synthesis [34, 35] but also in music [13, 21, 36]; b) NSynth Wavenet [5] is a Wavenet-architecture trained on NSynth musical notes; c) OpenL3 [30] was trained multi-modally on AudioSet audio and video, on two distinct subsets: music and environmental sounds; d) Coala [16] was trained multi-modally on Freesound audio and their corresponding tags.

We experimented with a variety of hyperparameter settings for the representations. The best results are in Table 2. We use  $\ell_1$  distance because it gave better results than  $\ell_2$  across the board. For Coala and NSynth Wavenet, normalizing improves the spearman scores, but harms the DCG across presets. Normalization had little effect on OpenL3. OpenL3 (music, mel256, 512) achieves the best score on DCG across presets, and its compactness makes it an appealing choice for the remaining experiments in the paper.

## 5. SIMILARITY BETWEEN AUDIO DATASETS

To evaluate the similarity between two datasets of audio samples  $X$  and  $Y$ , we use the maximum mean discrepancy (MMD) [37]. We use the following MMD formulation, assuming  $X$  and  $Y$  both have  $n$  elements:

$$\text{MMD}(X, Y) = \frac{1}{nn} \sum_{i,j=0}^n 2 \cdot d(x_i, y_j) - d(x_i, x_j) - d(y_i, y_j) \quad (2)$$

MMD allows us to use our chosen distance measure—OpenL3 (music, mel256, 512)  $\ell_1$ —as the core distance  $d$ .

For Surge and DX7, we selected sounds with midi pitch 69 and velocity 64. We also generated a set of 4-second samples of

MMD	std	corpus 1	corpus 2
4.396	0.123	white	white
21.409	4.729	dx7	dx7
23.732	3.615	FSD50K	FSD50K
24.130	5.251	torchsynth	torchsynth
27.824	9.821	surge	surge
2751.519	80.955	torchsynth	surge
2884.843	67.264	surge	dx7
3001.857	71.888	torchsynth	FSD50K
3637.845	79.265	torchsynth	dx7
4756.952	112.705	surge	FSD50K
7413.105	111.897	dx7	FSD50K
13202.202	61.558	white	FSD50K
16985.319	92.992	white	torchsynth
18488.926	67.277	white	surge
20374.929	78.886	white	dx7

Table 3: MMD results comparing different audio sets, including the stddev of the MMD over the 1000 trials.

white-noise, and used excerpts from the FSD50K evaluation set [4], which is broad-domain audio, trimmed to 4 seconds. From each corpus, we randomly sampled 2000 sounds, to match the size of the smallest corpus (Surge). We performed 1000 MMD trials, each time comparing  $n = 1000$  sounds from one corpus to  $n = 1000$  sounds from another, randomly sampled each trial. To estimate the diversity within a particular corpus, we evaluated MMD over 1000 distinct 50/50 partitions of the corpus.

Table 3 shows the result of average MMD computations between different audio corpora. 0.0 would be perfectly identical, only occurring if the two corpora had identical sounds. Some results are expected, whereas some are counter-intuitive and suggest potential issues in our use of the OpenL3 distance measure. These results are sometimes perceptually incoherent, and suggest that the use of the auditory distance measures explored may impede progress in automatic synthesizer design, as we will illustrate in the following section.

- White-noise is the most similar to itself of all comparisons.
- FSD50K broad-domain sounds are, strangely, considered to have less within-corpus diversity than torchsynth or Surge sounds. However, the variance is high enough that it is hard to have statistical confidence in this unexpected result.
- More troubling are low-variance estimates that torchsynth is more similar to FSD50k than it is to the dx7 synth. *A priori*, one would expect that synthesizers would sound more similar to each other than broad domain audio.
- White noise is the least similar to DX7 synth sounds of all corpora.

## 6. TORCHSYNTH HYPER-PARAMETER TUNING

How can we guarantee the maximum diversity of sounds within a particular nebula? Similarly, to what extent can torchsynth adopt the characteristics of a separate given corpus of audio? Recall from §3 and Figure 4 that for each module parameter, the choice of scaling curve is a hyperparameter. Initial hyperparameters were chosen perceptually and based upon prior-knowledge of typical synth design.

In principle, we can use MMD (Equation 2) as an optimization criterion to tune these hyperparameters a) to maximize sonic diversity; or b) model the characteristics of another dataset. We use Optuna [38], initializing with 200 random grid-search trials, and subsequently using CMA-ES sampling for 800 trials. In each trial, we generate 256 random torchsynth sounds with the Optuna-chosen hyperparameters. Hyperparameter curves were sampled log-uniform in the range [0.1, 10]. The top 25 candidates were re-evaluated using 30 different MMD trials, to pick the best hyperparameters. However, MMD estimates are only as good as the underlying similarity metric (OpenL3- $\ell_1$ ) that it uses.

For these experiments, the authors and non-author musicians conducted blinded listening experiments of the tuned nebula and our manually-chosen nebula, and listened to 64 random sounds. Only after independent qualitative evaluation did we unblind the nebulae.

### 6.1. Restricting hyperparameters

Many torchsynth 1.0 Voice sounds (the default nebula) have an eerie sci-fi feel to them. To find the drum nebula, we used Optuna to choose hyperparameters to *minimize* the OpenL3- $\ell_1$ -MMD against 10K one-shot percussive sounds [39]. In this experiment, no hyperparameters were frozen; all were permitted to be tuned at once. We had hoped to find that OpenL3- $\ell_1$ -MMD would find appropriate percussive curves.

Overall, we found the sounds of the resulting drum nebula unpleasant to listen to. Many of the sounds did not resemble percussion, and others made use of extreme use of high and low oscillator tunings. We suspect that the many wide pitch-modulation sweeps present in the resulting audio were an attempt on the part of the optimizer to match the broadband energy in the target drum transients.

Curious to see if this negative result was due to a failure of the distance measure, or instead a systemic limitation in the design of the torchsynth 1.0 Voice, we next hand-tuned the hyperparameters to create a sensible drum nebula, which is shared as part of our repository. Many of the resulting sounds have a quality akin to early drum machines, and the distribution of sounds is overall much more percussion-like. We encourage the reader to listen to this nebula, which is available on torchsynth site.

The process of hand-designing this drum nebula revealed to the authors one clear limitation in torchsynth 1.0: all synth parameters are sampled independently. In kick drums, for example, low-tuned oscillators tend to correlate to short, snappy envelope settings; but such envelopes are not appropriate for all percussive sounds. In future work, we are interested in investigating multivariate sampling techniques, which would allow more focused cross-parameter modal sound sampling.

### 6.2. Maximizing torchsynth diversity

We also attempted to tune our hyperparameters to maximize torchsynth MMD, i.e. increase the perceptual diversity of sounds generated by torchsynth itself. As before, Optuna was used to choose hyperparameters that maximized the OpenL3- $\ell_1$ -MMD and thus increase the diversity of sounds. Nonetheless, the “optimized” nebula exhibited pathologies in pitch, favoring extremely low and high pitches. It may be that OpenL3- $\ell_1$  overestimates perceptually diversity in these frequency ranges. We performed numerous experiments restricting the hyperparameters Optuna was permitted

to modify, such as prohibiting changes to midi f0 and VCO tuning and mod depth. Consistently, listeners preferred our manually design nebula to automatically designed ones in blind tests. We consider this another important negative result that points to the need for further work in automatic synthesizer design.

## 7. OPEN QUESTIONS, ISSUES, AND FUTURE WORK

Many of our experiments in automatic synthesizer design hinge on having a perceptually-relevant auditory distance measure. OpenL3 (music, mel256, 512)  $\ell_1$  performed well on our quantitative synthesizer experiments (Table 2), but exhibited some issues in the context of this task in qualitative listening tests, in particular its insensitivity to extreme pitch and inability to model percussion.

Learning a perceptually-relevant auditory distance measure is an open research question. Manocha *et al.* [40] use manually-annotated “just noticeable differences” (JND) trials generated using active learning to induce a perceptual distance measure. However, these experiments only work with speech and do not include pitch variations, so their model was inappropriate for our task.

The lack of a perceptually accurate auditory distance measure, at least in the context of this task, prevented us from precisely estimating the perceptual diversity of sounds that is expressible by torchsynth, as well as other synthesizers like Surge and DX7. We present, then, the following open question: How do we craft an auditory distance measure that can *perceptually* measure (and thereby optimize) synthesizer diversity, or similarity to an existing sound corpus?

A perceptually-relevant auditory distance measure for music opens the door to many possible advances in synthesizer design, including: estimating and maximizing the diversity of synthesizer, mimicking existing synthesizers through automation, inverse synthesis, automatic transcription, and the other tasks described in §2.

torchsynth 1.0 focuses on high throughput and creating a diverse synth1B1 dataset. There are a handful of improvements we want to add to torchsynth: 1) Stress-tested differentiable modules, 2) Subtractive filters, 3) Additional architectures including FM synthesis, 4) Multivariate parameter selection, 5) High-throughput modules that resemble human speech, and 6) A standardized modular architecture for high-throughput audio *effect* research.

Despite these open questions, we believe that the synth1B1 corpus is a significant and useful contribution to the world of audio ML research, for its enormous size, speed, and corresponding multi-modal latent parameters.

## 8. CONCLUSIONS

We release synth1B1, a multi-modal corpus of synthesizer sounds with their corresponding latent parameters, generated on-the-fly 16200x faster than realtime on a single V100 GPU. This corpus is 100x bigger than any audio corpus present in the literature. Accompanying this dataset is the open-source modular GPU-optimal torchsynth package. We hope that larger-scale multi-modal training will help audio ML accrete the benefits demonstrated by previous NLP and vision breakthroughs.

We freely release pitch and timbre datasets based upon human-designed synthesizer presets, and novel evaluation tasks on which we benchmark a handful of audio representations. We present several novel research questions, including how to estimate and maximize the diversity of a synthesizer, as well as how to mimic existing synthesizers. We outline issues and open research questions

that currently impede this sort of experimental work, in particular demonstrating negative results of auditory distance measures.

## 9. ACKNOWLEDGMENTS

The authors wish to thank Alexandre Défossez, Carson Gant, Colin Malloy, Dan Godlovitch, Gyanendra Lucky Das, Humair Raj Khan, Nicolas Pinto, and Will Maclellan and the reviewers for their help.

## 10. REFERENCES

- [1] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean, “Large language models in machine translation,” in *EMNLP-CoNLL*, 2007.
- [2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, and Pranav Shyam *et al.*, “Language models are few-shot learners,” in *NeuroIPS*, 2020.
- [3] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos, “Universal audio synthesizer control with normalizing flows,” in *DAFx*, 2019.
- [4] Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra, “FSD50K: an open dataset of human-labeled sound events,” *CoRR*, vol. abs/2010.00475, 2020.
- [5] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan, “Neural audio synthesis of musical notes with wavenet autoencoders,” in *ICML*. PMLR, 2017.
- [6] Jason Naradowsky, “Amp-Space: A large-scale dataset for fine-grained timbre transformation,” in *DAFx*, 2021.
- [7] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, “LibriSpeech: An ASR corpus based on public domain audio books,” in *ICASSP*, 2015.
- [8] Smule Inc., “DAMP-VPB: Digital Archive of Mobile Performances - Smule Vocal Performances Balanced,” 2017.
- [9] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter, “Audio Set: An ontology and human-labeled dataset for audio events,” in *ICASSP*, 2017.
- [10] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li, “YFCC100M: The new data in multimedia research,” *Communications of the ACM*, 2016.
- [11] Jacob Kahn, Morgane Rivière, Weiyi Zheng, Evgeny Kharitonov, Qiantong Xu, Pierre-Emmanuel Mazaré, Julien Karadayi, Vitaliy Liptchinsky, Ronan Collobert, Christian Fuegen, Tatiana Likhomanenko, Gabriel Synnaeve, Armand Joulin, Abdelrahman Mohamed, and Emmanuel Dupoux, “Libri-light: A benchmark for asr with limited or no supervision,” in *ICASSP*, 2020.
- [12] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere, “The Million Song Dataset,” in *ISMIR*, 2011.

- [13] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever, “Jukebox: A generative model for music,” *arXiv preprint arXiv:2005.00341*, 2020.
- [14] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.
- [15] Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello, “Look, listen, and learn more: Design choices for deep audio embeddings,” in *ICASSP*, 2019.
- [16] Xavier Favory, Kostantinos Drossos, Tuomas Virtanen, and Xavier Serra, “Coala: Co-aligned autoencoders for learning semantically enriched audio representations,” in *ICML Workshop on Self-supervision in Audio and Speech*, 2020.
- [17] Aaqib Saeed, David Grangier, and Neil Zeghidour, “Contrastive learning of general-purpose audio representations,” in *ICASSP*, 2021.
- [18] Allan Seago, Simon Holland, and Paul Mulholland, “A critical analysis of synthesizer user interfaces for timbre,” in *BCS-HCI*. 2004, Research Press International.
- [19] Andrew Horner, James Beauchamp, and Lippold Haken, “Machine tongues XVI: Genetic algorithms and their application to fm matching synthesis,” *Computer Music Journal*, 1993.
- [20] Matthew John Yee-King, Leon Fedden, and Mark d’Inverno, “Automatic programming of VST sound synthesizers using deep networks and other techniques,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.
- [21] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts, “DDSP: Differentiable digital signal processing,” in *ICLR*, 2020.
- [22] Kivanç Tatar, Daniel Bisig, and Philippe Pasquier, “Latent timbre synthesis,” *Neural Computing and Applications*, 2020.
- [23] Curtis Hawthorne, Andrew Stasyuk, Adam Roberts, Ian Simon, Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck, “Enabling factorized piano music modeling and generation with the maestro dataset,” in *ICLR*, 2019.
- [24] Cyrus Vahidi, George Fazekas, Charalampos Saitis, and Alessandro Palladini, “Timbre space representation of a subtractive synthesizer,” in *Timbre 2020*, 2020.
- [25] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton, “A simple framework for contrastive learning of visual representations,” in *ICML*, 2020, PMLR.
- [26] Daisuke Niizumi, Daiki Takeuchi, Yasunori Ohishi, Noboru Harada, and Kunio Kashino, “BYOL for audio: Self-supervised learning for general-purpose audio representation,” *CoRR*, vol. abs/2103.06695, 2021.
- [27] John M. Grey, “Multidimensional perceptual scaling of musical timbres,” *Journal of the Acoustical Society of America*, 1977.
- [28] John M. Chowning, “The synthesis of complex audio spectra by means of frequency modulation,” *Journal of the Audio Engineering Society*, 1973.
- [29] Joseph Turian and Max Henry, “I’m sorry for your loss: Spectrally-based audio distances are bad at pitch,” in *ICBINB@NeurIPS*, 2020.
- [30] Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello, “Look, listen and learn more: Design choices for deep audio embeddings,” in *ICASSP*, 2019.
- [31] Christian J. Steinmetz and Joshua D. Reiss, “auraloss: Audio focused loss functions in PyTorch,” in *DMRN+15*, 2020.
- [32] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu, “A theoretical analysis of NDCG type ranking measures,” in *COLT*. 2013, PMLR.
- [33] Roger N. Shepard, “Geometrical approximations to the structure of musical pitch,” *Psychological review*, 1982.
- [34] Xin Wang, Shinji Takaki, and Junichi Yamagishi, “Neural source-filter waveform models for statistical parametric speech synthesis,” *IEEE ACM Trans. Audio Speech Lang. Process.*, 2020.
- [35] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim, “Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram,” in *ICASSP*, 2020.
- [36] Adrien Bitton, Philippe Esling, and Tatsuya Harada, “Neural granular sound synthesis,” *CoRR*, vol. abs/2008.01393, 2020.
- [37] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola, “A kernel two-sample test,” *JMLR*, 2012.
- [38] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *ACM SIGKDD*, 2019.
- [39] Antonio Ramires, Pritish Chandna, Xavier Favory, Emilia Gómez, and Xavier Serra, “Neural percussive synthesis parameterised by high-level timbral features,” in *ICASSP*, 2020.
- [40] Pranay Manocha, Adam Finkelstein, Richard Zhang, Nicholas J. Bryan, Gautham J. Mysore, and Zeyu Jin, “A differentiable perceptual audio metric learned from just noticeable differences,” in *Interspeech*, 2020.