BLIND ARBITRARY REVERB MATCHING

Andy Sarroff and Roth Michaels

iZotope, Inc.
Cambridge, MA, USA
asarroff@izotope.com

ABSTRACT

Reverb provides psychoacoustic cues that convey information concerning relative locations within an acoustical space. The need arises often in audio production to impart an acoustic context on an audio track that resembles a reference track. One tool for making audio tracks appear to be recorded in the same space is by applying reverb to a dry track that is similar to the reverb in a wet one. This paper presents a model for the task of "reverb matching," where we attempt to automatically add artificial reverb to a track, making it sound like it was recorded in the same space as a reference track. We propose a model architecture for performing reverb matching and provide subjective experimental results suggesting that the reverb matching model can perform as well as a human. We also provide open source software for generating training data using an arbitrary Virtual Studio Technology plug-in.

1. INTRODUCTION

The audio production industry relies on audio engineers to create consistent, coherent narratives out of audio arriving from multiple sources. For instance, re-recording engineers and mixers need to mix audio from Automated Dialogue Replacement (ADR) into scenes that already have production dialogue. The ADR audio must be indistinguishable from originally recorded production dialogue. Documentary editors receive audio and video from several sources and combine them together with voiceover to tell a story. Audio segments might be sourced across multiple years and recorded on different devices and microphones, as well as in different acoustic environments. Podcasters interview subjects in several spaces, but they want all sources to sound like they're having a conversation in the same room. Subtle differences in audio can disorient listeners, distracting them from the narrative. Creating cohesion is integral to providing a good listening experience.

Consistent acoustic cues, such as reverb, are essential to creating the perception that all sources were recorded in the same place [1]. In practice, an audio engineer might spend an enormous amount of time hand-tweaking parameters of an artificial reverb generator to make dry audio sources sound similar to reverberant ones. This is a challenging task because there may be nearly infinite combinations of reverb generation parameters. In other words, the audio engineer must manually search an incredibly large parameter space to find the right sound. On top of this, certain combinations of reverb parameters may interact with each other in hard-to-predict ways.

In this paper, we present a model for the task of "reverb matching." The model takes as input a reverberant audio signal and infers the parameters or preset of an artificial reverb generator. These, in turn, may be used to add reverb to a non-reverberant target audio signal, saving an audio engineer the time-consuming effort of manually searching the reverb parameter space. We also provide open source software for generating a large amount of training data through many audio plug-in instances running in parallel.

In the following section we provide an overview of related work. Section 3 defines the reverb matching task, and Section 4 explains how we created a custom dataset for the task. Next we describe the model architecture in Section 5. Finally, we provide the statistical results of an experiment involving human subjects (Section 6), followed by concluding remarks in Section 7.

2. RELATED WORK

The reverb matching task falls under the more general category of "sound matching," which has recently gained interest in the research community. In the sound matching task, there exists some target audio, and we wish for a model to infer a set of parameters or a preset corresponding to an audio processing unit. When populating the audio processor with the inferred parameters or preset, it should be able produce audio that sounds similar to our target audio. Most papers have attended to the task of sound matching for virtual instrument synthesizers, which are capable of generating a highly diverse set of sounds, some of which imitate acoustic instruments. In these cases a model is fed a target signal, and the model infers the set of parameters associated with a synthesizer that will generate audio having similar pitch, timbre, and envelope characteristics.

Perhaps one of the first examples of sound matching for virtual instrument synthesizers can be found in [2], which proposes a simple linear regression model for inferring the parameters to an arbitrary synthesizer. The authors test their model on a wide range of synthesizers. Hand-picked features such as root mean square energy and zero-crossing rate are the input to their model, along with meta-features like first and second order time derivatives of the features. Using principal components analysis (PCA), the authors attempt only to infer parameters that contribute the highest variance to their training data. Their model is capable of matching sounds of arbitrary duration, as is ours. However, between the design choices of linear model, hand crafted input features, and method of choosing parameters, we expect that their approach would have difficulty matching sounds where there's a high degree of interdependence between parameters.

Modern approaches to matching the sound of virtual instrument synthesizers use deep learning methods. The real-valued parameters of the Dexed synthesizer plug-in, which is based on Virtual Studio Technology (VST), are matched in [3] from target

sounds transformed to Mel-frequency cepstral coefficients. The authors test the efficacy of several neural networks, including one that has, similar to our proposed model, a bidirectional recurrent layer. One of the advantages of using a recurrent network is that input sequences may be of arbitrary length. However their models were apparently only trained and tested on audio having a duration of 1 sec. In our case, we wish to match reverberant audio which may be of arbitrary duration.

A custom synthesizer employing subtractive and frequency modulation synthesis is matched in [4] using a deep convolutional model. The input to the model is either the Short-time Fourier transform (STFT) or raw audio and the outputs are the real-valued parameters of the synthesizer, quantized to 16 discrete levels and represented as one-hot vectors. This is the only work in our review that conducted a formal listening test, as we do. Listening tests are arguably at least as important as objective tests for evaluating sound matching models, as it is difficult to map objective test error to perceptual judgment of quality. Yet there exists several fundamental differences between our work and theirs. For instance, we do not quantize real-valued parameters and we use a recurrent neural net architecture. The convolutional architectures proposed by [4] expect fixed-length input sequences and have small kernel sizes providing receptive fields that are at most fractions of a second long. Our architecture has no constraints on the signal length and should be able to capture longer-term dependencies due to the gated recurrent architecture. We believe these differences may be important to characterize important attributes such as the length and shape of a reverberant tail.

The Flow Synthesizer [5] is perhaps the most sophisticated model, at the time of writing, for matching virtual instrument synthesizer sounds. The authors of the Flow Synthesizer present a generative model that includes a variational autoencoder which provides a latent space for the target audio. The distribution over the latent space is transformed through a bijective regression flow to the parameters of the Diva synthesizer. As a result, the Flow Synthesizer can simultaneously learn a lower dimensional manifold for reconstructing the target audio, as well as a mapping from the manifold to the parameters of the synthesizer. The model is trained on real-valued parameters to the synthesizer and, similar to [2], only attempts to infer parameters leading to the greatest variability in the data, as determined by PCA.

The sound matching task has not been applied to many domains outside of virtual instrument synthesizers. An example of reverb matching (which is explicitly defined in Section 3) can be found in [6]. The authors first train a universal background model (UBM) of reverberant audio. They subsequently adapt the UBM to a Gaussian mixture model for each preset associated with a set of reverb generators. This approach does not scale well when the number of presets under consideration becomes large.

There have also been several proposals for "acoustic matching." For instance, [7] suggests a method for characterizing recordings so that audio embeddings associated with the same impulse response have smaller geometric distance than those associated with differing impulse responses. The reference and target embeddings are subsequently used as conditioning signals in a waveform-to-waveform model to transfer the acoustic context from one recording to the next. An important difference between the work of [7] and ours is that the former relies on strong assumptions that the reverberant recordings can be fully characterized by an impulse response.

A defining characteristic of our approach is the fact that our system is designed to match natural as well as unnatural reverb. Our model makes no assumptions about whether the reverb can be characterized by an impulse response or that it was produced by a linear time invariant system such as a physical acoustic space. There is no requirement that the reverberant signal be decomposed into a dry source and reverb kernel.

The reverb matching task has constraints that make it unique from tasks that involve matching the sounds of a virtual instrument synthesizer. First, a reverb generator may be thought of as an audio "effect," where an underlying audio source undergoes processing and is mixed again with the original source. Because of this, we want the model to "ignore" the underlying source material, making an inference that is independent of the affected source. The model must be able to reliably infer the correct reverb settings, even when the reverberated target audio has been generated using different source material. Second, the duration of a reverberated source may vary greatly according to the reverb time of the reverb generator. Virtual instrument synthesizers may also exhibit varying duration due to changes in release time, but the audible tail of reverb may last many seconds, a less likely scenario with typical virtual instrument synthesizer patches.

3. REVERB MATCHING

Consider an artificial reverb generator that reverberates an input signal and mixes the reverberated output with the input. We denote the set of all possible input signals, output signals, and parameters that control the characteristics of the reverb generator as $\mathcal{X}$, $\mathcal{Y}$, and Φ respectively, and we define a family of reverb generating functions

$$\mathcal{G} = \{\boldsymbol{g}_{\boldsymbol{\phi}} : \mathcal{X} \mapsto \mathcal{Y} \mid \boldsymbol{\phi} \in \Phi\}. \tag{1}$$

(We denote vector-valued functions and variables in boldface, whereas scalar values are denoted in normal typeface. Vector elements are indicated using subscripts.) Thus, given an arbitrary dry input signal $\boldsymbol{x} \in \mathcal{X}$, mixing ratio $\pi \in [0, 1]$, and reverb parameters $\boldsymbol{\phi} \in \Phi$, a particular reverb system is summarized as following.

$$\begin{aligned} \boldsymbol{y} &= \pi\, \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}) + (1 - \pi)\, \boldsymbol{x} \\ &\in \mathcal{Y}. \end{aligned} \tag{2}$$

In the reverb matching task, we'd like to recover $\boldsymbol{g}_{\boldsymbol{\phi}}$ and π from $\boldsymbol{y}$, without knowing anything about $\boldsymbol{x}$. Notably, $\boldsymbol{g}_{\boldsymbol{\phi}}$ may be non-differentiable so that a direct solution is not possible or practical. We approach this task by training a temporally adaptive neural net $\boldsymbol{f}$ parameterized by $\boldsymbol{\theta}$ so that, building on Equation 2, the following holds true.

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{y}) = [\hat{\pi}, \boldsymbol{g}_{\hat{\boldsymbol{\phi}}}] \quad \text{and} \tag{3}$$

$$\begin{aligned} \hat{\boldsymbol{y}} &= \hat{\pi}\, \boldsymbol{g}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{x}) + (1 - \hat{\pi})\, \boldsymbol{x} \\ &\approx \boldsymbol{y}. \end{aligned} \tag{4}$$

In this work, we take a data-driven approach to optimizing the parameters of the model. Due to the novelty of the task, there are no off-the-shelf datasets available for reverb matching so we must build a custom dataset. Our method for making training data is described in general terms in the following section. (More explicit details related to our experiment are provided in Section 6.)

4. DATA

We build our own custom dataset by first choosing an artificial reverb generator $\mathcal{G}$ that is capable of producing enough different styles of reverb to satisfy our domain and codomain of interest, $\mathcal{X}$ and $\mathcal{Y}$. In practice, we choose a VST version of a reverb audio plug-in.

Starting with a corpus of non-reverberated audio, we synthesize a dataset,

$$\mathcal{D} = \{(\boldsymbol{g}_{\boldsymbol{\phi}}, \pi, \boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{g}_{\boldsymbol{\phi}} \in \mathcal{G},\, \pi \in [0,1],\, \boldsymbol{x} \in \mathcal{X},\, \boldsymbol{y} = \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x})\}\,. \quad (5)$$

There are a few actively developed command line tools for batch processing large-scale audio data through a VST plug-in, e.g., Mrs. Watson[1] and RenderMan[2]. However we found that offline batch processing tools such as these could fail to render audio properly under certain conditions. In particular, unexpected results were encountered with reverb plug-ins that render reverb tails extending beyond the duration of the input audio file.

We mitigated these issues by writing a batch processing server that generates audio data from an arbitrary VST plug-in. The server, which we provide as open source software[3], was built with Max by Cycling '74[4]. The application runs an HTTP server written in Node.js to process HTTP requests specifying a dry audio file and arbitrary VST plug-in parameter values. Each request triggers the application to processes the dry audio file through an instance of a VST plug-in and saves an output audio file after the plug-in's output gain settles to zero.

Our server is designed to process many files concurrently. For this work, the number of data files processed in parallel was limited to a hand-tuned number of reverb plug-in instances that could safely run with the available CPU overhead on our hardware. An area of future development is to detect audio dropouts due to CPU overload and automatically adjust the number of data files being rendered in parallel.

For each example of dry audio, we uniformly sample a reverberator $\boldsymbol{g}_{\boldsymbol{\phi}}$ from $\mathcal{G}$ and mixing ratio π in $[0, 1]$. We subsequently generated one or more associated examples of reverberated audio using the batch processing server. The specific dry data and set of reverberators $\mathcal{G}$, chosen to correspond to the codomain of interest, are discussed with respect to our experiment in Section 6.

5. MODEL

The model, shown in Figure 1, is a non-causal temporally adaptive neural network. Raw time-domain reverberant audio of arbitrary duration is transformed to the frequency domain using an STFT.[5] Frames are further grouped into subsequences of maximum length L frames. Frame groups are processed through a stack of bidirectional recurrent layers [8]. A *recurrent layer* is a first-order stateful function that is often applied to temporal sequences of inputs. The output at the current time step is a nonlinear function of an affine transformation of the layer input and the current state. The current state is typically the output of the previous step. A *bidirectional recurrent layer* additionally applies the nonlinear function, with different weights, to the sequence in reverse order.

The recurrent cell of each direction of each layer is a gated recurrent unit (GRU) [9]. A GRU includes a reset and an update gate. The reset and update gates are each a nonlinear function of an affine transformation of the input and previous state. The reset gate is used to create an intermediate state that "forgets" some portion of its current state. An intermediate output is produced as a nonlinear function of a different affine transformation of the input and the intermediate state. The update gate is used to mix a portion of the previous state with a complementary portion of the intermediate output. Hence a GRU may be trained to adaptively reset and update relevant portions of the state, based upon the current input and previous state. A bidirectional gated recurrent unit (BGRU) applies two GRUs, one to the input sequence (forward GRU), and another to the time-reversed input sequence (reverse GRU). The output of the reverse GRU is time-reversed and concatenated with the output of the forward GRU.

The final time step of the last recurrent layer is sliced and followed by a fully connected layer, which projects the last time step of the recurrent outputs to the required output dimensionality D, optionally followed by an activation function. We apply mean pooling over the inferences pertaining to groups of subsequences. For training, we use a subset of the dataset $\mathcal{D}^T \subset \mathcal{D}$. In the rest of this paper, we let $N = |\mathcal{D}^T|$, and any particular element in the training set is indicated by a superscript index.

We consider two versions of the model.

1. A regression model, which infers the reverb parameters $\boldsymbol{\phi} \in \Phi$, where $\Phi \subset [0,1]^D$.
2. A classification model, which infers the posterior probability $\boldsymbol{q}(\mathcal{G}|\boldsymbol{y})$, where $D = |\mathcal{G}|$.

In both versions, the model additionally infers π, the mixing coefficient. The objective function is

$$\mathcal{L} = \alpha\,\mathcal{L}^{\pi} + (1-\alpha)\,\mathcal{L}^{g}\,, \quad (6)$$

where $\mathcal{L}^{\pi}$ is the mean-squared error of the model's mixing ratio inference and α is a term that weights the relative influences of the losses. Each model and its loss $\mathcal{L}^{g}$ is described in more detail below.

5.1. Regression Model

The regression model directly infers reverb parameters $\hat{\boldsymbol{\phi}} \in \mathbb{R}^D$. In this version, there is no activation function applied to the fully connected layer, and the network can be described as follows.

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{y}) = [\hat{\boldsymbol{\phi}}, \hat{\pi}]\,. \quad (7)$$

The objective function $\mathcal{L}^{g}$ is the mean-squared error of ground truth and inferred reverb parameters, $\boldsymbol{\phi}$ and $\hat{\boldsymbol{\phi}}$.

5.2. Classification Model

The classification model infers one of D discrete reverberators, represented as a one-hot vector. Denote the following function $\boldsymbol{h}$, which maps a reverberator to a one-hot vector.

$$\boldsymbol{h} : \mathcal{G} \rightarrow \left\{ \boldsymbol{p} \in \{0,1\}^D \,\middle|\, \sum_{i=1}^{D} p_i = 1 \right\}\,. \quad (8)$$

[1] http://teragonaudio.com/MrsWatson

[2] https://github.com/fedden/RenderMan

[3] https://github.com/iZotope/max_vst_renderer

[4] https://cycling74.com/products/max

[5] We've informally tested various transformations of the STFT as input, including Mel-scale frequencies, log amplitudes, and cepstral coefficients. Although not shown in our results, we found that the STFT input consistently performed best for the reverb matching task.

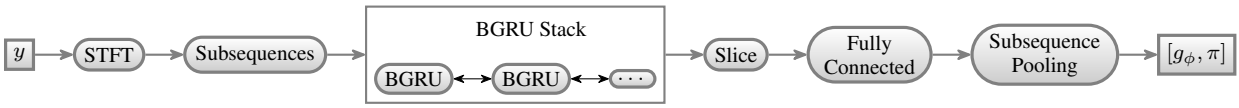

Figure 1: *Model architecture.*

Table 1: *List of PhoenixVerb parameters.*

Parameter		
Diffusion	Diffuser Type	Diffuser Size
Predelay	Early Attack	Early Time
Early Slope	Envelope Attack	Envelope Time
Envelope Slope	Reverb Size	Reverb Time
Xover Frequency	Low Mid Balance	Damp Frequency
Damping Factor	Early Level	Reverb Level
Width	Output Filter Type	Out Frequency

We denote the ground truth probability distribution of a particular reverberator as $\boldsymbol{p} = \boldsymbol{h}(\boldsymbol{g}_\phi)$. The first D elements of the fully connected layer are passed through a soft-max activation function, which we denote $\boldsymbol{q}$ so that the network can be described as follows.

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{y}) = [\boldsymbol{q}, \hat{\pi}]\,. \tag{9}$$

We use a cross-entropy loss function $\mathcal{L}^g$ of $\boldsymbol{q}$ relative to $\boldsymbol{p}$.

6. EXPERIMENTS

We chose monophonic reverberant dialogue as our codomain of interest. We used several speech corpora to build a dataset. (See Section 4 for a description of how the reverberant data was generated.) By far, the largest amount of data was culled from the VCTK speech corpus [10], which consists of 109 subjects reading newspaper texts, and consists of approximately 44 hours of audio. We randomly selected 86 subjects for training data and put the rest aside for validation and testing. The VCTK speech corpus consists largely of monotone non-emotional speaking, so we added a few other datasets of emotional speech to the training data. These included the Berlin Database of Emotional Speech [11] and the Surrey Audio-Visual Expressed Emotion Database [12].

We chose the PhoenixVerb VST audio plug-in by Exponential Audio, which has a broad set of tunable parameters for generating many types of natural-sounding reverb. The parameter list is provided in Table 1. We note, however, that any reverb generator satisfying one's matching requirements should suffice. For each dry audio file in our dataset, we chose at random a PhoenixVerb preset form the list of presets shown in Table 2. These presets, selected largely from the "ADR and Post" category of PhoenixVerb's presets, were chosen by an expert audio engineer for their appropriateness to the codomain of interest.

The model was trained and tested using reverberated monophonic audio sampled at 16 kHz. (High quality resampling was performed when necessary.) The STFT was computed using a window size of 512 samples with the same number of coefficients. The input to the model was the first 257 non-redundant magnitude coefficients of the STFT. The hop size was 128 samples. We used two stacked layers of BGRUs. The sub-sequence length L was set to 122, resulting in groups of subsequences approximately 1 sec in duration. The outputs of the model corresponded to either the 90 presets listed in Table 2 or the 21 parameters listed in Table 1 (classification model and regression model, described below). We used Adam [13] to optimize the model and trained it until there was no improvement in loss for 20 epochs on a held-out validation set.

Model performance was evaluated using MUSHRA (MUltiple Stimuli with Hidden Reference and Anchor) [14], a method designed to evaluate the quality of a lossy audio codec with respect to an uncompressed reference. For the reverb matching task, we adapt part of the MUSHRA protocol for the purpose of evaluating the relative quality of several model variants. It is important to point out that the goal of our task is to automatically add reverb to a "dry" sound so that it sounds the same as a reverberated reference. Because we are not evaluating compression codecs, our goal diverges slightly with MUSHRA. Nevertheless, MUSHRA is a helpful tool to determine quality of algorithms. Our experimental design is discussed below, along with any significant variations from the published MUSHRA protocol.

6.1. Design

Sixteen listening examples, or "trials", were provided in random order. The duration of each example ranged from 3.4 to 10.4 seconds. Half of the examples were "synthetic" and half were "natural" (described below). Each trial had a "visible" reference track, which is reverberated audio. For each trial, subjects were asked to evaluate the pairwise similarity of the visible reference with each of five stimuli, presented in random order. The stimuli are described below.

Hidden reference The same audio as the visible reference.

Human model A reverberated example created by an expert audio engineer, who used the PhoenixVerb plug-in in combination with a digital audio workstation to manually perform the reverb matching task. The engineer was allocated a maximum of 120 seconds, per trial, to perform manual reverb matching. (Therefore the time limit imposed on the engineer allowed at least an order of magnitude more time than the duration of each trial example.)

Random model A PhoenixVerb preset (from Table 2) and the value for mixing ratio were selected uniformly at random.

Regression model PhoenixVerb parameters (from Table 1 and mixing ratio predicted by a trained regression model.

Classification model PhoenixVerb preset (from Table 2) and mixing ratio predicted by a trained classification model.

The participants were asked to rate each example along a numerical continuum from 0 to 100. There was no time limit per trial. Participants were presented one unused example before the experiment began to get accustomed to the experimental environment. The experiment was conducted online using the open source webMUSHRA interface [15]. Participants were asked to conduct

the experiment in a quiet environment using high quality headphones, but no other specific controls were applied to the listening context.

6.1.1. Synthetic References (Trials 0–7)

These examples were drawn from the test partition of the VCTK corpus, in which adult men and women were recorded reading fragments of newspaper stories. The vocal deliveries in the corpus are generally not emotive, and the recording conditions are consistent. The "dry" recordings were reverberated using presets from PhoenixVerb. The presets were drawn randomly from a list selected by an expert audio engineer (Table 2) and the mixing ratio was drawn from a uniformly random distribution between 0 and 1. These examples are very similar to the ones that were used to train our models. However, none of the voice actors in this set had been used when training the model.

6.1.2. Natural References (Trials 8–15)

These are internally sourced field recordings that are "natural," in the sense that they were recorded by a production team, either on a sound stage, or at another naturally reverberant environment. In some cases, we used paired dry and reverberant audio recorded with different microphones located at different distances with respect to the subject speaking. In other cases (Trials 8, 13, and 14), we used unpaired reverberant recordings and dry versions were created using [16]. Since "natural" field recordings were "from the wild," there were no ground-truth reverb parameter settings associated with these recordings.

6.2. Differences from MUSHRA

This section describes how our MUSHRA test deviated from the established protocol, which was originally designed for evaluating the quality of lossy audio compression algorithms. We note that such deviations from the protocol are widely used in the literature when subjectively evaluating generic audio algorithms.

6.2.1. Hidden Reference

In the established MUSHRA protocol, participants are instructed that the reference is hidden among the comparison clips. Given this knowledge, it is therefore expected that the participants rate at least one example with a score of 100, per trial. This provides the experimenters a mechanism for rejecting participants who provide the hidden reference a low rating. Due to an oversight, we did not provide this information to our participants. As a result some participants rated the hidden reference lower than 100. However, we applied statistical analysis to each individual's ratings and determined that no participant rated other models higher than the hidden reference, even when the hidden reference was given, on average, a rating lower than 100.

6.2.2. No Anchors

We did not provide any hidden anchors in the comparison clips. Because we were not evaluating audio quality, it was difficult to design a true "anchor." An anchor should degrade the reference by a known quantity, but there are too many perceptually important and confounding dimensions in reverberation for us to design a true anchor. For instance, we could create an anchor that reduces

Table 2: *List of PhoenixVerb presets used in training the classification model and testing the classification and regression models.*

Preset	
Tight Snare	Tiny Ancient Plate
Tiny Ancient Plate 2	Tiny Vocal Plate
Dessert Plate	In the Nursery
Sml Clean Chamb	Sml Mid Chamb 2
Sml Mid Chamb 3	Chicken Chamber
Skinny Cow Chamb	Recital Hall
Recital Hall 2	Ruby's Cube 2
Live Room	Live Room 2.
Live Room 3	Live Room 4
Live Room 5	Live Room 6.
Live Room 7	Live Room 8
Live Room 9	Live Room 10
Muted Live Room	Muted Live Room 2
Cookin Kitchen	Cookin Kitchen Too
Balance Room	Airy Room
Airy Room 2	Room With a View
Room With a View Too	Live Vocal Booth
Smooth Vocal Booth	Wally Room
Wally Room 2	Wally Room 3
Polly's Wall Room	Kick Boom Room
Kick Boom Room 2	Empty Office
Empty Office 2	Lge Live Room
Lge Live Room 2	Carpet Crypt
Carpet Crypt 2	Carpet Crypt 3
Carpet Crypt 4	Carpet Crypt 5
The Bear Gets You	Ski Slope
Snowbird Diamond	Little Blue Car
Little Blue Car 2	Little Blue Car 3
Little Blue Car 4	Little Blue Car 5
50's Hot Rod	60's Hot Rod
Front Yard	Front Yard 2
Front Yard 3	Front Yard 4
Front Yard 5	Back Yard
Back Yard 2	Back Yard 3
Back Yard 4	City Street
City Street 2	City Street 3
City Street 4	City Street 5
Afterthump	Boxed In
Inverse	Inverse 2
BongosMediumRoom	BongosLargeRoom
CongasSmallRoom	CongasMediumRoom
TamboraMediumRoom	TamboraLargeRoom
GuiraRoom	StereoKick
KickAir	Hip-HopSnare
Hip-HopSnare 2	Bass Thumper

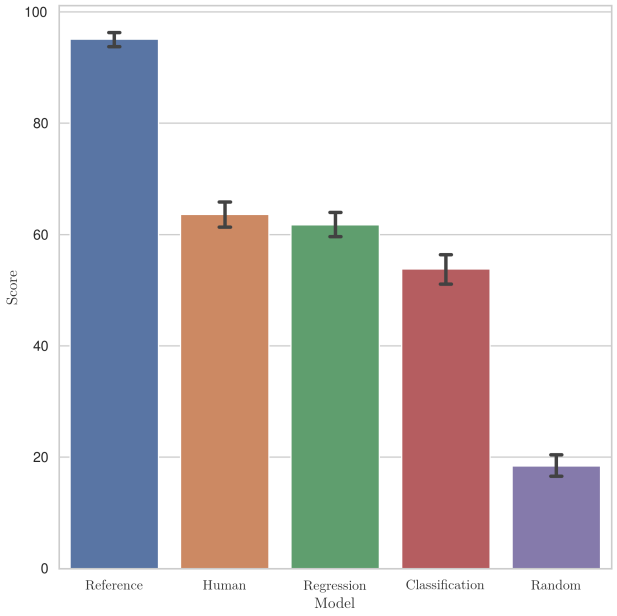

Figure 2: *Average ratings and* 95 % *confidence intervals of models. (All trials.)*

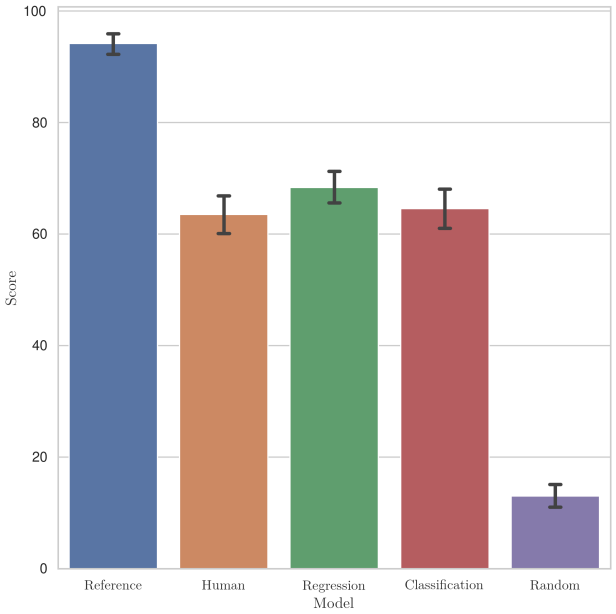

Figure 3: *Average ratings and* 95 % *confidence intervals of models. (Synthetic trials.)*

or increases the mixing ratio relative to the reference. However, mixing ratio might be easily confused with reverberation time. If the reference has a short reverberation time, the anchor would be less meaningful and easier to confuse. Instead, the "Random Model" provides the closest stimulus to a hidden anchor. The random model provides us a mechanism for evaluating a baseline performance and it is expected, on average, to be evaluated much worse than the other stimuli.

6.3. Results

We had 28 adult participants in the experiment. All participants are employees at an audio technology company where the average employee may be considered a highly experienced listener. We validated participants and trials by checking whether any model was given a significantly better score than the reference.

Graphical results are depicted for the full set of trials, as well as "synthetic" and "natural" partitions. The bar plots show the average ratings for each stimulus. Bar plot whiskers depict the 95 % confidence interval, as computed using the bootstrap method with 1000 iterations. Overall, and as shown in Figure 2, the hidden reference was provided the highest score and the Random Model was provided the lowest score. When examining the full set of trials, we determine that the Regression Model performed about as well as a human expert. This is apparent by observing the corresponding overlapping confidence intervals in 2. We also note that the regression model apparently performs better than the classification model, with non-overlapping confidence intervals.

When examining the partition of trials that used only "synthetic" examples (Figure 3), the regression model and classification model, as evaluated by our experiment participants, was indistinguishable in performance from human reverb matching. This observation is based on the overlapping confidence intervals for all three models.

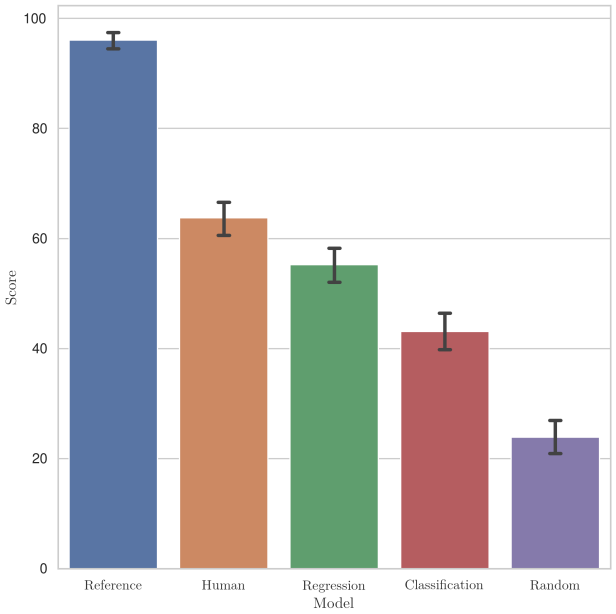

Figure 4: *Average ratings and* 95 % *confidence intervals of models. (Natural trials.)*

When examining results corresponding to "natural" examples, shown in Figure 4, human subjects showed a clear order of preferences: Hidden Reference > Human Expert > Regression Model > Classification Model > Random baseline. All preferences were determined to be statistically significant on the "natural" data. Even though the "Regression" model does not perform as well as a human expert engineer, we note that it performs much better than the baseline, and that the model takes orders of magnitude less time than a human to provide its inferences.

7. CONCLUSION

We presented a model for the reverb matching task, in which one would like to automatically transfer the reverberant characteristics of a reference audio track to a "dry" audio track. Notably, the underlying source material of the "wet" and "dry" tracks may not be the same. Such a model helps an audio engineer to avoid an exhaustive manual search over the parameter space of an artificial reverb generator.

We presented a subjective study where the codomain of interest is reverberant speech dialogue, of the type commonly used in automatic dialogue replacement for movies and TV. We provided results corresponding to test data partitioned into "synthetic" and "natural" examples. Considering the results of the full test set and its partitions, we determine that the regression model does a good job at the reverb matching task. It is as good as a human expert, especially when the testing data is similar to the data used to train the model. It performs a little worse than an expert human on more natural data, but far better than a random baseline. We expect that we may improve the model's performance for in-the-field recordings if we provide more varied data during training. As mentioned above, the training data was overwhelmingly non-emotive, which doesn't align well to the types of recordings that would typically occur for tasks like ADR. We leave training on a more heterogeneous dataset to future work.

8. ACKNOWLEDGMENTS

Many thanks to Kurt Werner for his invaluable insights while preparing this paper. We also thank the DAFx reviewers for their comments.

9. REFERENCES

[1] Jens Blauert, *Spatial hearing: The psychophysics of human sound localization*, MIT press, Cambridge, MA, 1997.

[2] Katsutoshi Itoyama and Hiroshi G. Okuno, "Parameter estimation of virtual musical instrument synthesizers," in *Joint Proceedings of the 40th International Computer Music Conference (ICMC) and 11th Sound & Music Computing Conference (SMC)*, Athens, Greece, Sept. 2014, pp. 1426–1431.

[3] Matthew J. 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*, vol. 2, no. 2, pp. 150–159, Apr. 2018.

[4] O. Barkan, D. Tsiris, O. Katz, and N. Koenigstein, "InverSynth: Deep estimation of synthesizer parameter configurations from audio signals," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 12, pp. 2385–2396, 2019.

[5] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla–Romeu-Santos, "Flow synthesizer: Universal audio synthesizer control with normalizing flows," *Applied Sciences*, vol. 10, no. 1, pp. 302, 2020.

[6] Nils Peters, Jaeyoung Choi, and Howard Lei, "Matching artificial reverb settings to unknown room recordings: A recommendation system for reverb plugins," in *Proceedings of the Audio Engineering Society Convention 133*, San Francisco, USA, Oct. 2012.

[7] Jiaqi Su, Zeyu Jin, and Adam Finkelstein, "Acoustic matching by embedding impulse responses," in *Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, May 2020, pp. 426–430.

[8] Mike Schuster and Kuldip K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, Doha, Qatar, Oct. 2014, pp. 1724–1734.

[10] Christophe Veaux, Junichi Yamagishi, and Kirsten MacDonald, "CSTR VCTK corpus: English multi-speaker corpus for CSTR voice cloning toolkit," 2012, Version 0.8.0.

[11] Felix Burkhardt, Astrid Paeschke, Miriam Rolfes, Walter F. Sendlmeier, and Benjamin Weiss, "A database of German emotional speech," in *Proceedings of the 9th European Conference on Speech Communication and Technology (INTERSPEECH-2005)*, Lisbon, Portugal, Sept. 2005, pp. 1517–1520.

[12] Sanaul Haq, Philip J. B. Jackson, and James D. Edge, "Audio-visual feature selection and reduction for emotion classification," in *Proceedings of the International Conference on Auditory-Visual Speech Processing (AVSP'08)*, Moreton Island, Queensland, Australia, Sept. 2008, pp. 185–190.

[13] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, USA, May 2015.

[14] *ITU-R recommendation BS. 1534-3: Method for the subjective assessment of intermediate quality level of audio*, International Telecommunication Union, Oct. 2015.

[15] Michael Schoeffler, Sarah Bartoschek, Fabian-Robert Stöter, Marlene Roess, Susanne Westphal, Bernd Edler, and Jürgen Herre, "webMUSHRA—A comprehensive framework for web-based listening tests," *Journal of Open Research Software*, vol. 6, no. 1, 2018.

[16] Shahan Nercessian and Alexey Lukin, "Speech dereverberation using recurrent neural networks," in *Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19)*, Birmingham, United Kingdom, Sept. 2019.