Appendix A Method Details

A.1 Differentiable impact force profile

ALGORITHM 1: Differentiable Impact Force Profile Computation

Input: Normalized magnitude estimate time series vector $\mathbf{m_{in}} \in \mathbb{R}^{T_{in}}$, normalized contact time scale estimate time series vector $\boldsymbol{\tau_{in}} \in \mathbb{R}^{T_{in}}$, final number of time samples T_{out}, and expected number of peaks M

Output: A predicted impulse profile over time, with T_{out} samples sampled at rate f_s Hz

$\mathbf{m_{out}} \leftarrow MagScaleFn(\mathbf{m_{in}})$;
$\boldsymbol{\tau_{out}} \leftarrow TauScaleFn(\boldsymbol{\tau_{in}})$;
$\mathbf{m_{out}} \leftarrow HannWindowResample(\mathbf{m_{out}}, T_{out})$;
$\boldsymbol{\tau_{out}} \leftarrow HannWindowResample(\boldsymbol{\tau_{out}}, T_{out})$;
$w \leftarrow \lfloor T_{out}/M \rfloor$;
$\mathbf{Ind} \leftarrow ArgmaxPool(\mathbf{m_{out}}, Kernel = w, Stride = w)$;
$\mathbf{t} \leftarrow ([\mathbf{Ind}-\delta] \circ \mathbf{m_{out}}[\mathbf{Ind}-\delta] + \mathbf{Ind} \circ \mathbf{m_{out}}[\mathbf{Ind}] + [\mathbf{Ind}+\delta] \circ \mathbf{m_{out}}[\mathbf{Ind}+\delta])/3f_s$;
return $F([0, 1/f_s, ..., T_{out}/f_s]; \mathbf{m_{out}}[Ind], \mathbf{t}, \boldsymbol{\tau_{out}}[Ind])$

Here, we describe the details of the algorithm we use in Section 3.2 for generating a physics-based series of impact impulses at audio sample rate over time, using as inputs two coarsely temporally discretized and normalized time series. These inputs are the outputs of an encoder in the case of our End-to-End experiment, or learnable time series parameter vectors in the case of our Analysis-by-Synthesis experiment. We will describe how we convert these normalized inputs into inputs for Equation 3 in a fully differentiable manner.

To model Equation 3, we must choose an integer value for M in the equation, corresponding to the number of impacts occurring during the interval. To do this, we fix M as a hyperparameter based on a reasonably tight upper bound of our expectation of the overall frequency of impacts, with the impacts spaced somewhat regularly across the interval. Because M is intentionally chosen to be an *overestimate* of the number of impacts occurring during most time intervals, our algorithm for Equation 3 will frequently instantiate more impacts than actually occurred in an interval. Impacts that have been placed in subintervals where no impact occurred should ideally approach magnitude $m_i = 0$.

Our differentiable impact profile algorithm, with pseudocode shown in Algorithm 1, takes a coarsely discretized time series of network outputs or variables corresponding to continuous normalized input parameters for estimating magnitude and τ values for M different impact impulses, each modeled by Equation 2. We pass the input parameters for the magnitude and the τ values through respective scaling functions, $MagScaleFn$ and $TauScaleFn$ in Algorithm 1. $MagScaleFn$ and $TauScaleFn$ are both exponential sigmoid functions. Each vector has T_{in} temporal elements, but with T_{out} being the number of samples in the final output audio, at an audio sample rate f_s Hz, the input time series are coarsely discretized such that $T_{in} << T_{out}$. Given an input vector of a time series of magnitudes $\mathbf{m_{in}} = [m_{in,0}, ..., m_{in,T_in}]$ with T_{in} temporal elements, $MagScaleFn$ bounds the magnitude estimates from 0 to 2:

$$MagScaleFn_i(m_{in,i}) = u_{mag} * sigmoid^{\lambda_{mag}}(m_{in,i} + b_{mag}) + l_{mag} \tag{4}$$

$$\mathbf{m_{out}} = \mathbf{MagScaleFn}(\mathbf{m_{in}}), \tag{5}$$

where b_{mag} is a hyperparameter bias for controlling initialization conditions, hyperparameters u_{mag} and l_{mag} control the upper and lower bounds respectively of the scaling function, and $\mathbf{MagScaleFn}$ stacks the output of Equation 4 into a vector. We set hyperparameter l_{mag} to a very small constant (10^{-7}), ensuring the scaled magnitude never fully collapses to 0 during optimization. The exponent λ_{mag} on the exponential sigmoid function is a hyperparameter controlling the slope of the function about $x = 0$.

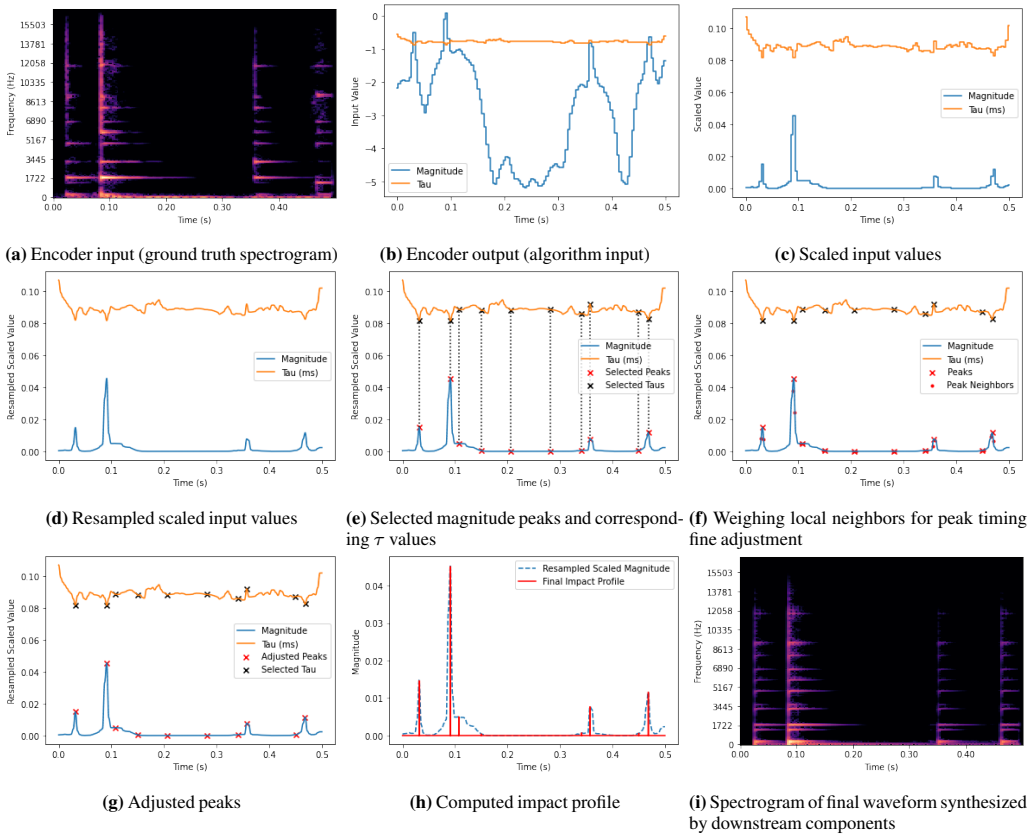
Given a similar input vector for the τ values, $\boldsymbol{\tau_{in}} = [\tau_{in,0}, ..., \tau_{in,T_in}]$, we produce the $\boldsymbol{\tau_{out}}$ scaled vector as follows:

$$TauScaleFn_i(\tau_{in,i}) = u_\tau * sigmoid^{\lambda_\tau}(\tau_{in,i} + b_\tau) + l_\tau / f_s \tag{6}$$

$$\boldsymbol{\tau_{out}} = \mathbf{TauScaleFn}(\boldsymbol{\tau_{in}}) \tag{7}$$

where f_s is the audio sample rate, b_τ is a bias learned for each material, hyperparameter λ_τ controls the slope on the sigmoid, hyperparameters u_τ and b_τ control the respective upper and lower bounds

of the scaling function, and **TauScaleFn** stacks the output of Equation 6 into a vector. We have the learnable bias b_τ because each material is likely to have a different distribution of τ, depending on softness of the object's material or contact surface. Harder materials should have smaller τ values, and softer materials should have larger τ values. The upper bound u_τ on this scaling function ensure that contact time scales τ remain within physically reasonable ranges (below 3ms for our experiments), and the lower bound l_τ ensures that the resulting impulse force profile's curve shape does not degrade too much as the width of the curve approaches the sample rate.



(a) Encoder input (ground truth spectrogram) **(b)** Encoder output (algorithm input) **(c)** Scaled input values

(d) Resampled scaled input values **(e)** Selected magnitude peaks and corresponding τ values **(f)** Weighing local neighbors for peak timing fine adjustment

(g) Adjusted peaks **(h)** Computed impact profile **(i)** Spectrogram of final waveform synthesized by downstream components

Figure 5: Steps of the Differentiable Impact Force Profile algorithm

We next resample both time series to the resolution of the audio sample rate (*e.g.*, 44100 Hz) using Hann windows. We select peaks in the resampled magnitude time series using an $ArgmaxPool$ operation, with window sizes based on the length of the time interval divided by M. Since selecting the peaks with $ArgmaxPool$ can lose any gradient with respect to the timing of the peaks, to recapture this gradient, we next adjust the selected peak timings by taking a magnitude-weighted average of the magnitude at the peak timings and a preceding and a succeeding neighboring magnitude of each selected peak. Finally, we select the magnitude and τ values from the time instant of each peak in the series. Through these differentiable steps, we now have the magnitude vector $\mathbf{m}$, the timing vector $\mathbf{t}$, and contact time scale vector $\boldsymbol{\tau}$, providing us with all the input parameters for Equation 3, producing a fully differentiable impact impulse profile with M peaks over the time interval. For an example walkthrough of each of these steps, see Figure 5.

A.2 Modal impulse response

For our modal impulse response, we will explain how we take normalized input parameters and convert them into input parameters of our modal impulse response equation from Section 3.3:

$$IR_o(t;\mathbf{f},\mathbf{g_o},\mathbf{d_o}) = \mathbf{g_o}^T[\exp(-\mathbf{d_o}t)\circ\sin(2\pi\mathbf{f}t)], \tag{8}$$

where $\mathbf{f} = [f_0,...,f_K]$, $\mathbf{g_o} = [g_0,...,g_K]$, $\mathbf{d_o} = [d_0,...,d_K]$ and $\circ$ is the Hadamard product.

We begin with vectors $\mathbf{a_{in}} = [a_0, ..., a_K]$, $\mathbf{b_{in}} = [b_0, ..., b_K]$, and $\mathbf{f_{in}} = [f_{in,0}, ..., f_{in,K}]$ which are normalized (unscaled) input vectors corresponding to the gains $\mathbf{g_o}$, dampings $\mathbf{d_o}$, and frequencies $\mathbf{f}$, respectively. For scaling input vector $\mathbf{a_{in}}$ into $\mathbf{g_o}$, we use a softmax scaling function :

$$ModalGainScaleFn_i(a_i) = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \tag{9}$$

$$\mathbf{g_o} = \mathbf{ModalGainScaleFn}(\mathbf{a_{in}}) \tag{10}$$

where $\mathbf{ModalGainScaleFn}$ stacks the output of Equation 9 into a vector. This softmax formulation maintains a normalized distribution of relative gains of each frequency component in the modal impulse response, ensuring that the overall magnitude of the impulse response is not a free parameter during optimization, but rather is completely deferred to the impact impulse force profile.

To convert the input vector $\mathbf{b_{in}}$ into the damping vector $\mathbf{d_o}$, we use an exponential sigmoid function:

$$DampingScaleFn_i(b_i) = u_{damp} * sigmoid^{\lambda_{damp}}(b_i + b_{damping}) + l_{damp} \tag{11}$$

$$\mathbf{d_o} = \mathbf{DampingScaleFn}(\mathbf{b_{in}}) \tag{12}$$

where $b_{damping}$ is a hyperparameter bias, hyperparameter λ_{damp} controls the slope of the sigmoid, u_{damp} and l_{damp} are hyperparemeters for the respective upper and lower bounds of the function, and $\mathbf{DampingScaleFn}$ stacks the output of Equation 11 into a vector. We set the value of the upper bound u_{damp} in Equation 11 to 10000 to ensure that the damping values do not cause the contribution of their corresponding mode to disappear from the final impulse response at the audio sample rate.

Finally, for scaling input vector $\mathbf{f_{in}}$ into frequencies vector $\mathbf{f}$, we use a scaling function from the authors of [24], where input parameters are used to fine-tune a somewhat even distribution of frequencies over a range of frequencies (*e.g.* the range of human audible frequencies from ∼20Hz - 20kKhz), with respect to a Mel scale for human audio perception. First, we will define some equations important to the Mel scale [41]:

$$MelToHz(x) = 700 * (10.0^{(x/2595.0)} - 1.0) \tag{13}$$

$$HzToMel(x) = 2595 * \log_{10}(1.0 + x/700.0), \tag{14}$$

And the equivalent rectangular bandwidth (ERB) equation attempts to approximate theoretical frequency bandwidth filters in human hearing [42]:

$$HzToERB(x) = 0.108x + 24.7, \tag{15}$$

Now using these equations, we scale $\mathbf{f_{in}}$ into $\mathbf{f}$ as so:

$$CenterFreq(i) = (1 - \frac{i}{K-1}) * HzToMel(f_{min}) + \frac{i}{K-1} * HzToMel(f_{max}) \tag{16}$$

$$FrequencyScaleFn_i(f_{in,i}) = CenterFreq(i) + \tanh(f_{in,i}) * HzToERB(CenterFreq(i)) \tag{17}$$

$$\mathbf{f} = \mathbf{FrequencyScaleFn}(\mathbf{f_{in}}) \tag{18}$$

where f_{min} and f_{max} are hyperparameters corresponding to the lower and upper bound of the frequency range, respectively. $\mathbf{FrequencyScaleFn}$ stacks the output of Equation 18 into a vector.

A.3 Reverb

Here we describe how we scale normalized input parameter vectors into parameters for our room impulse response described in Section 3.4, which we use to model reverberation effects,

$$IR_e(t_{IR}; \mathbf{g_e}, \mathbf{d_e}) = \mathbf{g_e}^T [\exp(-\mathbf{d_e} t) \circ filt(\mathcal{N}(t), B_e)], \tag{19}$$

where $\mathbf{g_e}$ and $\mathbf{d_e}$ are the gains and damping factors per frequency band, respectively, and have each have dimension B_e.

Given input vector $\mathbf{g_{in}} = [g_{in,0}, ..., g_{in,B_e}]$, we produce $\mathbf{g_e} = [g_{e,0}, ..., g_{e,B_e}]$ using a similar scaling function as that from Equation 5:

$$ReverbGainScaleFn_i(g_{in,i}) = 2.0 * sigmoid^{\log(10)}(m_{in,i} + b_{reverb_gain}) + 10^{-7} \tag{20}$$

$$\mathbf{g_e} = \mathbf{ReverbGainScaleFn}(\mathbf{g_{in}}), \tag{21}$$

where b_{reverb_gain} is a hyperparameter bias, and $\mathbf{ReverbGainScaleFn}$ stacks the output of Equation 20 into a vector.

For the dampings, we scale input vector $\mathbf{d_{in}} = [d_{in,0},...,d_{in,B_e}]$ to produce $\mathbf{d_e} = [d_{e,0},...,d_{e,B_e}]$.

$$ReverbDampingScaleFn_i(d_{in,i}) = 2.0 + \exp(d_{in,i} + b_{decay}) \tag{22}$$

$$\mathbf{d_e} = \mathbf{ReverbDampingScaleFn}(d_{in}) \tag{23}$$

where b_{decay} is a hyperparameter bias, and $\mathbf{ReverbDampingScaleFn}$ stacks the output of Equation 22 into a vector.

With these differentiable scaling functions, we convert input parameter vectors $\mathbf{g_{in}}$ and $\mathbf{d_{in}}$ into parameter vectors $\mathbf{g_e}$ and $\mathbf{d_e}$, respectively, to be used as the inputs for Equation 19.

Appendix B Analysis by Synthesis Experiment Details

Here we describe some further details about the data collection and methods used for our experiment in Section 4.1, then show some additional results.

Figure 6: Physical setup for Analysis by Synthesis dataset collection. A microphone at the top left records audio from the suspended ceramic mug being struck by the impact hammer on the right.

Dataset To collect our dataset, we first suspended each of the four everyday objects shown at the bottom right of Figure 4 on a string to minimize the effects of contact dampings on each object's vibrations. We then struck each of them with a PCB 086C01 impact hammer with a force transducer and a hard nylon tip. We took synchronized recordings of audio with a PCB 378A06 microphone and this impact hammer while striking the objects repeatedly in roughly the same place. We adjusted gains on the microphone to maximize volume but prevent clipping for each object. We collected at least 60 seconds of data per object, then reviewed each object's data to find a 3 second clip for each object where the audio did not clip, and the impact forces were frequent and varied enough to be interesting to try to capture. Our data collection setup is shown in Figure 6.

Optimization Setup Our optimization used all the green elements in Figure 1, with the nuance that the inputs to the *Impact Force Profile Generator*, instead of coming from the encoder, were two variable vectors for the two normalized time series input parameters $\mathbf{m_{in}}$ and $\boldsymbol{\tau_{in}}$ required by the scaling functions of the generator (see Equations 5 and 7 in Appendix A.1). For this experiment, in order to mitigate the risk of falling into local minima early in the optimization, we also added zero-mean Gaussian noise to the magnitude variable vector. We controlled the standard deviation of the noise with a learnable variable vector representing a time series of standard deviations, with the same temporal resolution as the magnitude time series variable vector and all elements initialized to 0.1. We randomly initialized all other parameters including these two variable vectors, then optimized our loss function comparing the output of our model with the ground truth original audio with respect to all parameters, using the Adam optimizer [29]. To be clear, we did not provide our model with supervision for the impact force profile.

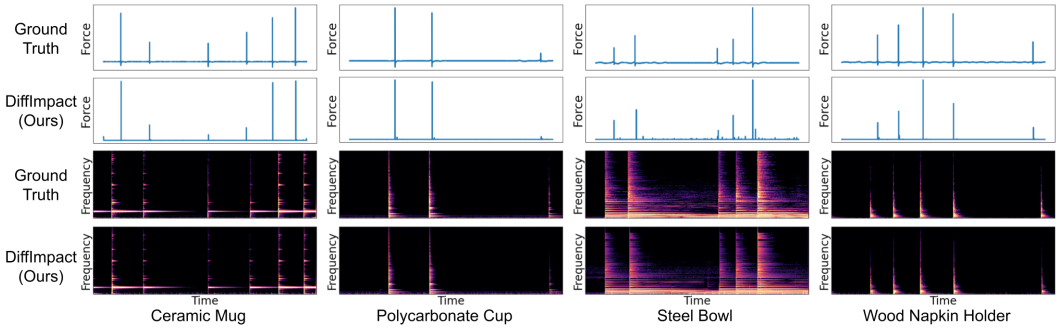


Figure 7: Comparing our model's estimates to ground truth time series of normalized contact forces and spectrograms, for our analysis by synthesis task on short recordings of each everyday object.

Full Results The results of using our model to fit parameters to each of the four objects are shown in Figure 7, and we include examples in our Supplementary Video for qualitative comparison. As we can see in Figure 7, each spectrogram demonstrates that the audio our model is able to produce for each object is very similar to its ground truth recorded audio. Furthermore, for each of the objects, our model fits an impact force profile which will most effectively produce audio approximating the ground truth recording, and as we can see, the timings and relative magnitudes of each of the impact impulses that our model has derived during the optimization are remarkably similar to the ground truth hammer force transducer recordings. This validates our hypothesis that our model is able to both generate realistic impact audio and extract physically interpretable parameters along the way, through this unsupervised analysis-by-synthesis process.

Appendix C End-to-End Learning Experiment Details

C.1 ASMR Bakery Dataset

(a) Wooden box **(b)** Glass bowl **(c)** Silicone pad

(d) Steel bar **(e)** Ceramic plate

Figure 8: Objects selected for our dataset from the ASMR Bakery YouTube channel

In searching for in-the-wild audio recordings of impacts, we discovered the ASMR Bakery YouTube Channel [1]. At time of writing, the channel features hundreds of YouTube videos of the creator

manipulating different everyday objects without speaking, providing an "unpolluted" dataset of manipulation sounds. We narrowed our search to videos with titles including "tapping", and used the annotations on each video to find interesting objects to select for our dataset. For the sake of our experiments, we assumed that each of the clips that were labeled as being of "tapping," consisted of only tapping sounds, *i.e.* were all real world impact sounds. However, in some of our clips, there were multiple occasions that the creator's in-hand manipulations of the objects inadvertently caused rubbing sounds in the course of tapping. These rubbing sounds were unfortunately unmodeled in our framework, causing our model to try to fit impact sounds to them. This demonstrates the importance of future work to expand our model with models for non-impact contacts such as rubbing and scratching. The creator used only fingertips and fingernails to tap each object in the clips we selected. We assumed that the fingernails and fingertips were small and damped enough that their modal vibrations' contribution to the sound was negligible, and therefore did not include their impulse responses as a contribution to the final sound in our framework's formulation. We only assumed that the fingertips and fingernails could cause acceleration sounds when they struck the object.

C.2 Our Model and Baselines

The high-level structure of our model used for this experiment is diagrammed in Figure 9. We first convert the spectrogram to Mel-frequency cepstrum coefficient (MFCC) features, which we fed into an encoder almost identical to that of [15]. The encoder consisted of a Gated Recurrent Unit (GRU), followed by a 1D temporal convolution with leaky ReLU activation, four fully connected layers with a leaky ReLU activation, followed by a GRU, then three more fully connected layers with leaky ReLU activation. Our autoencoder structure diverges from [15] after the encoder. We fed the outputs of this encoder directly to our model as the normalized input parameter vectors to our generators which we had combined in a directed acyclic graph (DAG) as summarized in Figure 1. We will refer to this section of our model as the *Synthesizing DAG* (see Figure 10), and compare it to two of our baselines. We passed the time series of outputs from the encoder as input to the Impact Force Profile generator, and used randomly initialized parameter variables as inputs for the other generators. To ensure that our model can effectively capture sharp impacts with short time scales without being too constrained by the lower bound of the scaling function in Equation 6, each of our generators produces its output at twice the final audio sample rate. Therefore, our final operation in our Synthesizing DAG performs a linear resampling to take our model's output back to final audio sample rate.

Both the DDSP Serra and DDSP Sin baseline models used the exact same encoder structure as our model but only vary the structure of the Synthesizing DAG by using pre-existing generators. This allows us to isolate the effects of our contributions. The DDSP Serra model used the same Synthesizing DAG as [15] (Figure 11), based on the Serra-Smith model[30], using encoder outputs to control both a harmonic oscillator and time-varying filtered noise, summed to make the final output. Since the modal vibrations of general objects are not necessarily harmonic like the musical instruments for which the Serra-Smith model was designed, we constructed the Synthesizing DAG for the DDSP Sin by replacing the harmonic oscillator of DDSP Serra with an unrestricted time-varying sinusoidal oscillator (Figure 12). These two models were trained with the same loss as our model, with the addition of spectrograms of lower window sizes (128 and 64) to mirror the loss used in the work which introduced these models [15, 24] (details in Appendix C.2).

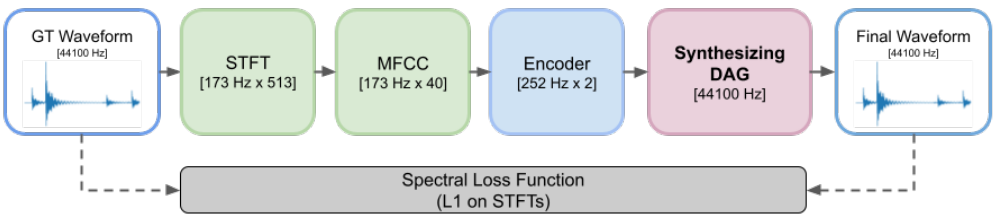


Figure 9: High-level Autoencoder Diagram

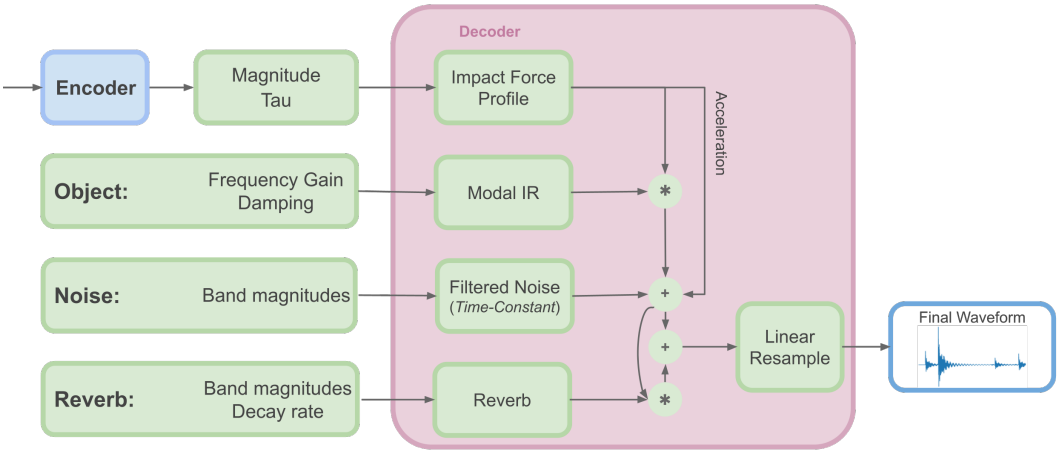


Figure 10: Proposed Synthesizing DAG.

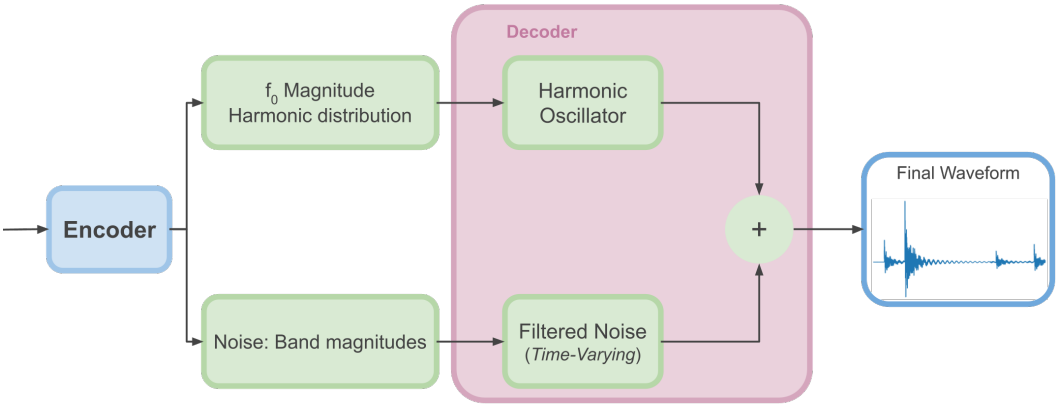


Figure 11: "DDSP Serra" baseline model Synthesizing DAG.

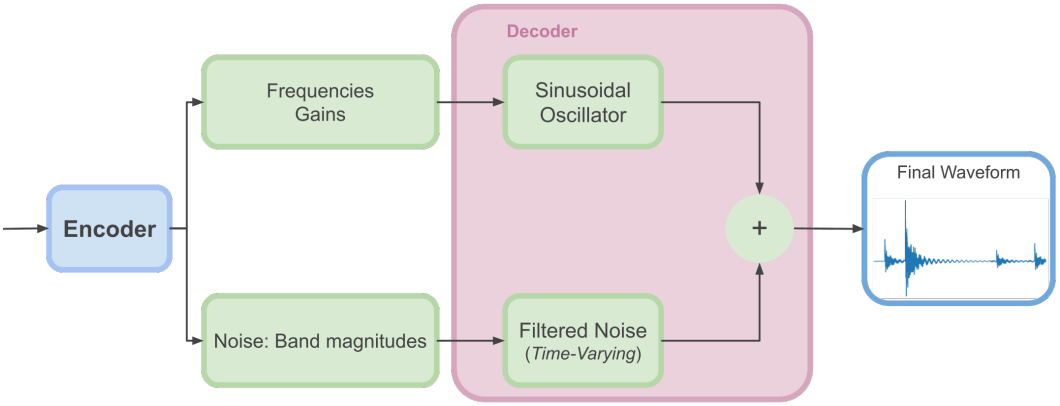


Figure 12: "DDSP Sinusoidal" baseline model Synthesizing DAG.

C.3 Human Study

For each of our published Human Intelligence Task (HIT) tasks on Amazon Mechanical Turk, we presented the workers with 12 questions containing randomly chosen tapping ASMR videos from our test set of each of the 5 different objects: wooden box, glass bowl, silicone pad, steel bar, and ceramic plate. Each of these videos had been redubbed with either our model's output or the output of

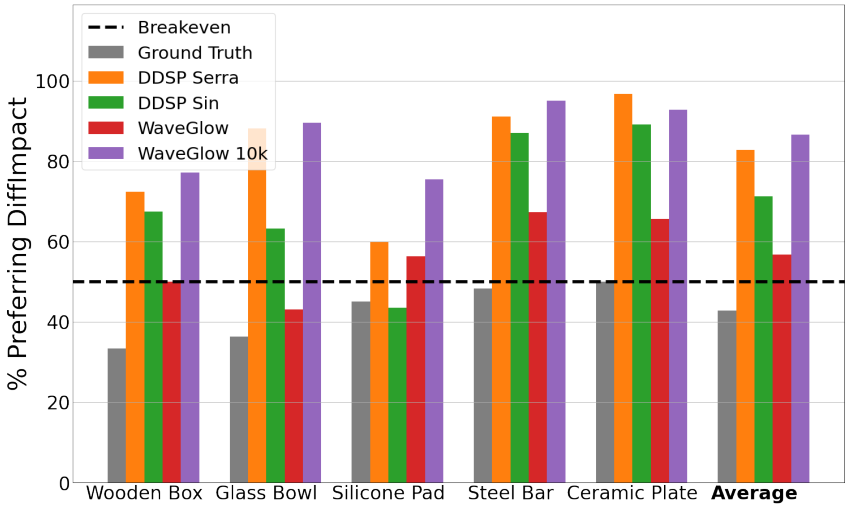


Figure 13: Comparing human study ratings of realism of different ASMR objects' test set outputs from each model to the output of the undertrained WaveGlow model ("WaveGlow 10k").

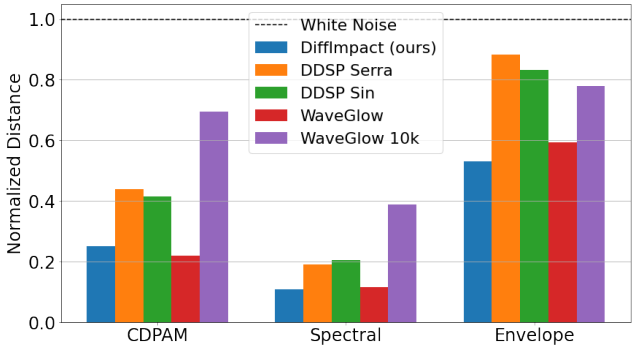


Figure 14: Comparing average computational audio distance metrics of test set output from different models to the output of the undertrained WaveGlow model ("WaveGlow 10k").

a baseline model, where the input to each model was the spectrogram computed from the ground truth original audio, or we left the video with its ground truth audio. In each question, we presented the participants with two videos, one from each treatment, and asked them to choose the one that better matches the sound they would expect from the object in the video. We conducted 30 HITs per model resulting in a total of 120 HITs. To detect fraudulent responses from users who were not following the instructions, we randomly selected 2 out of 12 questions in each HIT as checkpoints where a video redubbed with white noise was presented as one of the options. We then removed the responses of the users who chose white noise in those questions from our study to produce the results we show. 15 out of 120 HITs were determined as fraudulent based on this metric. Our final results included answers from the remaining 105 HITs containing a total of 1050 questions across all models.

C.4 Effects of Training Time on WaveGlow Performance

In Section 4.2, we claim that our model is able to learn to generate audio which is realistic enough to be narrowly rated as more realistic on average than audio from the WaveGlow model [16], as shown in Figure 3. However, WaveGlow requires much more computing resources to train, requiring about 300 hours to train a single object's model on a desktop computer with an Nvidia Quadro P5000 GPU, versus our model and other baselines requiring about 1 hour. To demonstrate the necessity of training the WaveGlow model for this long, here we show the results of our experiments from training each WaveGlow model for only 10,000 steps, requiring about 30 hours on our single GPU desktop, whereas the results we have presented as "WaveGlow" in all figures are from models which have been trained for 70,000 steps for each object, requiring about 300 hours on our desktop. Results are shown in Figures 13 and 14.

Appendix D Source Separation Experiment Details

In this experiment, the only supervision used during the optimization were the sound clips from each pairwise impact, and the labels of which object and tool are present in each clip. These labels can be considered as weak supervision.

ALGORITHM 2: Modal Impact Sound Source Separation Optimization

Input: Batch of recorded impact waveforms $\mathbf{W} \in \mathbb{R}^{B \times T}$, weak labels of N_o distinct object IDs present in each waveform $\mathbf{L_o} \in \mathbb{W}^B, N_o \leq B$, and weak labels of N_t distinct tool IDs present in each waveform $\mathbf{L_t} \in \mathbb{W}^B, N_t \leq B$

Output: Object modal parameters $\mathbf{M_o} \in \mathbb{R}^{N_o \times 3F_o}$, tool modal parameters $\mathbf{M_t} \in \mathbb{R}^{N_t \times 3F_t}$, tool reverb parameters $\mathbf{R_t} \in \mathbb{R}^{B \times 3F_R}$, contact force parameters $\mathbf{C_f} \in \mathbb{R}^{3B}$, acceleration sound scalars $\mathbf{A} \in \mathbb{R}^B$, environment noise parameters $\mathbf{N} \in \mathbb{R}^{F_N}$, and noise scalars $\mathbf{N_s} \in \mathbb{R}^B$

$\mathbf{M_o}, \mathbf{M_t}, \mathbf{R_t}, \mathbf{C}, \mathbf{N}, \mathbf{N_s} \leftarrow InitParameters()$;

for $i \leftarrow 1$ *to* S **do**

$\mathbf{M_{o,temp}} \leftarrow Replicate(\mathbf{M_o}, \mathbf{L_o})$;

$\mathbf{M_{t,temp}} \leftarrow Replicate(\mathbf{M_t}, \mathbf{L_t})$;

$\mathbf{F} \leftarrow F([0,...,T]; \mathbf{C_f})$;

$\mathbf{W_o} \leftarrow \mathbf{F} \circledast IR_o([0,...,T]; \mathbf{M_o})$;

$\mathbf{W_t} \leftarrow \mathbf{F} \circledast IR_o([0,...,T]; \mathbf{M_t})$;

$\mathbf{W_N} \leftarrow \mathbf{N_s} \circ N([0,...,T], \mathbf{N})$;

$\hat{\mathbf{W}} \leftarrow \mathbf{W_o} + \mathbf{W_t} + \mathbf{W_N} + \mathbf{A} \circ \mathbf{F}$;

$\hat{\mathbf{W}} \leftarrow \hat{\mathbf{W}} + \hat{\mathbf{W}} \circledast IR_e([0,...,T]; \mathbf{R_t})$;

$\mathbf{M_o}, \mathbf{M_t}, \mathbf{R_t}, \mathbf{C}, \mathbf{N}, \mathbf{N_s} \leftarrow GradientStep(\mathcal{L}(\mathbf{W}, \hat{\mathbf{W}}), \{\mathbf{M_o}, \mathbf{M_t}, \mathbf{R_t}, \mathbf{C}, \mathbf{N}, \mathbf{N_s}\})$;

end

return $\mathbf{M_o}, \mathbf{M_t}, \mathbf{R_t}, \mathbf{C}, \mathbf{N}, \mathbf{N_s}$

Algorithm details We show abstracted pseudocode for the impact sound source separation optimization algorithm in Algorithm 2. After iteratively optimizing all the parameter variables with this algorithm, we generate separated object and tool sounds by using the generation code in the inner loop, but eliminate components from the generation process which are irrelevant to each separated source we are synthesizing. For example, we can synthesize a "denoised" version of the impact sound by omitting the contribution of the noise $\mathbf{W_N}$ from all summations within the generation code in the inner loop. For synthesizing the object modal sounds, we take $\mathbf{W_o}$ alone as the final sound. To produce the separated tool sounds, including the acceleration sound, we take $\mathbf{W_t}$, add acceleration sound $\mathbf{A} \circ \mathbf{F}$, finally add this result with the convolution of itself and the reverberation filter $IR_e([0,...,T]; \mathbf{R_t})$. For further details, we have included our source code for this in our Supplementary Materials.

Material classification model For training a material classification model which can classify object materials from audio, we first begin with the Sound-20K dataset [40], which has thousands of synthesized audio recordings of different virtual objects of different materials falling on surfaces in a virtual environment. We first filter the available audio samples down to those from virtual objects made of materials we tested for our experiments ("ceramic", "polycarb" "steel", and "wood"), and then balance our dataset by taking the same amount of recordings from each material. We then compute the spectrogram of the audio sample with a window size of 512, to ensure a balance of frequency and temporal resolution, so that the input can capture and characterize the rapidly decaying modal frequency components in each impact. We split our data with a 90-10 train-validation ratio.

Our model takes this input, passes it through a Gated Recurrent Unit (GRU) layer, followed by two fully-connected layers with ReLU activations, and a final fully-connected layer with a softmax activation. During training, we use dropout on the outputs of the GRU and hidden layers to prevent overfitting, and train using the Adam optimizer [29] until a local minimum is reached in the loss on the validation data. Note that this model has not been trained on any real audio data.

Additional classification results In Figure 15, we show the confusion matrices comparing how well our pre-trained material classification model can classify the material of the objects and tools from the original audio versus the audio that our model generates as their "separated" impact audio. The

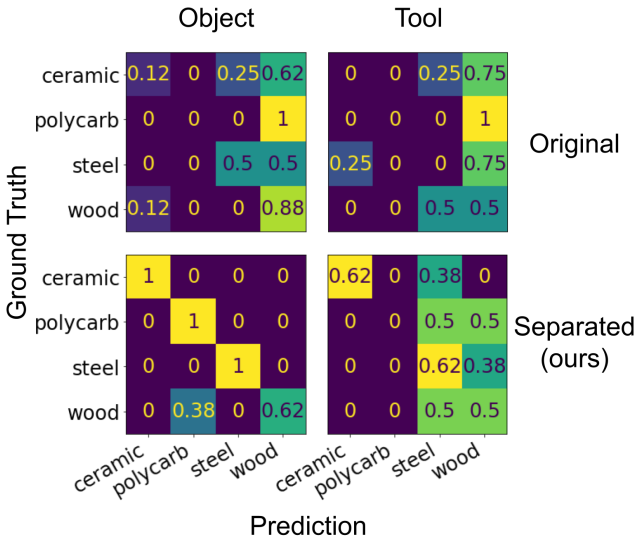


Figure 15: Confusion matrices for classifying material of the struck object and the tool with a classifier trained on Sound20k. (**Top Left**) Classifying the object with the original audio. (**Bottom Left**) Classifying the object with our separated audio. (**Top Right**) Classifying the tool with the original audio. (**Bottom Right**) Classifying the tool with our separated audio.

material classification model clearly struggles to classify the original audio with the correct material. We would expect that the sound the tool makes during the impact *pollutes* the sound that the object makes when impacted. Therefore, we would expect the classifier to occasionally waver between classifying the audio with either the object or the tool material. However, our confusion matrix at the far left of Figure 15 shows that the model classifies the material of every audio sample with the polycarbonate cup as wood, though only two of the eight recordings of striking the polycarbonate cup use the wooden spoon. This suggests that this pollution of the tool and object sound together can cause the classifier to classify the audio as the material of *neither* of the objects.

Our trained classifier model generally struggles with classifying wood sounds correctly. This is especially evident in the tool classification confusion matrices on the right of Figure 15. Wood sounds are very broad depending on the type and structure of the wood, *e.g.*, hardwoods sound very different than soft woods. Therefore, differentiating between wood and other materials with higher dampings, such as polycarbonate, using only sound may be especially challenging.

Young's modulus metrics Young's modulus is a material property which quantifies the stiffness of a material. Knowing an object or tool's material stiffness is important for a robot to reason about that object or tool's affordances. For hard materials, precisely measuring Young's modulus traditionally requires invasive or destructive measurement of precisely fashioned samples [43] or requires highly specialized measurement equipment and dedicated sensors with which robots are generally not equipped [44, 45]. Robots have been shown to be capable of using force sensors to measure soft objects [46] and soft human tissues [47], but these objects have Young's modulus values which are many orders of magnitude lower than those of hard materials such as ceramics and steel. In a modal model of object vibrations, the sound of an object's modal response is highly influenced by the Young's modulus value of its material [21]. Vibrations and sound have been successfully used to infer Young's modulus of materials of rods of known geometry [48] and of simulated objects [22]. Inspired by this work, we determine how accurately we can predict Young's modulus values for the material of each object and tool from the impact sounds separated by our approach and compare to the same source separation baselines we used for the material classification experiment in Section 4.3. We train a regression model on only the Sound20k dataset of synthetic impact sounds of objects of different materials [40], using their Young's modulus values as labels. We then use this model, trained only on synthetic audio, to predict Young's modulus of the materials of the real objects and tools from each approach's source-separated impact sounds. We measure error in terms of percent error of the estimate relative to the ground truth value.

Table 2: Estimating Young's modulus from source-separated audio.

	Object Regression Error (%)		Tool Regression Error (%)	
Model	Dataset Min	Instance Min	Dataset Min	Instance Min
Raw audio	263.4	263.4	243.3	243.3
NMF [38]	174.4	93.4	153.3	97.8
DAP [39]	161.0	71.9	227.7	104.1
DiffImpact (ours)	**59.6**	**46.1**	**118.0**	**86.6**

Young's modulus regression model We used a slightly different model architecture and training regimen for the regression task of estimating Young's modulus from sound versus our model for material classification from sound. We used the entire Sound20k dataset for training and validation, without filtering based on materials, in order to increase the training set size, splitting the dataset with a 90-10 training-validation split. We obtained labels for Young's modulus from the material configuration files accompanying the dataset. Each clip in the dataset is a $\sim$2-second simulated recording of an object bouncing on a surface, so each clip includes the sound from multiple impact events. Because the clips from our source separation audio were each 1 second long and included the sound of exactly one impact event, we designed our architecture to be more invariant to the number of impact events or length of clip. We thus replaced the recurrent unit of our classification model with three dilated 1D temporal convolution layers, each with ReLU activations, followed by a maximum pooling operation across the entire temporal dimension for each feature channel. Similar to our material classification model, we followed this with two fully-connected layers with ReLU activations, with a final fully connected layer to output the final Young's modulus regression estimate. To mitigate overfitting, we trained this model with dropout regularization on the output of the maximum pooling layer and each of the hidden fully connected layers. Because the Young's modulus values for different materials spanned multiple orders of magnitude, we trained against a mean squared logarithmic error loss with the Adam optimizer [29] until the validation error had reached a minimum. Note that this model was not trained on any real audio data.

Young's modulus results The results of this regression task are shown in Table 2. Similar to the results of the classification experiments in Section 4.3, we evaluate the performance of our regression model on the outputs of our framework against its performance on the outputs of baselines using two different schemes, since our baselines do not explicitly differentiate which output comes from which source, object or tool. In the "Dataset Min", we take the minimum between the mean percent error of treating the first output and treating the second output as the object sound. We do the same for tools. Under the "Instance Min" scheme, we take the minimum percent error between the two outputs for each instance instead of across the whole dataset, then take the mean across all instances. Similar to the classification results, both schemes offer the baselines an advantage, but our DiffImpact outperformed them, while also explicitly differentiating between the object and the tool. Also similar to our classification results, our framework performs better on the object sounds than the tool sounds, perhaps for the same reasons which could explain the difference in performance between tool and object classification, detailed in Section 4.3.

Appendix E Model Assumptions

Assumptions about the quantities and features influencing our model are organized in Table 3 by whether they are known, observable and estimated by our model, or unobservable. We either assume unobservable quantities to be constant, select them as a hyperparameter, or estimate them to a relative scale. Priors are induced on estimated quantities by the hyperparameters selected for bounds of the scaling functions (see Appendix A) and those selected for initialization and biases (see Appendix F).

For the recording, we have the ground truth waveform, but for recordings from the wild, the microphone's position, gain, and directionality (*e.g.*, cardioid or omnidirectional) are assumed to be unobservable but constant for each recording. Because these quantities are unobservable, we do not model acoustic transfer, which would depend on the microphone's position relative to the object [49] and properties of air in the environment [50]. We instead assume any effects of acoustic transfer to be part of the impact magnitude and object modal response. For the impact, because we do not observe the microphone position and gains, we cannot estimate absolute quantities of impact force magnitudes without calibration. For the count of impacts, we know exactly one impact occurs in recordings

Table 3: Model assumptions of information and physical quantities.

	Known	Observable (Estimated)	Unobservable (Assumed)
Recording	• Waveform		• Microphone position • Microphone gain • Microphone directionality
Impact	• Count (Sec 4.3)	• Magnitude (relative) • Time scale • Timing	• Count (Sec 4.1 and 4.2) • Magnitude (absolute) • Location
Object	• Instance ID • Tool instance ID (Sec 4.3)	• Mode frequencies • Mode gains (relative distribution) • Mode dampings	• Mode count • Mode gains (absolute) • Contact conditions
Environment		• Static noise gains • Reverb response gains • Reverb response decays	• Geometry

of Section 4.3, but we do not have ground truth counts of the number of impacts in recordings of Sections 4.1 and 4.2 and select this as a hyperparameter as described in Appendix A.1. Our model estimates the time scales (sharpnesses) and timings of impacts (shifted by the unobserved sound travel time from object to microphone), though estimated magnitudes are proportional rather than absolute.

For objects, we only know the object's instance ID (as well as the tool instance ID for Section 4.3). We select a reasonable number of salient modes as a hyperparameter, then estimate the frequencies and dampings of those modes. For mode gains, because we do not have absolute impact magnitudes as previously explained, we estimate a normalized relative distribution. We also do not know the exact contact conditions of the object, which will dampen certain frequencies depending on the region, force, and material properties of each contact on the object [51]. We instead assume contact damping to be negligible or part of the object's intrinsic modal impulse response. Striking an object in different locations excites each mode of the object at a different gain [18], but since we do not observe the location of each impact, we assume the location of impact to be constant for our all our models.

Finally, while we estimate static measurement noise, we have measured neither the geometry nor the standalone impulse response of our environment for reverberations. Room reverb responses are often specifically measured with a small explosion of air to record the independent response of the room [52]. A popular model for a room's reverb response is a modal model, which can require 1000-2000 separate modes to characterize [53]. Because we have no such isolated measurement or knowledge of the room geometry (nearby surfaces' distances, angles, and absorptions) from recordings in the wild, and we cannot separate out reverb mode responses from object mode responses, we instead model only the late stage reverb response of the room with exponentially decaying filtered noise [52].

Appendix F Hyperparameters and Initialization

Here we make some high-level observations about the effects of different choices of hyperparameters and initialization. Specific values can be found in our released code.

Loss function Perhaps the most influential hyperparameters are those of the loss function, the multi-scale spectrogram loss detailed in Section 3.5. With spectrograms, there is an inherent tradeoff between the time resolution and the frequency resolution, which are inversely and directly proportional to the window size, respectively. For the object modal response, frequency resolution is important for precisely estimating modal response frequencies, and time resolution is important for precisely characterizing the decay of each mode to estimate damping. Time resolution is especially important for precisely localizing the onset of the impact sound, which is upstream of characterizing every other feature of the impact sound. For example, mistakenly estimating the timing of the impact to be slightly after the ground truth timing could cause the model to characterize the object's modal impulse response as only the tail end of the ground truth response, where many modes will already have decayed significantly.

The intention of using multiple scales of spectrograms is to attain the best of both worlds in terms of both frequency and time resolution. However, natural spectrograms can present an optimization landscape which is far from smooth. Spectrograms with high time resolution are finer-grained and

less smooth on their temporal axis, causing much more potential for local minima in optimizing the temporal alignment of the synthetic impact sound with the ground truth impact sound. Spectrograms with lower time resolution present a much smoother optimization landscape with respect to timing. This same concept can be an important consideration with respect to the risks of optimizing frequency values of the modal response using spectrograms with large window sizes and high frequency resolution. Even though the final loss is summed across the spectrograms of different resolutions, a given spectrogram could dominate this loss and resulting gradients, such that the other spectrograms could not rescue the optimization from a local minimum. For these reasons, we eliminated the two spectrograms with the smallest window sizes from the loss function of [15] for our loss function on impact sounds. Also for these reasons, whenever we could assume only one impact occurred during a clip as we could in Section 4.3, we initialized the timing of the impact to the location of the maximum of the absolute value of the waveform across the recorded clip.

Generator biases Tuning bias hyperparameters for the different generator functions is important for ensuring faster convergence to reasonable values. During the optimization, each generator is essentially competing to explain different aspects of the sound. Starting the optimization with too high of gains for one generator could cause that generator's output to dominate the sound. For example, the background noise generator could overfit its filter gains to match the sounds of impacts as closely as possible with its time-constant noise.

Frequency resolutions Our Modal IR, Filtered Noise, and Reverb generator functions each could compute outputs for inputs with varying resolutions of frequencies. More specifically, for the Modal IR, the number of modes could be varied, and for the Filtered Noise and Reverb generators, the number of noise bands for each could be varied. The frequency resolutions of these generators presented a tradeoff between expressiveness and robustness to overfitting. For example, in our source separation experiment with a very small dataset, choosing too high of frequency resolutions for both the object and tool modal models in the Modal IR increased the likelihood that an object's separated sound would include faint artifacts of sound from one or more of the tools which struck it. Choosing a lower frequency resolution for each Modal IR forces our model to fit the more salient modal components of each object and tool.