A UNIVERSAL TOOL FOR GENERATING DATASETS FROM AUDIO EFFECTS

Stefano FASCIANI (stefano.fasciani@imv.uio.no)[1],
Riccardo SIMIONATO (riccardo.simionato@imv.uio.no)[1], and
Aleksander TIDEMANN (aleksander.tidemann@imv.uio.no)[1]

[1]*Department of Musicology*, **University of Oslo**, Norway

ABSTRACT

This paper presents an open-source software tool that simplifies and automates the generation of accurate and reproducible datasets from audio effect devices. The tool is universal as it is designed to work with both software and hardware devices, presenting analog or digital interfaces. Specifically, this tool facilitates the process of recording the processed sound while varying effect control parameters, with four different paradigms offered. The tool has an interactive graphical user interface that facilitates interfacing audio effects and configuring dataset generation options. The generated dataset is organized as a collection of sample-accurate files, which makes further processing or analysis with offline scripts simpler. The paper details the tool's design and its extensive set of functionalities, which are informed by research in machine learning modeling of audio effects. While discussing the tool's applications, we also explore its potential in other contexts.

1. INTRODUCTION

1.1 Audio Effects

Audio effects, also known as effect units or effect processors, are electronic devices that alter the sound of an audio source using analog or digital signal processing techniques. These are widely used in music production, music performance, and broadcasting. They comprise of distortions, dynamic, filter, modulation, pitch, and time effects, although hybrid or combined effects are also common. These effects can be standalone devices or integrated within sound synthesizers, mixing boards, amplifiers, or performance-oriented systems. They can be implemented using analog circuitry, digital signal processors, or in some old instances, mechanical components. Standalone hardware effects vary in form factor, including pedals, tabletop units, and rack-mountable devices. The advent of standardized audio plugin formats in the 1990s led to an increase in software implementations of digital audio effects. These process the sound using techniques available exclusively in the digital domain, such as finite impulse response filters, or are designed to emulate classic analog effects. This approach, called virtual analog [1], emerged in the 1990s, and a variety of numerical techniques have been proposed since then, to replicate the nonlinear and time-variant characteristics of these devices.

Audio effects present standardized interfaces for input and output audio signals. Most effects process either one (mono) or two (stereo) audio channels, while larger numbers of audio channels are found in spatialization and surround processors. On hardware devices, the audio interface uses standardized unbalanced or balanced connectors, such as RCA jacks, 3.5mm or 6.35mm TRS or TS phone jacks, or XLRs. These are used to transport AC-coupled, line-level variable voltage analog signals. Some devices may also feature microphone or instrument level inputs. In software devices, uncompressed digital audio is generally exchanged through buffers of channel-interleaved audio samples, where the buffer size depends on the system's audio drivers. Each audio sample is represented as a 16- or 24-bit integer or as a 32- or 64-bit floating point number.

Audio effects typically provide control parameters, enabling users to adjust and customize the sound-altering process. These parameters influence specific, and often perceptually relevant aspects of the audio processing chain. In hardware devices, control parameters are mostly interfaced to variable resistors in the analog circuitry, while, in digital devices, they determine values of variables within the digital signal processing algorithms. Audio effects vary significantly in their control parameters. Besides the considerable variability in their number, the interfaces also differ. In hardware devices, these parameters are usually adjusted through dedicated or shared (i.e., re-mappable) knobs, sliders, switches, and buttons. Alternatively or additionally, they may be controlled also through analog Control Voltage/Gate (CV/Gate) interfaces or through a Musical Instrument Digital Interface (MIDI), with the latter being more widespread. MIDI-controlled parameters, using the common 1.0 standard, are indexed with a 7-bit integer and have a limited 7-bit resolution. The 2.0 MIDI standard, introduced in 2020, overcomes this limitation, but it's yet only available on a limited number of systems. CV/Gate offers theoretically infinite resolution, though the signals can be generated or received by a digital interface with finite resolution, typically in the range of 10 to 16 bits. Software effect units often use the Virtual Studio Technology (VST) or Audio Units (AU) formats. In these cases, all control parameters are indexed with an integer and represented by floating-point numbers in the range of 0.0 to 1.0.

1.2 Datasets of Audio Effects

A dataset generated from an audio effect typically includes output recordings for different values of the control parameters. As audio effects don't generate audio signals, the dataset is intrinsically linked to the selected input signal. The input signal selection largely depends on the specific application for which the dataset is developed, though it commonly includes a variety of stimuli such as sinusoidal tones, sine sweeps, amplitude sweeps, and combinations of noises. Excerpts of various music genres, recordings of individual percussive and non-percussive musical instruments, and singing and spoken voices are also frequently included. Copyright of the selected stimuli determines the extent to which the resulting dataset can be used and released. The output for the selected input signal is recorded for all combinations of the control parameters to be included in the dataset. However, this is not a strict requirement, and input of different durations and content can be used. The number of parameter combinations can vary significantly depending on how many variable parameters and their possible values are included in the dataset. While the term "permutation" might be more mathematically appropriate to identify unique settings of the control parameters, we use "combination" as we formally consider the value of each individual parameter belonging to a different set. Indeed, even after normalization, each parameter included in the dataset may have a different range and sampled values. There is no standardized way to arrange the dataset. However, recent efforts have aimed to minimize the likelihood that dataset differences can lead to significantly different experimental results. This has led to the development of an open-source library that handles datasets in their current distribution modes while regulating possible variability [2]. Common practices in developing an audio effect dataset include: providing a separate uncompressed audio file for each parameter combination with an identical sampling rate, and typically associating filenames to metadata in a Comma Separated Value (CSV) file. Encoding metadata in the filename is a less common option. The sampling rate can also vary significantly across datasets, with 44.1, 48, and 96 KHz being the most common choices. The selected stimuli input file is also usually included in the dataset. Datasets may also be arranged as a single audio file for the recording. In this case, boundaries for segmenting the recorded audio according to different combinations of parameters must be provided.

1.3 Applications and Uses of Datasets

Datasets of audio effects have various applications, and the specific purpose can influence the content and structure of the dataset. One use for these datasets is for modeling an audio effect using data-driven methods such as black- or gray-box approaches. Deep learning techniques have become increasingly popular in this context in recent years. Datasets can also be used to evaluate the accuracy of numerical models, even when these models are developed with explicit white-box techniques that do not rely on the device's data. In addition, these datasets can be utilized for offline precision characterization of an audio effect, measuring level, gain, noise, harmonic and intermodulation distortion, relative phase, crosstalk, and transfer function. Datasets can be used to compare different effect units that are designed to apply similar alterations to audio signals, such as those from different manufacturers, production batches, or after exposure to varied operating conditions or life cycles. Intelligent mixing systems, an emerging field, is another context where audio effect datasets can find significant applications. For example, datasets can be used in developing algorithms for recognizing applied audio effects and estimating parameters of an effect that generates processed signals. Datasets can be used to generate a semantic or perceptually-relevant model of an effect's response to parameter changes, assisting in the navigation, selection, and generation of parameter presets for a specific unit. In a creative context, these datasets can represent a rich and varied sonic corpus due to the significant alterations that audio effects can produce on the sonic content of an audio signal. Such data can be used as material for granular or concatenative synthesis, for music production, or artistic installations.

1.4 Challenges in Generating Datasets

There are several challenges associated with the generation of audio effect datasets. Firstly, it can be a time-consuming and tedious process. Assume three variable parameters are included in the dataset, with 10 sampled values from each, resulting in 1000 combinations. If we record 60 seconds of audio for each combination, it requires 16.6 hours of continuous recording time. Depending on the application, datasets may require a significantly higher number of sampled values for each parameter. While the total recording time might not be a significant issue, changing settings and restarting the recording could potentially introduce errors if executed manually. Consistency across recordings is critical- parameter settings, recording levels, and synchronization of playback and recording must be maintained, sometimes down to sample-accurate levels. If the setup must be dismantled and reassembled daily, there may be accuracy issues due to inconsistencies in the setup, particularly concerning settings controlled by non-digital means, such as gains in the audio interface or in the effect unit. These inconsistencies and errors can be detrimental to specific applications that use the dataset. The results generated by designs using the dataset may present flaws or lack precision, eventually traced back to the dataset itself after lengthy inspections. Automating the dataset generation process can save time and reduce the chances of errors and inconsistencies. However, automation needs to consider the specific interface of the selected audio effect and requires software development and validation that could consume the saved time.

The tool we propose in this paper addresses these challenges, offering fully automated and accurate dataset generation that is compatible with most hardware and software audio effects. It provides a high degree of accuracy and exposes an extensive set of functionalities to users through a comprehensive Graphical User Interface (GUI). The design and implementation, detailed in the follow-

ing sections, are informed by our research in modeling analog audio effects using deep learning techniques, an application that requires highly accurate datasets, as explained later in the paper. The tool, named as Dataset Generator for Musical Devices (DGMD) is available as open-source software at `https://github.com/stefanofasciani/DGMD`. In this paper, we present the DGMD version focused on audio effects, while a separate version focusing on sound synthesizers is still under development.

2. RELATED WORKS

Variable control parameters are found in both audio effects and sound synthesizers. Managing these during dataset generation and handling the potentially large number of combinations pose a common challenge. Furthermore, effects and synthesizers often share similar interfaces and formats. Automatic and offline rendering of software sound synthesizers is a widespread practice in research modeling different aspects of these algorithms. Commonly, a basic synthesis algorithm is explicitly programmed to generate such a dataset. However, software sound synthesizers in VST format have also been used, providing a more realistic and complex study case. Additionally, software sound synthesizers in the VST format have been used, offering more realistic and complex study cases. In this context, RenderMan [1] —a command-line VST instrument host with Python bindings, written in C++—represents a highly effective and efficient option for dataset generation. RenderMan not only facilitates dataset creation but also provides analysis of the rendered audio. It has been utilized in [3] and in [4] for generating datasets to train systems for the automatic programming of sound synthesizers.

The Timbre Space Analyzer and Mapper [5]-a previous work by the first author—part of the conceptual inspiration behind the DGMD tool-is a mixed MATLAB and Cycling '74 Max system. It allows for the analysis and modelling of a sound synthesizer's timbre response, as well as the control of synthesis parameters from various representations of the timbre space. To accomplish this, the system internally computes a dataset from a VST sound synthesizer subject to variations of selected control parameters.

The dataset used to train real-time neural network models of guitar amplifiers, presented in [6], is generated from a hardware analog amplifier. The amplifier's parameters are controllable only through manually operated knobs. To ensure precision, consistency, and repeatability—challenges hard to overcome with a human operator—the authors have developed a robotic control system for the parameter knobs, consisting of an array of electric motors. Another notable aspect of this work is their sampling approach for control parameters. Instead of using a fixed grid, which may lead to overfitting, they employ a random sampling strategy. Parameters are drawn from a uniform distribution for each combination added to the dataset, ensuring an unbiased sampling of the control space throughout the dataset.

3. DESIGN AND IMPLEMENTATION

3.1 Objectives and Choices

The tool we propose for generating datasets from audio effects has been designed to achieve the following objectives:

- It should support any audio effects, ranging from software plug-ins to hardware effects with MIDI, CV/Gate, or manual interfaces.
- It should minimize manual intervention by automating the entire dataset generation process.
- It should be cross-platform, easy to install, and user-friendly, with all functionalities accessible through a single GUI.
- It should support the generation of sample-level accurate and reproducible datasets.

For the implementation, we have selected Cycling '74 Max [2] because it offers the best trade-off between the objectives listed above. It interfaces with audio plug-ins through a native object, while supporting a variety of audio drivers that can cover virtually any sound card needed to interface an external audio effect. Max allows the building of standalone cross-platform applications with complex GUIs, solving any problems related to operating system-specific system libraries, graphic libraries, audio, and MIDI drivers, as well as compiling and building the application. Furthermore, Max is a reasonable choice for ensuring the longevity of our tool. This environment has been updated and maintained for over 30 years and can handle any changes in operating systems that will inevitably occur in the future.

The time required to generate a dataset from software plugins can be significantly reduced using Max's non-realtime audio driver, which renders audio at the maximum rate allowed by the CPU computational power. While the same or better efficiency could potentially be achieved using VST hosts for text-based programming languages, we still see significant value in using Max. Notably, when using Max, the real-time mode can be used to open the plugin's GUI, select variable parameters, and fine-tune their range to be included in the dataset. This can be done while listening to the effect's output before recording the dataset, a crucial feature in this context. Then the non-realtime driver can be used to generate the dataset offline. In contrast, a fully script-based offline approach, aside from not allowing one to listen to, interact with, and view the plug-in's GUI, also requires advanced knowledge of the audio effect controls' range, name, or index. These could vary significantly from those documented in manuals or visible on the plug-in's GUI.

While Max offers precise signal processing objects for sending audio and control parameters to audio effects, as well as for recording the effects' outputs with sample-level consistent accuracy, it does have limitations in handling

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

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

non-native data structures and in implementing algorithms that require iterative loops. To overcome this, we have coded a significant portion of the DGMD in JavaScript, which Max natively supports and integrates through dedicated objects. Utilizing JavaScript advantages the cross-platform aspect, although execution time and synchronization with the signal processing part are not guaranteed. Hence, we have limited all time-critical operations such as interfacing with the audio effect to Max's signal processing components. Non-time-critical tasks, such as file I/O operations, are generally implemented in a combination of JavaScript and Max objects operating at message rate.

3.2 Interfacing Audio Effects

The tool can send and receive audio from effects with one (mono) or two (stereo) channels while controlling up to ten different effect parameters. These are the limits of the current implementation, which can be easily extended. Software plugin effects are hosted directly inside the DGMD, with audio and parameters exchanged using the VST interface. Hardware effects are interfaced using an audio interface to send and receive audio signals. Most audio interfaces also feature a MIDI output, which can be used to send parameters to those effects with parameters digitally controlled via MIDI. Effects with a CV/Gate parameter interface instead, require a more specific type of audio interface with DC-coupled audio outputs, and one audio channel per controlled parameter. In Figure 1 we have illustrated how software and hardware effects are interfaced to the DGMD.

In the case of hardware effects that lack an analog or digital interface for the parameters, we suggest a MIDI-controlled electro-mechanical interface as an alternative to manual control. This type of interface could be constructed using various low-cost microcontroller boards that include a USB class-compliant MIDI interface. The microcontroller would receive MIDI control change messages used to set the angle of an array of 270 servo motors, each corresponding to a specific controlled effect parameter. The microcontroller should feature a number of Pulse Width Modulation (PWM) output pins at least equivalent to the number of parameters to be controlled. Consequently, the audio effect appears to the DGMD interface as if it includes MIDI-controllable parameters. Such hardware and firmware design for interfacing with most manual effects is relatively simple and generic, and we have included an example in the tool's repository. However, the physical connection between the servo motors and the effect unit is device-specific and must be custom-made, for example by 3D printing the required parts to hold the motors in place and to clamp onto the effect unit's knobs.

4. FUNCTIONALITY AND FEATURES

In this section we detail all functionality and features we have included in the DGMD tool, motivating their integration against specific applications. The GUI is visible in Figure 2, where yellow overlay letters are used to label sections serving different purposes. Detailed information about the functionalities is displayed to users when hovering the mouse over each interactive element of the GUI.

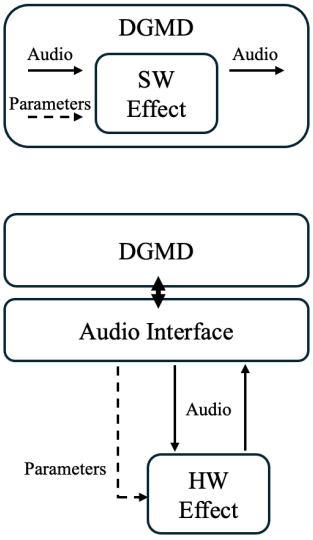

Figure 1. Interfacing between the DGMD tool and audio effects. Software (SW) plugin audio effects are directly hosted within the DGMD, with audio and parameters exchanged through the VST interface (top). In contrast, hardware audio effects are connected via an external audio interface, with parameters controlled through MIDI or CV/Gate signals (bottom).

4.1 Device Selection and Interface

Section 'G' of the GUI allows users to specify the target audio effect device as either an internal software plugin or an external hardware unit. Users can select the number of audio channels for sending the stimulus to the effect and for recording the processed sound, which can be configured as mono or stereo. This section also lets users adjust and monitor the pre-recording gain of the audio signal from the effect and listen through the monitoring loudspeakers.

When working with internal plugins, section 'H' enables the selection of the specific plugin from system folders or the manual loading from a file. The plugin's GUI can be displayed in a separate window, and the 'Monitor Plugin' feature allows users to see the name and value of parameters changed in the plugin's GUI on the DGMD. This is critical because the DGMD tool displays the VST interface's parameters' names and values, which usually differ from those visible on the plugin's GUI. The VST names and values are necessary for determining what information to include in the dataset.

For external audio effects, the audio interface and selected channels for sending and receiving audio are chosen in section 'E'. If the effect has MIDI-controllable parameters, the output port and channel are set and enabled in 'F', and the corresponding MIDI control change numbers for each parameter are set in one of the ten 'D' sections. Additionally, CV control of parameters can be activated in 'F', with the associated specific DC-coupled output audio channel for each parameter defined in 'D'.

The dataset's recording sampling rate is determined in

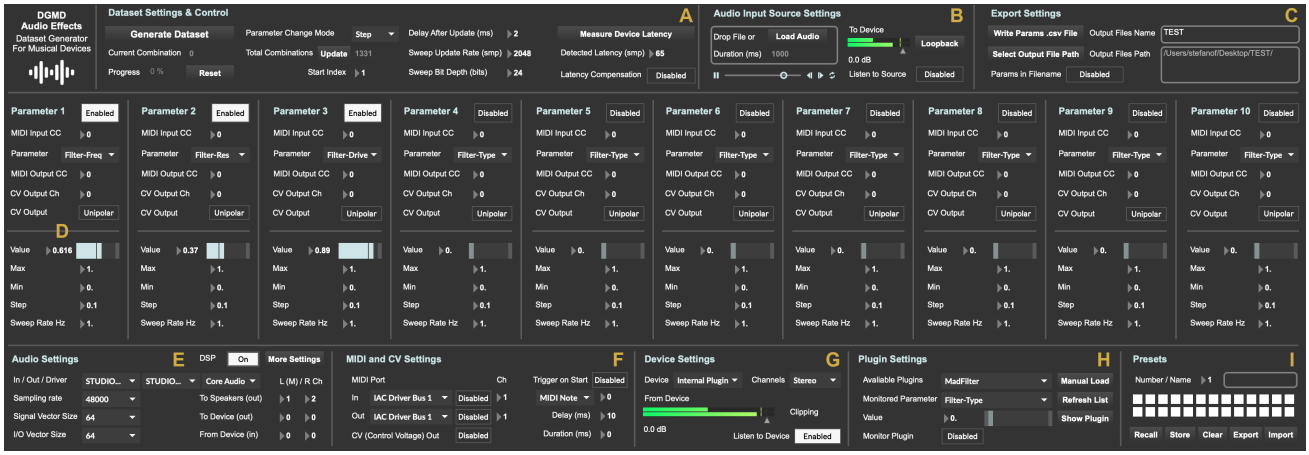

Figure 2. GUI of the Dataset Generator for Musical Devices (DGMD) tool, audio effects version. The capital letters in the yellow overlays indicate the different sections of the interface.

section ‘E’, where users can also choose the audio interface channels for the monitoring loudspeakers to listen to what is sent to and received from the audio effect.

We have included the option for users to manipulate the plugin or external effect’s parameters via a MIDI controller, which is advantageous for testing parameters and ranges to include in the dataset and for setting parameters during manual dataset generation. Users can designate and enable the specific MIDI input port and channel for this purpose in ‘D’, and set the parameter-specific MIDI control change numbers in sections ‘D’.

4.2 Parameters Options and Change Modes

The GUI of the DGMD, as shown in Figure 2, features ten identical sections labeled ‘D’, each corresponding to a specific effect parameter to be varied for dataset generation. A parameter value will only be changed by the tool during dataset generation if it is set to ‘Enabled’ in the GUI. To generate the dataset, at least one parameter must be enabled. Enabled parameters on the GUI don’t necessarily have to be situated next to one another. For software plugin effects, a drop-down menu at the top of the ‘D’ sections allows users to select a parameter from the list of those available from the VST interface. For external hardware effects, within the same section, users can choose either the MIDI control change number or the audio channel for the CV signal. When selecting a CV signal, it is also possible to specify whether the controlled parameter is unipolar or bipolar, enabling the DGMD to map the parameter range to the appropriate output voltage values.

In the bottom portion of each section ‘D’, the current value of the parameter is shown both as a numerical value and as a horizontal slider. These indicators are interactive, allowing user adjustment and also changing automatically during dataset generation. Parameter values are consistently represented in the normalized range from 0.0 to 1.0. Additionally, users can select a specific range of interest-from ‘Min’ to ‘Max’-for each parameter to be varied in the dataset, helping to reduce the time required for generation and to limit the overall size of the output files.

The dataset can be generated according to four different ‘Parameter Change Mode’, specified in the section ‘A’: *Step*, *Random*, *Manual*, or *Sweep*.

In *Step* mode, the DGMD generates the dataset by sampling the variable parameters using a regular grid. Each parameter is sampled using an independent ‘Step’ value, which users can specify in section ‘D’. For instance, the settings displayed in Figure 2 show two enabled parameters, both with a selected range from 0.0 to 1.0. The first, however, has a step of 0.1, resulting in 11 different values, while the second has a step of 0.2, yielding 6 different values. As a result, and also as automatically indicated in section ‘D’ before dataset generation, the dataset will include 66 different parameter combinations. This provides users with an indication of how long the dataset generation will take, given the selected settings. While the current lowest step value is 10^{-5}, it is important to note that when parameters are controlled via MIDI, values are sent as 7-bit integers. In this case, appropriate step values are calculated as $1/2^N$, where N is the number of bits, ranging from 1 to 7, to consider. In devices with CV-controlled parameters, stages of quantization may also be present, especially when the signal processing is handled digitally. In these cases, CV analog signals may be acquired by ADCs with resolutions ranging from 10 to 16 bits.

Upon initiating dataset generation, the DGMD initializes a generator of parameter combinations. Within this generator, each combination is associated with a unique integer index. These indices are used progressively to query the generator, which then produces combinations on demand, one at a time, instead of storing them in a table—an approach that could consume a significant amount of memory. Subsequently, the parameter combinations are sent sequentially to the audio effect in ascending order according to their index. For each combination, the audio stimulus is sent to the audio effect, and the output is recorded. Audio data is initially read from and written into memory buffers to ensure sample-accurate synchronization. It is transferred to disk only after recording is complete, and before a new parameter combination is sent to the effect. A separate audio file is created for the output recording of

each parameter combination. Users have the option to set a delay time between updating the parameters on the audio effect and the start of playback for the input stimulus as well as the recording of the output. This delay accommodates any latency in the parameter update response time, which may vary from several milliseconds to fractions of a second, depending on the specific interface. In this case, an effect with electromechanically-controlled parameters represents the worst-case scenario.

Given that the dataset may include a large number of parameter combinations, potentially leading to a lengthy generation process, we have introduced a feature that allows users to begin generation from any selected parameter combination index. Coupled with the ability to save all settings in user-defined presets, the deterministic and reproducible behavior of the generator of parameter combinations, and the fact that each combination produces a separate audio file, users are empowered to create large datasets over multiple sessions without compromising the consistency or accuracy of the dataset.

In *Random* mode, the DGMD generates the dataset by drawing values from a series of independent uniform distributions, each corresponding to a different parameter. These distributions are confined to the selected range for each parameter, and in this context, the step value serves as a quantizer. This can be particularly useful when parameters are controlled using a low-resolution interface. For instance, with a range from 0.0 to 1.0 and a step of 0.01, only 101 distinct values can be drawn. Similar to step mode, in random mode parameter combinations are generated and sent to the audio effect on the fly, one at a time. However, in this case, the process is not reproducible, which is why the generated combinations are progressively stored in a memory table. Users need to specify the number of different random parameter combinations to be drawn to complete the dataset. The overall automated dataset generation process is the same as that described for step mode, including the option to delay the initiation of recording after dispatching a new set of parameters to the effect, and the ability to start from an index greater than one. However, the latter only affects the numbering of the recorded audio files, as a previously generated table including random combinations cannot be reconstructed. In such cases, users must manually merge the tables, which are exported to a file.

When the dataset is used to train a machine learning model of the audio effect, and the parameters are employed to condition the model's inference, sampling the parameters with a high-density uniform grid may lead to overfitting. In this scenario, random sampling emerges as a more effective strategy to mitigate overfitting. Nonetheless, for sparser grids — that is, those with fewer sampled values per parameter — a uniform sampling ensures better coverage of the effect parameter space. Still, a uniform grid, even if dense, might be the preferred choice for applications focused on analyzing, characterizing, or comparing the response of an audio effect.

The *Manual* mode enables users to set the parameters of the effect by hand. This mode is only semi-automated; each parameter combination is individually added to a memory table when users manually initiate the process of sending the stimulus to the audio effect and recording its output. This mode may serve various purposes: for example, it can be used to sample the parameter space using criteria different from those previously mentioned, likely involving only a limited number of combinations, or it can assist in the generation of datasets with external hardware audio effects with a manual interface for parameter adjustment. In such cases, users should duplicate the settings made on the effect unit within the DGMD interface before triggering the recording for each combination.

The *Sweep* presents major differences compared to other modes. Parameters are continuously varied while recording the dataset, each following a triangular wave with arbitrary range and frequency. In section 'D' of the GUI, it is possible to set the 'Sweep Rate', which is the frequency of the triangular oscillator controlling each individual parameter, which spans from the individual 'Min' to 'Max' defined in the same section. In this case, the concept of parameter combinations no longer holds, and the desired dataset size is expressed by the number of times the playback of the input stimulus signal is repeated. For each repetition, the effect output signal is recorded into an audio file. Also, the sweeping parameters are recorded at audio rate into separate multichannel audio files. As the recording processes are synchronized, each track on both resulting files presents an identical duration expressed in number of samples.

Control parameters for audio effects with a CV interface are transmitted as an audio signal. However, for all other interfaces, the triangular waves need to be sampled, and their values transmitted to the effect device. In section 'A' of the GUI, users can specify the 'Sweep Update Rate,' expressed as the number of audio samples, a feature effective only when the CV output is disabled. This determines the rate of a clock driving a sample-and-hold block, which downsamples the triangular waves and generates the actual set of values sent to the effect. Consequently, when the control parameters for the audio effects are recorded, the resultant triangular wave appears stepped. Moreover, to account for the eventual limited resolution of the interface controlling the parameters, in section 'A' users can select the bit depth for quantizing sweep values before recording.

We introduced the sweep mode for generating datasets that can be used to assess how audio effects respond to continuous variations of parameters. It can also serve to compare the behavior of an audio effect model versus the original device when parameters are varied continuously and gradually. Besides, a dataset wherein effect parameters are not fixed during the audio recording can be used for experimental training of machine learning models of audio effects. This allows to explore the extent to which such an approach might yield better results compared to models using a grid of fixed parameter combinations, which are required guessing outputs for unseen parameter values. The information embedded in the dataset and the potential ability of a model to learn from it can significantly vary with the selected rates. Slower variations may prove ad-

vantageous for training models. Also, in order to maximize coverage of the parameter space, the rates of parameters should not bear an integer multiple relationship. For repeatability in dataset generation, the phases of the triangular oscillators are reset to zero at the beginning of the dataset generation process.

4.3 File Input and Outputs

The audio file including the stimulus signal utilized as input for the audio effect can be loaded in section 'B' of the GUI. The stimulus is reproduced automatically during the dataset generation, while the GUI allows users to manually playback or loop the signal to test the audio effect while adjusting the dataset generation options. This section also lets users adjust and monitor the pre-effect gain of the stimulus signal and listen through the monitoring loudspeakers. We have chosen to use a stimulus from a file rather than an integrated signal generator as the latter may not be as comprehensive in terms of options as existing libraries or tools. Also, an integrated stimulus generator may fall short when users want to use segments of music or instrument recordings as stimuli.

When recording the output of an external hardware audio effect, the signal undergoes processing by the effect device, but it also passes through the output and input of the audio interface. While most audio interfaces provide relatively transparent sound, there could be setups where this might not hold true or applications where the contribution of the audio interface can not be overlooked. Consequently, we have added an option in section 'B' to record the stimulus file after the audio interface loopback. In this scenario, the output of the audio interface must be connected to its input. When training machine learning models of an audio effect, it is more appropriate to use the recording of the stimulus loopback as input rather than the original stimulus file.

The DGMD generates several output files. In section 'C' of the GUI, users can specify the directory where output files are saved and a prefix used for all generated files. In *Step*, *Random*, and *Manual* modes, the tool generates a separate uncompressed mono or stereo wave audio file for each parameter combination included in the dataset, recording the effect output. Audio filenames include an integer index, which serves to retrieve the associated combination of parameters from a table exported as a Comma Separated Value (CSV) file. When working with internal software plugins, the CSV file also includes the VST parameter names. Though the CSV file with parameter combinations is exported upon completion of dataset generation, users can manually export the file if the dataset generation is manually interrupted. In all modes, parameter combinations are generated internally and cannot be imported from an existing CSV file. The generator of parameter combinations for step mode is initialized when the 'Update Step Mode Combination' button in section 'A' is pressed. The generator for step mode and the memory table that holds parameter combinations for random and manual modes are cleared when the 'Reset' button is pressed. Additionally, in section 'C,' users can choose to incorporate the associated parameter values also into the names of the recorded audio files.

In *Sweep* mode, two uncompressed wave audio files are generated for each repetition of the input stimulus signal. The first file contains either a mono or stereo recording of the output processed by the effect. The second file is a multichannel wave file with many channels equivalent to the number of enabled parameters, and it includes recording at audio rare of the sweeping parameters. The filenames of both audio files include an integer index, which facilitates pairing the recorded output with the associated sweeping parameters. As previously discussed, these signals undergo a process of downsampling when the CV parameter output is not enabled. Given that parameter values change very frequently in this mode, recording them as audio signals ensures perfect alignment with the recorded output and simplifies the process of utilizing the dataset - particularly when modeling the audio effect using machine learning techniques requiring matching input-output pairs.

The generated wave audio files have a sampling rate identical to the system sampling rate, which is set in section 'D' of the GUI. The sample format is configured to a 32-bit floating point for recording the audio effect output and a 24-bit integer for parameter recording in sweep mode.

4.4 Latency Compensation

Audio effects may present some input-output latency, which varies based on whether they are analog or digital, as well as their internal signal processing architecture. In the DGMD environment, as well as in other digital audio recording environments, there are other components contributing to add significant latency.

Audio effects can present some latency between the input and output audio. This latency varies depending on whether the effects are analog or digital, and it is also influenced by the internal signal processing architecture. Within the DGMD environment, as well as other digital audio recording environments, other components contribute further, adding significant latency. For internal software plugins within Max 'Signal Vector Size' has an impact on the latency. This setting specifies the number of audio samples processed at once by Max's signal processing routines. For external hardware effect units, Max 'I/O Vector Size' accounts for additional latency, as this determines the number of audio samples buffered and exchanged with the audio interface at a time. This value usually has a minimum setting of 64 samples for most audio interfaces. Both 'Signal Vector Size' and 'I/O Vector Size' can be adjusted in section 'E' of the GUI. This additional latency introduced by the DGMD environment determines a significant misalignment between the input stimulus signal and the recorded effect output, which can be detrimental for effect modeling purposes.

The DGMD tool includes a utility to detect the latency, measured in audio samples, of the audio effect, taking into account all additional components mentioned above. When measuring, the audio effect's parameters should be set to return a signal that is as dry as possible. Measurements can be initiated in section 'A', where the detected latency is displayed. Users have the option to automatically

compensate for this latency. In this case, recordings of the effect output continue past the end of the stimulus signal by an equal number of audio samples to the detected latency. Once the recording ends, and before being written to file, a number of samples equal to the detected latency are removed from the start of the recorded audio buffer. This ensures alignment between the effect input and output signals. Users can use the latency measurement utility with different values of 'Signal Vector Size' and 'I/O Vector Size' to determine the actual latency of the effect, which should be a constant offset added to the buffering-related latency. Then, this value can be manually subtracted from the detected latency value to generate a dataset that still accounts for the internal latency of the effect.

4.5 Other Functionalities and Features

All user-defined settings, which determine the interfacing of the effect and dataset generation options, can be saved into a preset. Each preset is associated with a user-defined name and an integer index in section 'I' of the GUI. These presets can be recalled to enable reproducible dataset generation, which is essential when a single dataset generation task needs to be executed over multiple sessions. The entire collection of presets can be exported to, or imported from a file.

Audio effects can include Low-Frequency Oscillators (LFOs) that independently vary one or more parameters in the internal audio signal alteration process. In some cases, an external trigger can reset the phase of the LFOs to zero. As such, in section 'F', we have provided a utility to send to the effect a Gate signal or a MIDI note with arbitrary delay and duration each time the playback of the input stimulus begins and output recording starts. This feature allows to obtain consistent and reproducible influence of the LFOs on the dataset.

Given that the dataset generation process can be lengthy, the progress percentage is displayed in section 'A' of the GUI. The overall duration can be estimated by multiplying the number of combinations by the duration of the stimulus file. This estimation does not account for minor delays due to parameter updates and file-writing to disk. However, if an internal software plugin is being used, users can switch the audio driver to 'NonRealTime' in section 'E'. This significantly speeds up the generation process as Max's signal processing routines are executed offline.

Lastly, section 'A' of the GUI dynamically changes with the selected parameter change mode, as the names, availability, and functionality of the options may vary across modalities. Extensive information is provided to users in small pop-ups when they hover the mouse over each interactive element of the GUI.

5. CONCLUSIONS

We have introduced a tool called Dataset Generator for Musical Devices (DGMD), specifically the audio effect version. DGMD is a standalone software for generating precise and reproducible datasets from any software and hardware audio effect. This tool offers a rich set of features and functionalities which are informed by research in the field of machine learning modeling of audio effects processing. Both DGMD and its accompanying video documentation are freely available as open-source software at `https://github.com/stefanofasciani/DGMD`.

6. REFERENCES

[1] V. Välimäki, S. Bilbao, J. O. Smith, J. S. Abel, J. Pakarinen, and D. Berners, "Virtual Analog Effects," in *DAFX: Digital Audio Effects*. John Wiley & Sons, Ltd, 2011.

[2] R. M. Bittner, M. Fuentes, D. Rubinstein, A. Jansson, K. Choi, and T. Kell, "mirdata: Software for reproducible usage of datasets," in *Proceedings of the 20th International Society for Music Information Retrieval (ISMIR) Conference*, 2019.

[3] M. J. Yee-King, L. Fedden, and M. 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, 2018.

[4] G. L. Vaillant, T. Dutoit, and S. Dekeyser, "Improving Synthesizer Programming From Variational Autoencoders Latent Space," in *Proceedings of the 24th International Conference on Digital Audio Effects (DAFx)*, Sep. 2021.

[5] S. Fasciani, "TSAM: a tool for analyzing, modeling, and mapping the timbre of sound synthesizers," in *Proceedings of the 13th Sound and Music Computing Conference (SMC)*, Hamburg, Germany, 2016.

[6] L. Juvela, E.-P. Damskägg, A. Peussa, J. Mäkinen, T. Sherson, S. I. Mimilakis, K. Rauhanen, and A. Gotsopoulos, "End-to-End Amp Modeling: from Data to Controllable Guitar Amplifier Models," in *Proceedings of the 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Island, Greece, 2023.