Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«ЗАСТОСУВАННЯ АРХІТЕКТУРИ MIDI-DDSP ДО ЗАДАЧІ ЗМІНИ ТЕМБРУ МОНОФОНІЧНИХ МУЗИЧНИХ ІНСТРУМЕНТІВ»**

Виконав: студент 4-го року навчання
освітньої програми «Комп'ютерні науки»,
спеціальності 121 Комп'ютерні науки

Бараннік Володимир Вікторович

Керівник: Швай Надія Олександрівна,
кандидат фіз.-мат. наук, ст. викладач

Рецензент: Шаповал Наталія Віталіївна,
к.т.н., старший викладач каф. штучного інтелекту ІПСА НТУУ «КПІ ім. Ігоря Сікорського»

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 20___р.

Київ - 2023

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., доктор фіз.-мат. наук
_____*Олійник Б.В.*
(підпис)
“____”_____ 2022

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
для кваліфікаційної роботи
студенту 4-го курсу, факультету інформатики
Баранніку Володимиру Вікторовичу

Тема: «Застосування архітектури MIDI-DDSP до задачі зміни тембру монофонічних музичних інструментів»

Зміст кваліфікаційної роботи:

Анотація.

1. Вступ.
2. Огляд літератури.
3. Методи оцінки якості зміни тембру.
3. MIDI-DDSP як архітектура для зміни тембру.
4. SynthCoder як архітектура для ресинтезу та зміни тембру.
5. Застосування у режимі реального часу.

Висновки.

Список використаних джерел.

Дата видачі “____”_____ 2022 Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____ 2023р.

№ з/п	Перелік робіт	Термін виконання етапу	Підпис наукового керівника	Дата ознайомлення наукового керівника	Примітка
1.	Отримання теми кваліфікаційної роботи.	01.09.2022			
2.	Ознайомлення з темою кваліфікаційної роботи.	01.10.2022			
3.	Розробка плану та структури роботи.	14.10.2022			
4.	Робота з науковою літературою, опис основних означень.	20.02.2023			
5.	Дослідження результатів отриманих в літературі.	13.03.2023			
6.	Проведення основних експериментів.	11.05.2023			
6.	Робота над текстовим оформленням результатів.	11.05.2023			
7.	Попередній захист кваліфікаційної роботи.	11.05.2023			
8.	Виправлення помилок та проведення додаткових експериментів.	22.05.2023			
9.	Захист кваліфікаційної роботи.	29.05.2023			

Науковий керівник _____
(ПІБ)

Виконавець кваліфікаційної роботи _____
(ПІБ)

Анотація

Тембр – надзвичайно важлива складова музики. З одного боку, сучасні музиканти часто прагнуть мати доступ до цифрових копій дорогих і складних у використанні акустичних інструментів. Крім того, існує великий попит на нові, експериментальні звуки. Зміна тембру з використанням методів глибинного навчання – перспективний спосіб вирішення обох задач. Однак наявні техніки зіштовхуються із суттєвими труднощами, пов'язаними з керованістю самими моделями та реалістичністю звуку. На щастя, існує інший підхід – диференційована цифрова обробка сигналів (DDSP) – який задовільним чином розв'язує вищевказані проблеми, хоч і має певні вади, які не можна ігнорувати. Її покращена версія, MIDI-DDSP, демонструє значно більш якісні та реалістичні результати. Однак її недолік полягає в тому, що вона оперує суто символічними вхідними даними – нотами й штрихами. У нашій роботі ми ставимо за мету оцінити можливості MIDI-DDSP у задачі зміни тембру, а також досягти кращого розуміння DDSP та MIDI-DDSP архітектур. Проведені експерименти показують, що MIDI-DDSP значно випереджає оригінальну DDSP архітектуру в задачі зміни тембру. Ба більше, ми показуємо, що можна значно спростити MIDI-DDSP архітектуру, досягнувши при цьому сумірної якості зміни тембру. Наостанок, ми досліджуємо можливості роботи спрощеної моделі у режимі реального часу, відкриваючи цим двері для подальшої адаптації всієї MIDI-DDSP архітектури. Наша робота має безпосередню практичну цінність і розширює розуміння застосувань методів глибинного навчання в домені аудіо.

Table of Contents

Abstract 4

1 Introduction 5

1.1 Relevance . . . 5

1.2 Objectives . . . 5

2 Related work 7

2.1 Brief overview of neural synthesis architectures . . . 7

2.2 DDSP as a neural synthesis and timbre transfer architecture . . . 8

DDSP-VST: realtime DDSP . . . 9

2.3 MIDI-DDSP neural synthesis architecture . . . 10

3 Evaluation of timbre transfer quality 13

3.1 Methodology . . . 13

Multi-scale spectral loss and why it is not suitable . . . 13

Mean opinion score and why it is not suitable . . . 13

Pairwise comparisons . . . 14

3.2 Tools . . . 15

Telegram bot . . . 15

3.3 Data collected . . . 15

4 MIDI-DDSP as a timbre transfer architecture 16

4.1 Hypothesis . . . 16

4.2 Experiments description . . . 16

4.3 Results . . . 17

Deliberately mismatching input audio . . . 20

5 SynthCoder as a timbre transfer architecture and why it fails 22

5.1 Hypothesis . . . 22

5.2 Description of experiments . . . 22

5.3 Results . . . 23

Resynthesis and timbre transfer evaluation . . . 23

Ablation study and comparison to original DDSP architecture . . . 25

6 Real time applications **26**
6.1 How to adapt MIDI-DDSP model to a DDSP-VST pipeline 26
6.2 Conversion to TFLite and proof of concept 26
6.3 MIDI-DDSP and increased frame size 27
Hypothesis 27
Experiments 27
Results 27

Summary **29**

References **30**

Abstract

Timbre plays a crucial role in music. On the one hand, those who make it often seek for more realistic digital copies of expensive and hard-to-play acoustic instruments. On the other hand, there is also a strong desire for new experimental sounds. Neural timbre transfer (the process of changing the timbre while preserving semantics) offers a promising solution, but existing techniques often struggle with realism and interpretability. The Differentiable Digital Signal Processing (DDSP) approach shows potential and is capable of performing timbre transfer but has room for improvement. The MIDI-DDSP, an enhanced version of DDSP, offers significantly better audio quality and realism. Nonetheless, it operates exclusively with symbolic inputs, such as MIDI notes and diverse expression controls. In this work, we aim to assess the timbre transfer capabilities of the MIDI-DDSP architecture and achieve better understanding of DDSP and MIDI-DDSP architectures. Through a series of experiments, we show that MIDI-DDSP significantly outperforms original DDSP architecture in the timbre transfer task. We also show that it is possible to greatly simplify MIDI-DDSP architecture while still achieving decent timbre transfer quality. At last, we explore the possibilities of running the simplified model in real time, thereby paving the way for real-time applications of the full MIDI-DDSP model. This research contributes to practical applications directly and provides insights into deep learning techniques in the audio domain.

1 Introduction

1.1 Relevance

Timbre is one of key aspects of any musical sound. Some genres, such as ambient and spectral music, are completely devoted to it [1][2]. There are 5934 virtual instruments (all of which generate sound) and 8153 effects (all of which process the sound and some manipulate the timbre) in the KVR database [3]. The compound annual growth rate of the music production is expected to be 7.31% and virtual instruments are "widely utilized by practitioners as well as non-professionals as the products offer a similar experience as a physical musical instrument" [30]. These facts indicate a clear demand for both new, musically interesting timbres and for more affordable digital copies of real instruments (both acoustic and analog ones).

Neural timbre transfer may be an alternative to existing solutions, providing us with interpretable, realistic and easy-to-get results, thus satisfying both demands. However, as we show in our work, most of existing neural synthesis and timbre transfer techniques struggle with at least one of the aformentioned aspects.

Differentiable digital signal processing (DDSP) approach promises to solve most of the existing neural synthesis issues and is even capable of performing timbre transfer [6]. However, the output quality is far from perfect [20].

Hence, in our work, we try to find a better solution to the task of realistic neural timbre transfer. Our objective holds immediate practical value and can also yield valuable insights into applications of deep learning in the audio domain.

1.2 Objectives

- Our main goal is to assess whether MIDI-DDSP [20] architecture is capable of performing timbre transfer.

- Our secondary goal is to better explain the performance of DDSP [6] & MIDI-DDSP models; to show some nuances that were omitted or explained poorly in the original papers.

- We constrain ourselves to the monophonic musical instruments, meaning that they play only one note at a time. Such constraint is useful as it reduces the

overall complexity of the problem by allowing to solve the polyphony and timbre transfer tasks separately.

To achieve our goals, we have conducted a series of experiments to assess the capabilities and restrictions of DDSP & MIDI-DDSP architectures:

1. Assessing timbre transfer capabilities of the MIDI-DDSP model trained on the violin dataset.
2. Assessing timbre transfer capabilities of the MIDI-DDSP model trained on the multi-instrument dataset.
3. Assessing timbre transfer capabilities of the MIDI-DDSP's SynthCoder trained on the violin dataset.
4. Assessing timbre transfer capabilities of the MIDI-DDSP's SynthCoder trained on the multi-instrument dataset.
5. Why SynthCoder is such good resynthesizer and bad timbre transfer model. Ablation study.
6. Real time applications of MIDI-DDSP and SynthCoder.
7. The role of frame size in SynthCoder.

We have also developed a telegram chat-bot to evaluate timbre transfer capabilities of different neural networks. It is available at our github page and can be used in other projects as well.

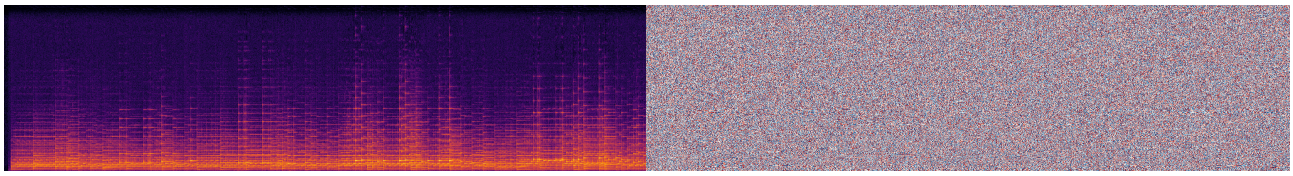
2 Related work

2.1 Brief overview of neural synthesis architectures

In general, most of audio synthesis models can be divided into two groups [7]:

- Those that generate waveforms directly:
  WaveNet [31], WaveRNN [9], DiffWave [8], Jukebox [18] and others.
- Those that generate some representation (e.g. magnitude spectrogram [28]) that is being converted to waveforms by some algorithm (e.g. by Griffin-Lim [15]):
  GANSynth [10], TimbreTron [27] and others.

There is no evidence for which kind of modelling is inherently better. However, due to the specific choice of the non-waveform output representation in the second type of the models (which also needs to be transformed to the waveform somehow), it is easy to lose some important information and generate waveforms of poor perceptual quality. For instance, when the model runs a STFT on the input waveform and preserves the magnitudes only, it then losses phase information which is crucial to human sound perception [7] [20].



(a) Magnitudes (b) Phase (imaginary part)

Figure 1: Plot of magnitudes and imaginary parts of STFT output [7]

The problem is that we cannot simply add phase information in addition to magnitude information because it is nearly impossible to model phase distribution in neural networks. Phase is an angle and it wraps up. Moreover, in the real-world audio, when the magnitude is close to 0, the phase is effectively random as the noise is clearly dominating over the signal [7]. Nevertheless, other representations could be used, such as instantaneous phase $\phi(t)$ and frequency $f(t)$ of the signal $x(t)$, which are shown to improve the audio quality in the IF-GANSynth

architecture [10]:

$$\phi(t) = \arg\left[x(t) + i \cdot \mathcal{H}\{x(t)\}\right]$$
$$f(t) = \frac{1}{2\pi} \cdot \frac{\mathrm{d}}{\mathrm{d}t}\left[\arg\left(x(t) + j \cdot \mathcal{H}\{x(t)\}\right)\right]$$

where $\mathcal{H}\{x(t)\}$ denotes the Hilbert transform of $x(t)$.

However, IF-GANSynth still has some minor phase issues. [10]

2.2 Differentiable digital signal processing (DDSP) as a neural synthesis and timbre transfer architecture

Differentiable digital signal processing (DDSP) [6] takes a different approach. Instead of generating the waveform directly or generating some audio-domain representation that gets converted to a waveform, it generates parameters for harmonic and noise synthesizers, which in turn generate the waveform. Such design has a strong inductive bias and solves the phase consistency issues, as the proposed harmonic synthesizer is parameterized only by fundamental frequency ($f0$) and per-harmonic amplitudes. Consequently, the neural network is responsible for only generating the fundamental frequency $f0$ and the synthesis parameters (amplitudes, harmonic distribution, and noise magnitudes). At last, the output of synthesizers can be optionally processed by the reverberation module.

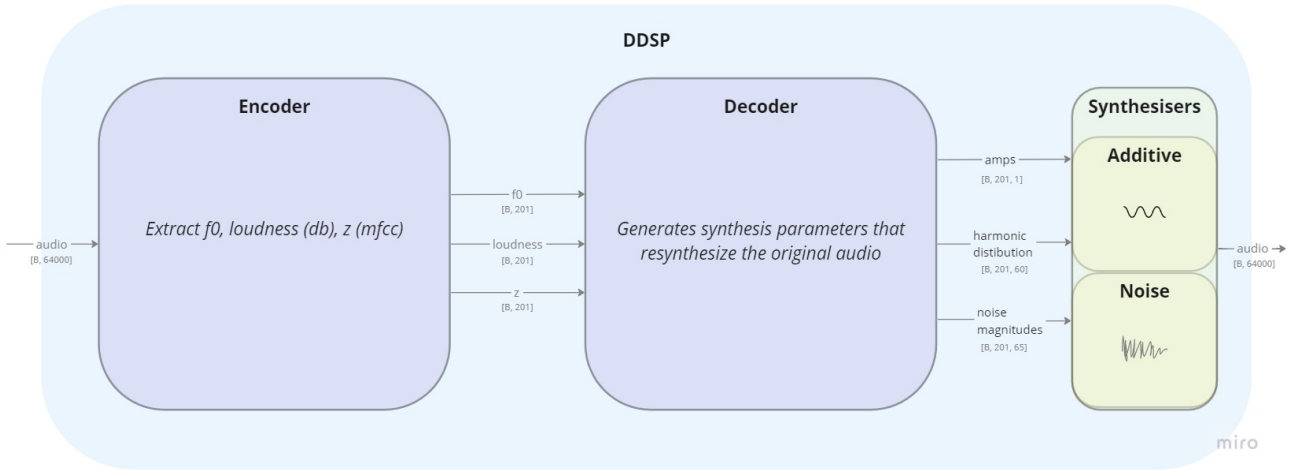


Figure 2: DDSP architecture.

As shown in original paper, DDSP architecture is capable of performing a range of tasks such as resynthesis, denoising, dereverberation and timbre transfer [6].

Encoder The encoder takes audio waveform as input and consists of:

1. Pretrained pitch-extraction CREPE model. [5]
2. Deterministic power or loudness extractor.
3. Optional $z(t)$ extraction model that uses GRU-RNN [19] and MFCC to extract features from input audio.

It outputs per-frame fundamental frequency $f0(t)$, loudness $ld(t)$ and feature vector $z(t)$.

Decoder The decoder takes $f0(t)$, $ld(t)$ and $z(t)$ as inputs and consists of:

1. GRU-RNN.
2. Stack of fully connected layers.

It outputs amplitudes and harmonic distribution for harmonic synthesizer and noise magnitudes for a noise synthesizer.

Number of parameters The model has 15,960,226 parameters, 12,751,176 of which come from the pretrained pitch-extraction CREPE model.

DDSP-VST: realtime DDSP

Another advantage of the DDSP architecture is that it requires little time to perform inference (as compared to autoencoder models) [6]. As a result, Google Magenta developed a C++ VST plugin [24] called DDSP-VST which runs the optimized DDSP TFLite models in real time [11].

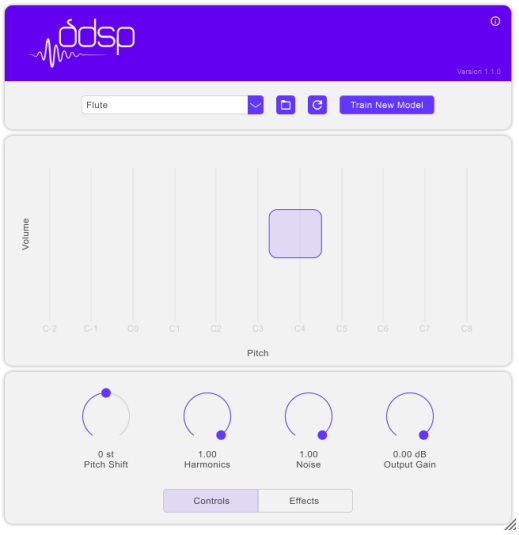


Figure 3: Screenshot of the running DDSP-VST plugin window.

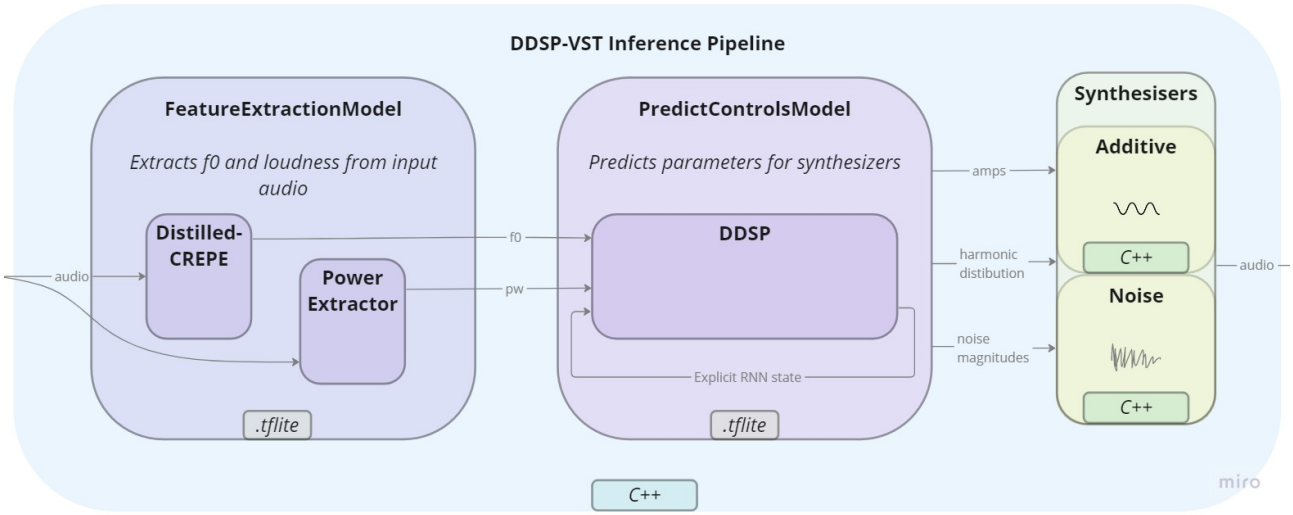


Figure 4: DDSP-VST Inference Pipeline

The DDSP-VST inference pipeline is similar to the original DDSP architecture, but it also has some key differences, which include:

1. There are two separate models in the DDSP-VST: one for feature extraction and the other one for synthesis parameters prediction.
2. Synthesizers are implemented in C++ to achieve greater performance [16].
3. Feature extraction model contains a distilled version of a full CREPE model [16].
4. PredictControlsModel takes power as input (instead of loudness).
5. The pipeline processes a single audio buffer of 1024 samples with scalar $f0$ and pw values (meaning that the frame size is 1024 samples as well).
6. As a result, PredictControlsModel handles state explicitly. The state is retained in memory between consecutive pipeline runs.
7. Both models must be converted to a TFLite format [26] in order to be ran from the C++ environment.

2.3 MIDI-DDSP neural synthesis architecture

MIDI-DDSP is the overhauled version of the DDSP architecture which aims to separate timbre and performance aspects of the sound [20]. To achieve this, it utilizes two separate modules: SynthCoder and ExpressionDecoder.

SynthCoder SynthCoder is a resynthesizer that extracts synthesis parameters which reconstruct the input audio.

ExpressionDecoder ExpressionDecoder is a module that extracts interpretable expression parameters from SynthCoder's synthesis parameters in a deterministic way. There expression parameters are: volume, volume fluctuation, vibrato, brightness, attack and volume peak position. Based on these parameters, it modifies $f0$ in an autoregressive way and produces synthesis parameters for the output audio.

Please refer to Figure 5 which shows the general architecture of the MIDI-DDSP model and Figure 6 for more details.

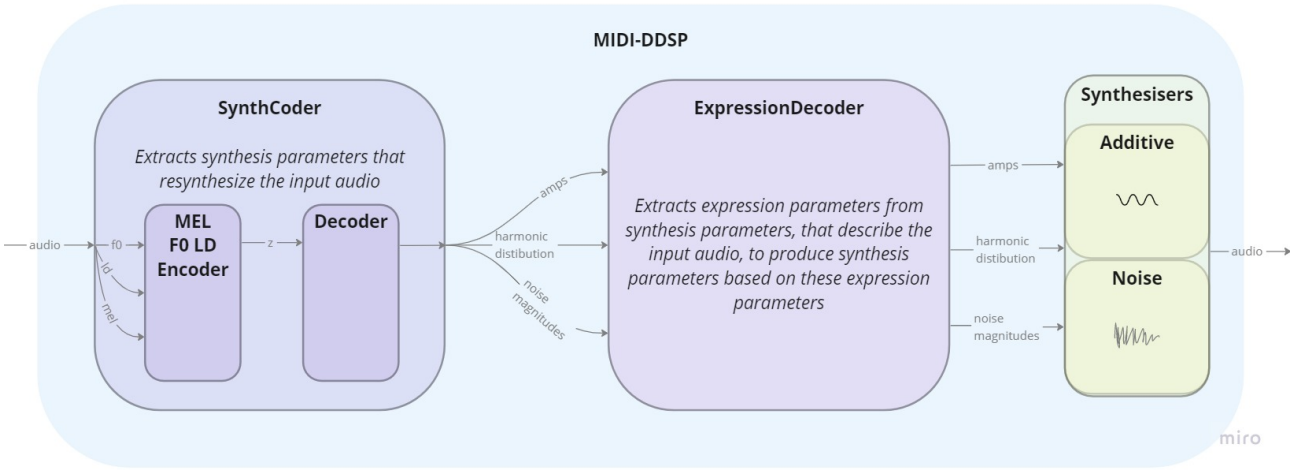


Figure 5: Simplified view of the MIDI-DDSP architecture.

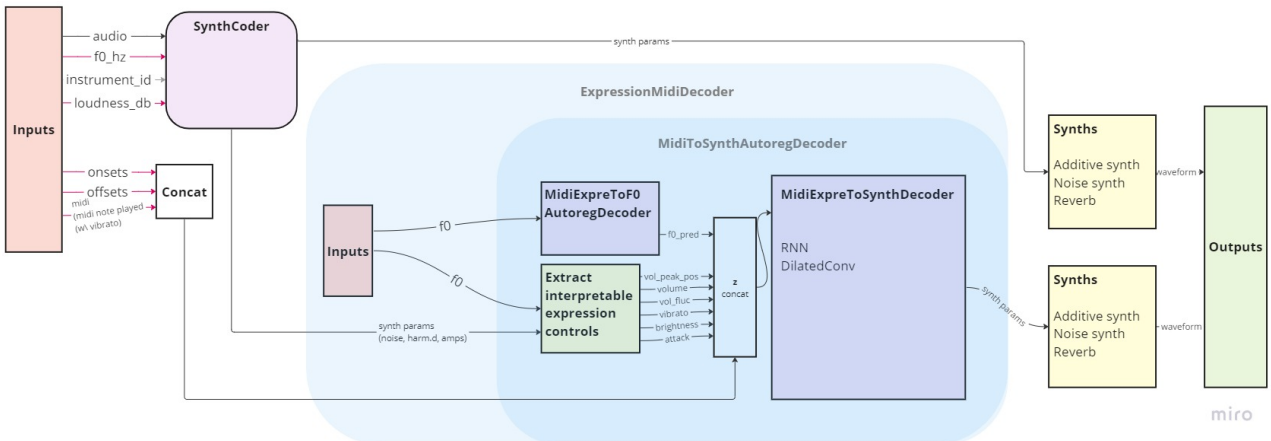


Figure 6: Detailed View of the MIDI-DDSP architecture.

Training and inference MIDI-DDSP model is trained in two stages:

1. Train SynthCoder on the reconstruction MSSL loss [6]. Do not run ExpressionDecoder for now.
2. Freeze SynthCoder and use it in the inference mode to feed its outputs to the ExpressionDecoder which is being trained.

As for the inference, only ExpressionDecoder is used.

3 Evaluation of timbre transfer quality

3.1 Methodology

Multi-scale spectral loss and why it is not suitable

Multi-scale spectral loss (MSSL) is the loss used for training DDSP and MIDI-DDSP's SynthCoder models. [6][20]

$$\mathcal{L}_{\text{MSSL}} = \sum_{i=1}^{[64,128,..,2048]} ||S_i - \hat{S}_i||_1 + \alpha||\log S_i - \log \hat{S}_i||_1 \tag{1}$$

Such loss provides us with information at multiple resolutions, so it is quite useful for signal comparison. However, we cannot use it to measure the quality of timbre transfer because our dataset lacks ground truth timbre transfer samples. And even if we had some (e.g. for oboe $\rightarrow$ violin), we could not simply compare the model's output to the ground truth violin sample. The reason is that there could be multiple waveforms that have roughly same spectral [20] and overall audio quality characteristics, meaning that they are all perceived as the violin playing same notes with similar articulation and overall audio quality. In other words, there is no one correct result in the timbre transfer task.

Mean opinion score and why it is not suitable

Instead, we can employ a listener based score. MOS (mean opinion score) is one of the possible perceptive metrics for the quality of the sound generated by neural networks [31].

The Mean Opinion Score (MOS) formula is given by:

$$MOS = \frac{1}{N}\sum_{i=1}^{N} S_i$$

where:

MOS is the Mean Opinion Score,

N is the total number of ratings,

S_i is the individual rating score for each sample.

However, our experiments showed that it is rather difficult for listeners to give a numeric score indicating the perceptual timbre similarity between two instruments. Furthermore, the 'warm-up effect' was too prominent here and we could not provide them with dozens of samples due to high fatigue, so we had to drop that metric.

Pairwise comparisons

Another option is to employ pairwise comparisons. This method is good because it is relatively easy for participants to compare the pair of samples and pick the best one. Here, the absolute scores are computed from the relative ones.

Question types We have asked three types of questions to the listeners:

1. Which audio sounds more realistic and has better quality?
2. Which audio sounds more like a source instrument of the timbre transfer? (here we are evaluating timbre reconstruction stability)
3. Which audio sounds more like a target instrument of the timbre transfer? (here we areevaluating timbre transfer capabilities)

Survey description When the survey is started, the listener is presented with one question of a randomly generated type. It is accompanied by a pair of randomly selected 4-second audio samples from the test dataset that includes both input and output audios for each model evaluated in this paper. There are 17 input audios in total and they were manually picked from the evaluation dataset to represent various articulations and melodies. So there is $n_{out} \cdot n_{models} + n_{in} = 17 \cdot 10 + 17 = 187$ audio samples in total. The question type is selected with following probabilities: $[0.3, 0.3, 0.4]$ (respective to the list above). Once the listener answered a question, it disappears, and a new question is presented. Listener has no knowledge about the presented audio and its origin. For the timbre similarity questions we also provide a sample of how the instrument sounds in real world. That audio is picked from a training dataset. It is useful as a reference for those who do not know names or sound of rarer musical instruments. Therefore, our survey can be characterized as a randomized and single-blind.

3.2 Tools

Telegram bot

We have developed a modular python telegram bot [22] to conduct such surveys. The code for the bot is available at our github repo.
The bot is built in a modular fashion and it is fairly easy to add new data sources as well as question types. To ensure efficient handling of multiple concurrent read and write requests, we utilize a locally hosted PostgreSQL relational database for data storage [21].

3.3 Data collected

We have collected 1067 pairwise comparisons from 89 listeners. There are 55 out of 55 possible unique unordered pairs of models, samples of which were presented to listeners ($C_n^k = C_{10+1}^2 = 55$).

4 MIDI-DDSP as a timbre transfer architecture

4.1 Hypothesis

Our main hypothesis is that MIDI-DDSP [20] can perform timbre transfer in at least one of two ways:

- If we pass audio recording of some melody, performed on an arbitrary instrument, as an audio input to a model, which was trained on a dataset containing only one (target) instrument. No other modifications required.
- If we pass audio recording of some melody, performed on an arbitrary instrument, as audio input to a model, which was trained on a dataset containing arbitrary instruments, including the target one. No other modifications required.

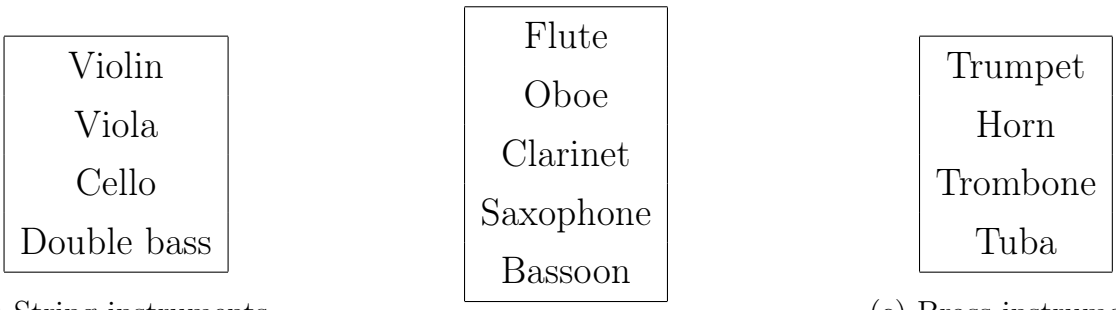
4.2 Experiments description

To test our hypotheses, we have trained two models: 'vn' (trained by us on a violin-only dataset) and 'all' (trained by Google Magenta on a multi-instrument dataset).

model name	batch size	lr	frame size (samples)	epochs	steps	MSSL [20]
all	16	0.0003	64	80	50000	4.9703
vn	16	0.0003	64	235	28669	5.4714

Figure 7: Parameters used for training MIDI-DDSP models.
All the other parameters are equal to default ones in the MIDI-DDSP repo. [13]

Both models were trained on a URMP dataset prepared by Google Magenta [12]. The dataset is annotated with pitch and notes data and consists of 4-second audio samples. URMP dataset in turn "comprises a number of simple multi-instrument musical pieces assembled from coordinated but separately recorded performances of individual tracks" and was created by University of Rochester [29].



(a) String instruments

(b) Woodwind instruments

(c) Brass instruments

Figure 8: All 13 musical instruments that are represented in the URMP dataset.

Dataset	Number of training examples	Number of evaluation examples
'vn'	1938	172
'all'	9982	833

Figure 9: Comparison of 'vn' and 'all' datasets in terms of example count.

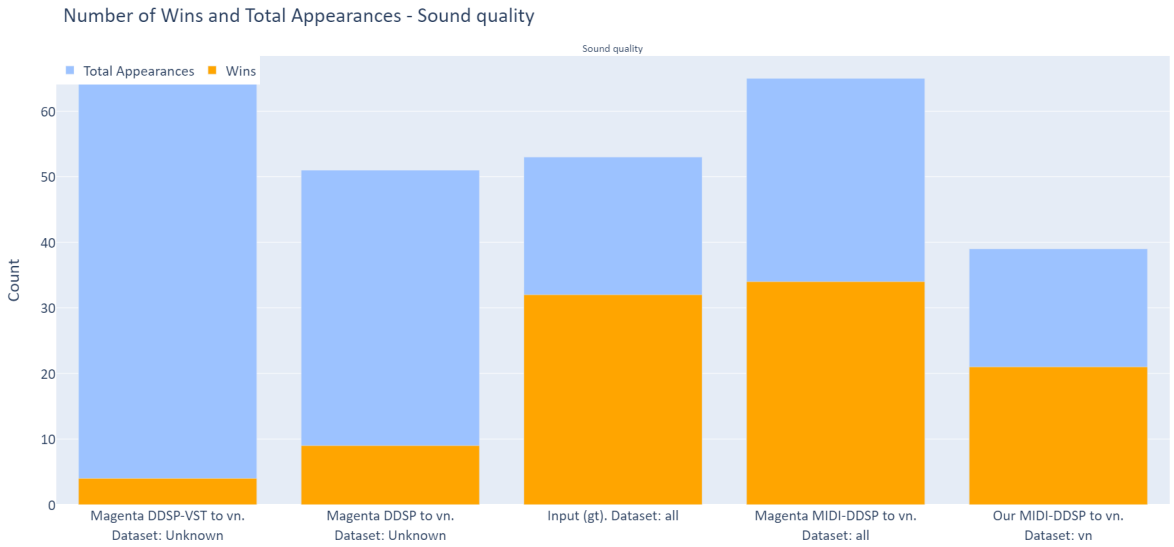
4.3 Results

Our experiments showed that MIDI-DDSP is capable of performing high-quality timbre transfer:

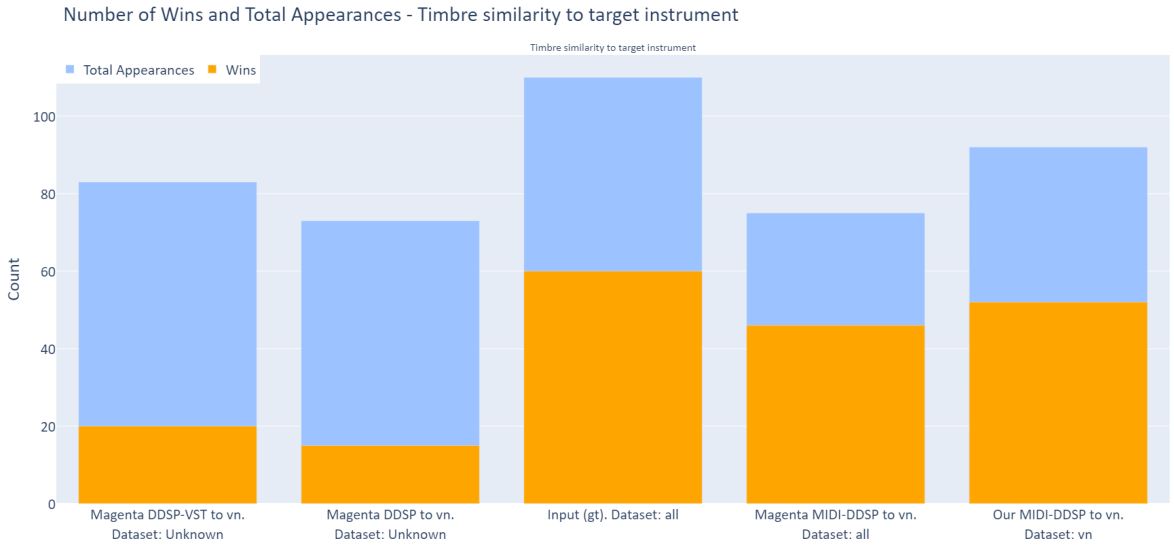
- Multi-instrument model successfully generates the timbre of the passed instrument_id and preserves articulations.
- Single-instrument model generates the timbre of the instrument it was trained on. However, it often ignores note onsets when playing repeated notes. It also required more training steps to achieve same spectral loss compared to a multi-instrument model. [1]

Based on the listeners' scores, MIDI-DDSP is significantly surpassing the baseline DDSP model in both realism, audio quality and timbre similarity to the target instrument (see 10). For a detailed visualization of the outputs from each module and input/output comparison, please refer to Figure 15. See 2 to learn about DDSP and DDST-VST datasets.

[1] The issue with onsets being ignored is less prominent in earlier epochs, e.g. in epoch 100. So it seems to be a sign of overfitting. Our hypothesis is that on small dataset, the model learns the timbre too late, and on that late stages it does overfit in terms of pitch.



(a)

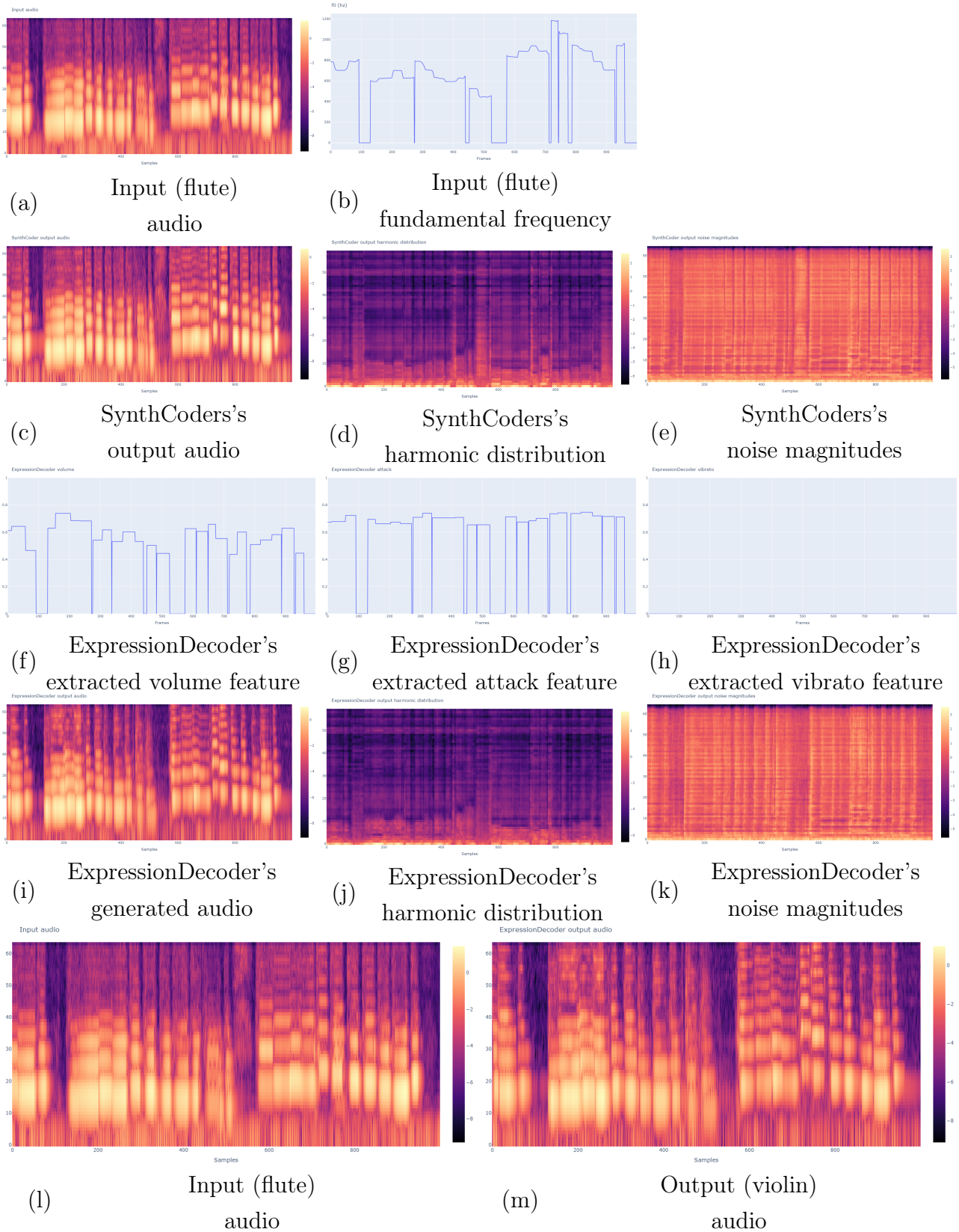


(b)

Figure 10: Number of wins for MIDI-DDSP & DDSP models in the listeners' survey (orange). (a) survey of sound quality, (b) survey of timbre similarity to target (i.e. timbre transfer capabilities).

Our survey shows that MIDI-DDSP is far superior in terms of sound quality and realism compared to original DDSP timbre transfer model: it is nearly as good as the ground truth audios. Regarding the timbre transfer capabilities, MIDI-DDSP exhibits notable improvements as well. Nevertheless, the sound of ground truth samples of the target instrument is generally rated slightly higher compared to results from the audio quality survey. The reason is that latter included violin $\rightarrow$ violin samples as well.

Amongst most notable MIDI-DDSP's audio quality issues are: long unrealistic reverberation tails and rare noises that are perceived as having some phase issues.



(a) Input (flute) audio

(b) Input (flute) fundamental frequency

(c) SynthCoders's output audio

(d) SynthCoders's harmonic distribution

(e) SynthCoders's noise magnitudes

(f) ExpressionDecoder's extracted volume feature

(g) ExpressionDecoder's extracted attack feature

(h) ExpressionDecoder's extracted vibrato feature

(i) ExpressionDecoder's generated audio

(j) ExpressionDecoder's harmonic distribution

(k) ExpressionDecoder's noise magnitudes

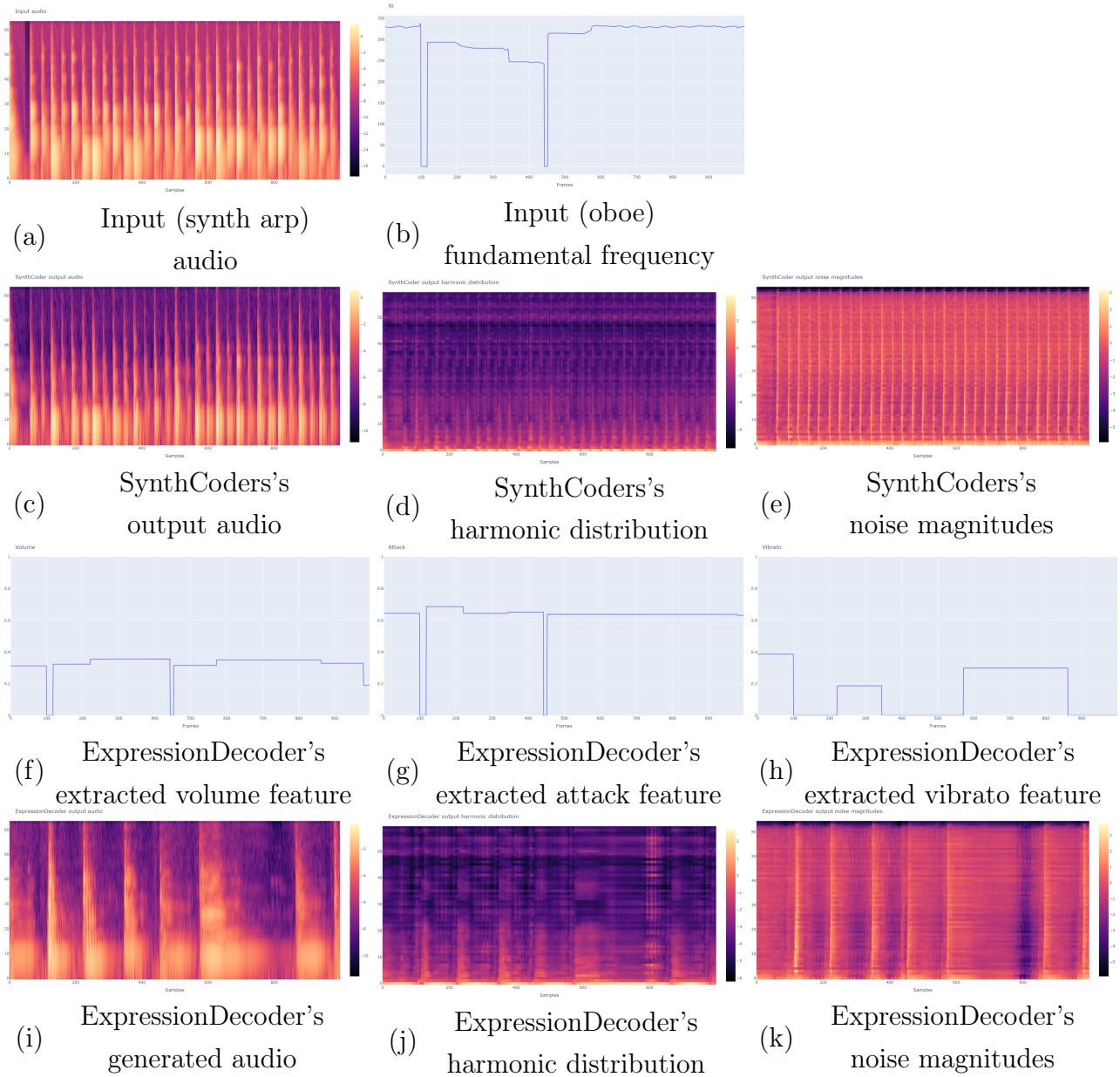
(l) Input (flute) audio

(m) Output (violin) audio

Figure 11: Intermidiate outputs for each module in the 'all' MIDI-DDSP model for flute → violin timbre transfer. The spectrograms are created using Mel-STFT [25] [28] with frame_size=64, n_fft=1024, num_mels=64.

Deliberately mismatching input audio

There is also an interesting property of the MIDI-DDSP architecture. If we take the input batch and replace its audio with some other audio while keeping $f0$, ld and other parameters unchanged, we obtain an output audio with a provided $f0$, but with timbre of target instrument_id. Moreover, the attack, vibrato and rhythm are of the provided audio (see 12 for more details).



(a) Input (synth arp) audio

(b) Input (oboe) fundamental frequency

(c) SynthCoders's output audio

(d) SynthCoders's harmonic distribution

(e) SynthCoders's noise magnitudes

(f) ExpressionDecoder's extracted volume feature

(g) ExpressionDecoder's extracted attack feature

(h) ExpressionDecoder's extracted vibrato feature

(i) ExpressionDecoder's generated audio

(j) ExpressionDecoder's harmonic distribution

(k) ExpressionDecoder's noise magnitudes

Figure 12: Intermidiate outputs for each module in the 'all' MIDI-DDSP model for oboe → violin timbre transfer with input audio of synth arp instead of oboe, whereas $f0$ and ld are unchanged.

The explanation for this is following: as we will show later, SynthCoder is not capable of performing timbre transfer, so the ExpressionDecoder is the module that actually performs timbre transfer and its inputs contain only expression parameters extracted from SynthCoder's synth parameters. It means that the entire timbre transfer process is parameterized solely via expression controls (as opposed to the direct parameterization via the audio or its spectral characteristics). Such a "bottleneck" appears to be the key factor enabling the timbre transfer capabilities of the audio reconstruction neural network.

[2]

[2] The training dataset of DDSP and DDSP-VST models is unknown, but it seems that DDSP used the dataset based on violin recording from MusOpen [4], while the DDSP-VST model was trained on the NSynth [14] dataset.

5 SynthCoder as a timbre transfer architecture and why it fails

5.1 Hypothesis

Considering the notable timbre transfer capabilities of the MIDI-DDSP architecture, it is reasonable to question whether the full model is necessary for the task. Could the SynthCoder alone be enough for it? The rationale is simple: architecture of SynthCoder resembles the original DDSP architecture which had proven its timbre transfer capabilities. Thus, it seems possible that the SynthCoder has timbre transfer capabilities as well. And if it is true, then we could end up with a smaller, simpler and faster architecture for the timbre transfer task.

5.2 Description of experiments

To test this hypothesis we ran the SynthCoder in various configurations alone, without the ExpressionDecoder module. The evaluation of these models followed the same methodology as the evaluation of the MIDI-DDSP models (section 3.1).

Name	Dataset	Target	Description	Epochs	Steps	MSSL
SynthCoder	URMP all	oboe	Evaluation of the timbre transfer to oboe.	81	10,000	4.1887
SynthCoder	URMP all	same	Evaluation of resynthesis.	81	10,000	4.1887
SynthCoder	URMP all	violin	Evaluation of the timbre transfer to violin.	81	10,000	4.1887
SynthCoder	URMP vn	same	Evaluation of resynthesis.	45	5,489	4.4246
SynthCoder	URMP vn	violin	Evaluation of the timbre transfer to violin.	45	5,489	4.4246
SynthCoder without mel	URMP all	violin	Ablation study of SynthCoder.	300	249,599	4.4014

Figure 13: List of all SynthCoder models from our experiments

5.3 Results

Our experiments show that SynthCoder architecture is good at both resynthesizing input audio and generalizing its resynthesis capabilities to unseen musical instruments, which means that it is bad at timbre transfer. It is shown clearly in the results of the survey (see 14).

Resynthesis and timbre transfer evaluation

Based on the listeners' scores, SynthCoders are best at resynthesis amongst all the other models except (meaning that it is better than DDSP and even MIDI-DDSP) which also aligns with its lower MSSL values (see 14b, 13 and 7). As for the audio quality, it exhibits variable results, ranging from best to mediocre (depending on the instrument) (see 14a). However, it is bad at timbre transfer (see 14c).

The other key finding is that the model, trained on the violin-only dataset, is still able to reconstruct other instruments (meaning that it generalizes to other instruments). However, the audio quality for these other instruments is rated lower than in the multi-instrument SynthCoder models. Finally, they also fail at performing timbre transfer. But what is the reason for such behavior?

Figure 14: Number of wins for SynthCoder, MIDI-DDSP & DDSP models per question type (orange). (a) is for sound quality, (b) is for timbre similarity to source (i.e. audio reconstruction capabilities), (c) is for timbre similarity to target (i.e. timbre transfer capabilities).

Ablation study and comparison to original DDSP architecture

We ran an ablation study to figure out why SynthCoder is not capable of performing timbre transfer. Our main hypothesis was that SynthCoder receives quite a detailed representation of the input audio, as compared to original DDSP architecture (which only possesses MFCC optionally extracted from the input audio) and to MIDI-DDSP's ExpressionDecoder architecture (which is parameterized by expression parameters only). So we modified the original SynthCoder architecture by removing mel spectrogram extractor and a stack of convolutions that processes it so that it becomes parameterized only by $f0$, ld and $instrument\_id$. The model was trained on the 'all' dataset to exclude the factor of worsened reconstruction generalizability which results in the higher timbre transfer capabilities. Our goal is to check whether the model can handle multiple instruments and perform accurate timbre transfer to each of them.

Based on the listener's scores, such architecture is indeed better at performing timbre transfer compared to other SynthCoder models (see 14c). It is even close to MIDI-DDSP model while having much less parameters ($1,380,734$ vs $11,868,933$ in the full MIDI-DDSP model). It is also worse then all the other SynthCoder models in terms of audio reconstruction, even though it is still better then DDSP, DDSP-VST and both MIDI-DDSP models. As for the sound quality, it is worse than all the other SynthCoder models, while being better than DDSP and DDSP-VST models (see 14c).

Drawing from these results, we can conclude that detailed representation of input audio hinders timbre transfer capabilities of the model.

6 Real time applications

While having a high-quality timbre transfer model is advantageous, its usefulness may be limited if musicians cannot apply it in real-time scenarios. And for such scenarios it must exhibit low latency, which can also speed up standard workflows. So aiming for real-time applications can yield broader benefits overall.

As was shown in related work, there is a DDSP-VST [11] real time timbre transfer plugin developed by Google Magenta. However, it performs worse then classic DDSP model (see 14a). On top of existing DDSP issues it also has greater problems with overall timbre quality and pitch detection (f0 swipes in particular).

Hopefully, due to similarities between MIDI-DDSP and DDSP architectures it seems reasonable to reuse existing DDSP-VST codebase while upgrading the synthesis prediction model that it runs. In our case, such model is MIDI-DDSP.

6.1 How to adapt MIDI-DDSP model to a DDSP-VST pipeline

Based on the key differences between the DDSP-VST inference pipeline and DDSP model itself (see 2.2), we conclude that in order to replace a DDSP PredictControlsModel with the MIDI-DDSP model without modifying the feature extraction model, following steps have to be performed:

1. Convert a model to the TFLite format.
2. Make the model work with a frame size of 1024.
3. Handle state for the RNNs explicitly.
4. Optimize the model as much as possible.

6.2 Conversion to TFLite and proof of concept

We made the SynthCoder fully convertible to the TFLite format and MIDI-DDSP is now partially convertible. The difficulty was that a lot of non-TensorFlow [17] operations were used in the MIDI-DDSP codebase and some of the TensorFlow operations had no exact equivalent in the TFLite environment (such as RFFT2D). You can find the adapted code at our github repo.

However, we did not manage to run the ExpressionDecoder as a TFLite model because it required enabling extended set of TFLite operations via the Flex Delegate [23] which was corrupting the output .vst3 for some reason.

Nevertheless, we have shown that it is possible to run SynthCoder in real time.

6.3 MIDI-DDSP and increased frame size

The next step is to make our SynthCoder model work with a frame size of 1024. However, all of our SynthCoder models are trained with a frame size of 64. Fortunately, we can feed an arbitrary number of frames to the model.

Hypothesis

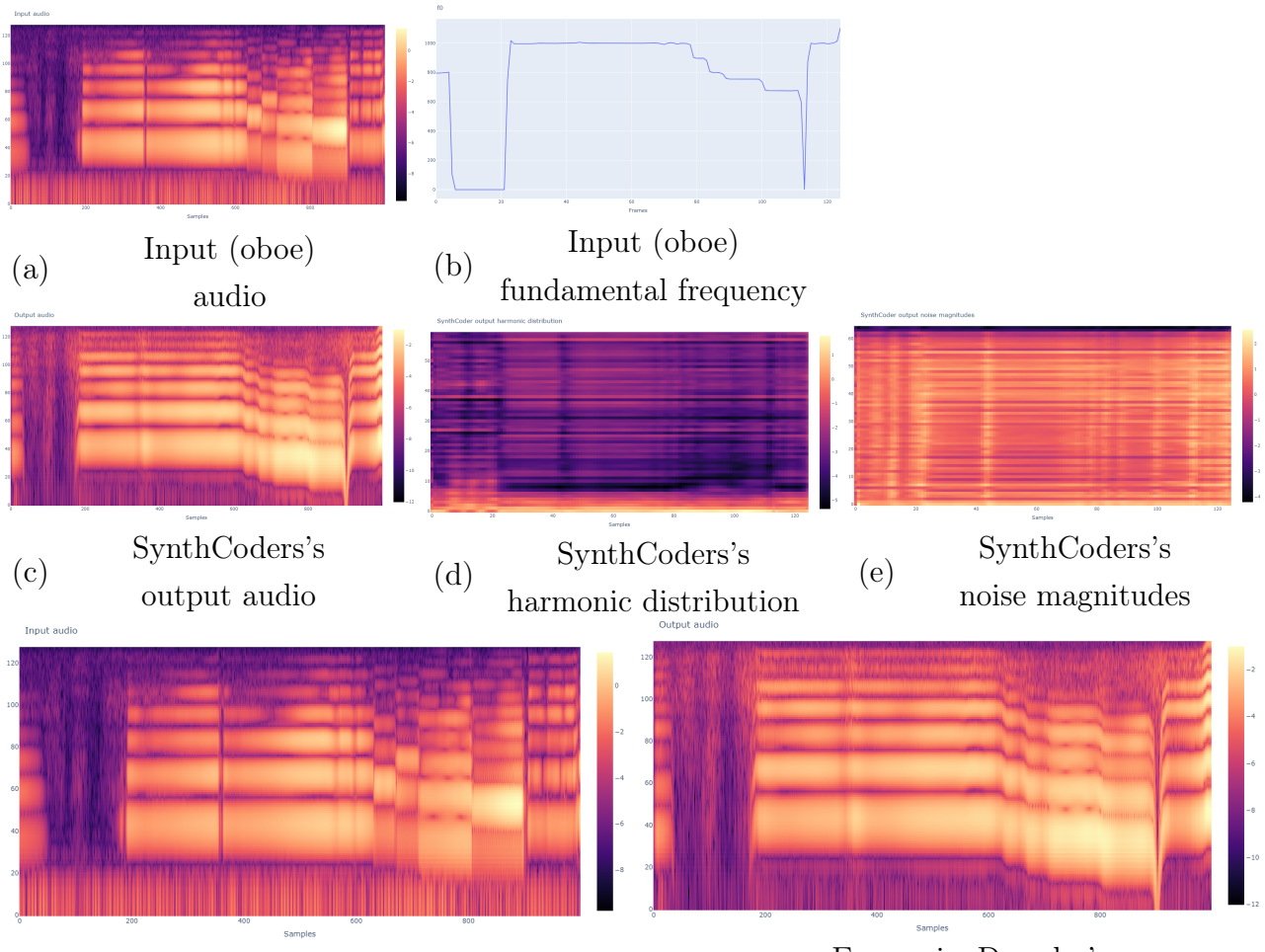
Our hypothesis is that we can feed 62 frames for a 4-second audio with a frame size of 1024 instead of feeding 1000 frames for a same 4-second audio with a frame size of 64 and get comparable audio quality to the model with a frame size of 64.

Experiments

To test this hypothesis, we implemented a simple input downsampler. As far as we experiment with SynthCoder, the inputs are constrained to $audio$, $f0$, ld and $instrument\_id$, which can be downsampled using by taking a mean value over each k timestamps, where k is a downsampling factor. We use the 'vn' SynthCoder from the 'vn' MIDI-DDSP model.

Results

Unfortunately, the audio realism, instrument articulations and overall timbre characteristics degrade significantly with greater downsampling factors. Even when $k = 8$ the model still performs bad. It has unrealistic f0 slides (as in the original DDSP PredictControlsModel) and all of the aforementioned issues (please see 15g).



(a) Input (oboe) audio

(b) Input (oboe) fundamental frequency

(c) SynthCoders's output audio

(d) SynthCoders's harmonic distribution

(e) SynthCoders's noise magnitudes

(f) Input (oboe) audio

(g) ExpressionDecoder's generated (violin) audio

Figure 15: Outputs of the 'vn' SynthCoder model for oboe → violin timbre transfer with inputs downsampled 8 times. The spectrograms are created using MelSTFT [25] [28] with frame_size=64, n_fft=1024, num_mels=128.

Summary

The study explores the capabilities of the MIDI-DDSP architecture. We have conducted a listener's survey and demonstrates MIDI-DDSP's significant advancements in audio quality and realism compared to DDSP. We have also shown that MIDI-DDSP is superior to the DDSP architecture not only in the task of audio reconstruction, but in the task of timbre transfer as well. Besides that, we have found an interesting property of mismatching input audio. Moreover, we have shown that SynthCoder succeeds at generalizing to unseen instruments but fails at timbre transfer task. We have explained such results by performing an ablation study which showed that detailed representation of input audio hinders timbre transfer capabilities of the models. Through the removal of mel spectrogram extraction and processing layers, we have successfully enabled SynthCoder to perform timbre transfer and attain results comparable to MIDI-DDSP. The study also investigates real time applications of the MIDI-DDSP model and provides a modified MIDI-DDSP source code that enables conversion of the SynthCoder to the TFLite format. As a proof of concept, we have shown that it is possible to run SynthCoder in real time. Nevertheless, increasing the frame size in the existing DDSP-VST inference pipeline, as demonstrated in our research, leads to a significant degradation in output quality. In this way, out findings contribute to practical applications in music production and provide valuable insights into the application of deep learning techniques in the audio domain.

References

[1] Ambient music genre overview | allmusic. AllMusic. URL: https://www.allmusic.com/style/ambient-ma0000002424 (date of access: 10.05.2023).

[2] Anderson, Julian. 2000. "A Provisional History of Spectral Music". Contemporary Music Review 19, no. 2 ("Spectral Music: History and Techniques"): p. 7–22.

[3] Audio Plugins ranked by popularity at KVR Audio. KVR Audio. URL: https://www.kvraudio.com/plugin-ranks.php (date of access: 12.05.2023).

[4] Bach Violin Partita no. 1, BWV 1002 - Download free sheet music. Free Sheet Music, Royalty Free & Public Domain Music | Musopen. URL: https://musopen.org/music/13574-violin-partita-no-1-bwv-1002/ (date of access: 08.05.2023).

[5] CREPE: a convolutional representation for pitch estimation / J. W. Kim et al. arXiv.org. URL: https://arxiv.org/abs/1802.06182 (date of access: 10.05.2023).

[6] DDSP: differentiable digital signal processing / J. Engel et al. arXiv.org. URL: https://arxiv.org/abs/2001.04643 (date of access: 14.05.2023).

[7] Dieleman S. Generating music in the waveform domain. Sander Dieleman. URL: https://sander.ai/2020/03/24/audio-generation.html (date of access: 14.05.2023).

[8] DiffWave: a versatile diffusion model for audio synthesis / Z. Kong et al. arXiv.org. URL: https://arxiv.org/abs/2009.09761 (date of access: 14.05.2023).

[9] Efficient neural audio synthesis / N. Kalchbrenner et al. arXiv.org. URL: https://arxiv.org/abs/1802.08435 (date of access: 13.05.2023).

[10] GANSynth: adversarial neural fudio synthesis / J. Engel et al. arXiv.org. URL: https://arxiv.org/abs/1902.08710 (date of access: 22.05.2023).

[11] Google Magenta. DDSP-VST repository. GitHub. URL: https://github.com/magenta/ddsp-vst (date of access: 11.05.2023).

[12] Google Magenta. Magentadata repository. Google cloud storage. URL: gs://magentadata/datasets/urmp/urmp_20210324 (date of access: 13.05.2023).

[13] Google Magenta. Midi-ddsp repository. GitHub. URL: https://github.com/magenta/midi-ddsp/blob/d7af42704a63b47267ae6a1bc0fee1ed7dc5 (date of access: 11.05.2023).

[14] Google Magenta. NSynth dataset. Magenta. URL: https://magenta.tensorflow.org/datasets/nsynth (date of access: 11.05.2023).

[15] Griffin D., Lim J. Signal estimation from modified short-time Fourier transform. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/1164317 (date of access: 11.05.2023).

[16] How is inference speed optimized? GitHub. URL: https://github.com/magenta/ddsp-vst/issues/14 (date of access: 11.05.2023).

[17] Introduction to TensorFlow. TensorFlow. URL: https://www.tensorflow.org/learn (date of access: 11.05.2023).

[18] Jukebox: a generative model for music / P. Dhariwal et al. arXiv.org. URL: https://arxiv.org/abs/2005.00341 (date of access: 08.05.2023).

[19] Learning phrase representations using RNN encoder-decoder for statistical machine translation / K. Cho et al. arXiv.org. URL: https://arxiv.org/abs/1406.1078 (date of access: 10.05.2023).

[20] MIDI-DDSP: detailed control of musical performance via hierarchical modeling / Y. Wu et al. arXiv.org. URL: https://arxiv.org/abs/2112.09312 (date of access: 14.05.2023).

[21] PostgreSQL. PostgreSQL. URL: https://www.postgresql.org/ (date of access: 08.05.2023).

[22] Python-telegram-bot. python-telegram-bot. URL: https://python-telegram-bot.org/ (date of access: 14.05.2023).

[23] Select TensorFlow operators | TensorFlow Lite. TensorFlow. URL: https://www.tensorflow.org/lite/guide/ops_select (date of access: 12.05.2023).

[24] Steinberg. VST3 developer portal. Steinmergmedia. URL: https://steinbergmedia.github.io/vst3_dev_portal/pages/index.html (date of access: 11.05.2023).

[25] Stevens S. S., Volkmann J., Newman E. B. A scale for the measurement of the psychological magnitude pitch. AIP Publishing. URL: https://doi.org/10.1121/1.1915893 (date of access: 11.05.2023).

[26] TensorFlow Lite | ml for mobile and edge devices. TensorFlow. URL: https://www.tensorflow.org/lite (date of access: 12.05.2023).

[27] TimbreTron: a wavenet(cyclegan(cqt(audio))) pipeline for musical timbre transfer / S. Huang et al. arXiv.org. URL: https://arxiv.org/abs/1811.09620 (date of access: 12.05.2023).

[28] tf.signal.stft | TensorFlow v2.12.0. TensorFlow. URL: https://www.tensorflow.org/api_docs/python/tf/signal/stft (date of access: 13.05.2023).

[29] URMP dataset. University of Rochester Blogs. URL: https://labsites.rochester.edu/air/projects/URMP.html (date of access: 12.05.2023).

[30] Virtual musical instruments market. Market Research Reports, Business Consulting | In-depth Insights | TMR. URL: https://www.transparencymarketresearch.com/virtual-musical-instruments-market.html (date of access: 11.05.2023).

[31] WaveNet: a generative model for raw audio / A. van den Oord et al. arXiv.org. URL: https://arxiv.org/abs/1609.03499 (date of access: 12.05.2023).