PTP: Partiture to Performance using Speech Synthesis with Encoder-Decoder models

Stan Ruijters

Supervisors:
Joost Broekens & Fons Verbeek

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/06/2024

Abstract

The structure of expressive speech shares a lot of similarities with music. With the performance of recent neural network-based speech synthesis architectures, it is interesting to investigate if such architectures generalize, and can be used to generate expressive music. Here, we investigate if existing text-to-speech based on encoder-decoders can be used convert a musical score to an expressive violin performance, in a process named "Partiture to Performance (PTP)". Tacotron 2 and WaveGlow are used as PTP model and vocoder respectively, two architectures intended for text-to-speech synthesis purposes. In addition, the PTXT grammar is proposed, which allows for a detailed and concise textual representation of sheet music. Results show that the model can output music with expressive features, although the expressiveness is more limited than seen in the ground truth. Nevertheless, the audio is more accurate to human recording than a reference generated using a digital soundfont, where each note is recorded in isolation and stitched together. Due to this, combined with the limitations regarding dataset size and use of early-stage architectures, it is concluded to be possible to generalize encoder-decoder text-to-speech architectures and use them for PTP.

Contents

1 Introduction

The artificial generation of expressive audio is a highly complex task. Several key features of the source audio signal must be captured to perform accurate synthesis using machine learning. For instance, speech can vary greatly depending on the person talking, the emotion they are trying to convey and in what context they are being recorded. On a smaller level, this can be quantified by features such as intonation, timbre and pitch variation, among numerous others [Kin03]. These features can be defined on varying scales; from short-term features that capture moment-to-moment variations in the audio signal, to long-term features that encapsulate broader context. Accurately representing these diverse characteristics poses a substantial challenge for speech synthesis systems.

Despite this, recent years have seen significant improvements in the performance of neural networks regarding speech synthesis tasks. Modern text-to-speech systems allow for generating natural-sounding voices with remarkable fidelity. Most of these systems use an encoder-decoder architecture, which has proven highly effective for converting text to a voice [TQSL21].

Similar to speech, music contains several key features that define its expressiveness as well. For example, by introducing variations in tempo, dynamics and playing style, a performer can alter how notes are played to add emotion to their interpretation of a piece. Here, the scale of impact will vary from individual notes to sequences of multiple measures. In the same way, numerous other types of audio can be modelled with attention to their unique features and scales of expressiveness. [KC90]

Given the similarities between speech and music, the question arises whether the architectures for text-to-speech synthesis tasks generalize to tasks such as generating expressive sequences of music. This thesis aims to investigate this question, by training a model to convert a piece of sheet music to expressive audio, through a process coined ”Partiture to Performance (PTP)”. This is done using an existing speech synthesis encoder-decoder architecture. To accomplish this, the custom ”Partiture-TXT (PTXT)” notation is proposed here as a textual representation of sheet music, providing concise and detailed input for the encoder. To properly use PTXT, several adjustments need to be made to the encoder, particularly regarding the text normalization used for speech synthesis and the encoding of text during preprocessing. The PTP model is trained on a dataset consisting of classical violin music. A vocoder needs to be trained from scratch for this as well, as one trained on speech does not produce intelligible audio when doing inference with violin music. To evaluate the expressiveness of the model, its output is compared to human recordings, as well as digital representations. Our results show that the PTP process is successful, indicating that encoder-decoder TTS architectures can generalize to other tasks including music generation.

To provide a clear background, Section 2 will discuss the musical theory used in this thesis. Additionally, Section 3 will show related research done in this field, which has inspired and supported this study. Section 4 highlights the process of training the desired model, by showcasing the used architecture, dataset preprocessing and training setup, and discussing the grammar specifically constructed for this task. The results of the model output are documented in Section 5. Section 6 discusses the limitations of the research, and provides a final evaluation of the viability of general expressive audio generation. The thesis is concluded in Section 7.

2 Musical notation background

A musical sequence can be split into measures, delineated by a measure bar. Each measure has an associated time signature and tempo. The time signature indicates how many beats are in each measure and how long a beat is in terms of note duration. For instance, a 3/4 time signature means there are three beats per measure, with each beat having the length of a quarter (1/4) note. The tempo is typically defined as beats per minute (BPM) and specifies how fast the music is played. A measure is constructed using notes, that signify what the performer should play at which time. Each note consists of three main elements:

1. A relative pitch, signifying the relative frequency of the note. Standard musical notation denotes pitch using 7 letters: A-G. Additionally, accidentals can be added to these letters that increase or decrease the pitch by half a step. In total, the combination of letters and accidentals allows for 12 different notes.

2. An octave number, that indicates the frequency domain of the pitch. An octave is defined as the set of notes from one pitch to the next of the same name. If the octave number of a pitch is incremented by 1, its frequency doubles. Often, the word pitch is used to encompass both the octave number and the relative pitch of a note. For the remainder of this thesis, the use of the word will refer to this definition, unless mentioned otherwise.

3. The duration, which represents the length of time a note is held. Note duration is a dyadic rational proportional to the time signature, meaning it can be written as a fraction whose denominator is a power of two. Examples include whole notes, half notes, quarter notes, and so forth. Notes can contain one or more dots, which add half of the original note length to its duration. For instance, a quarter note with two dots can be seen as a note with a length of $1/4 + 1/8 + 1/16 = 7/16$. Additionally, two notes of similar pitch can be played as one note if they are connected using a tie. In this case, the second note is not played separately but instead adds its duration to the initial note.

Notes can have various annotations that alter their playing style. Notable to this thesis is staccato, which indicates a note should be held for only a short time, often introducing a moment of silence before the next note is played. Similarly, notes can be played legato, where notes are held until the transition to the next note, giving a more connected feel.

In music theory, playing multiple notes simultaneously is defined as a chord. Alternatively, chords can also be played as an arpeggio, where the notes are played quickly in ascending or descending order. Times where no notes are played are classified as a rest. The duration of the rest depends on the consecutive time no note is played, and follows the same duration metric as notes.

Finally, measures can be repeated when repeat brackets are added to measure bars. When the piece reaches the closing repeat bracket for the first time, the piece will repeat all measures starting from the opening repeat bracket, or from the first measure if no starting bracket was specified. Additionally, it is possible to have different final measures for each repeat, by using first and second ending brackets, which indicate alternative endings for the repeated section.

3 Related Work

The systems used for modern neural network-based speech synthesis tasks require a pair of deep learning models. An acoustic model predicts an intermediate output from phoneme and acoustic prompts, and a vocoder converts this output to a waveform representing human-like speech [TQSL21]. Wang et al. were pioneers of this technique, when they demonstrated the Tacotron architecture in 2017 [WSRS+17]. The paper describes a sequence-to-sequence acoustic model that converts text to a spectrogram. Then, the Griffin-Lim algorithm [GL84] is used as a non-neural vocoder to reconstruct a waveform. The next year, the architecture would be improved and published as Tacotron 2 [SPW+18], using an encoder-decoder system as the acoustic model and switching to a neural network-based vocoder approach. Since then, the paired model approach for neural network-based speech synthesis has been improved, with newer models such as TransformerTTS [LLL+19] showing higher performance than Tacotron 2, and FastSpeech [RRT+19] providing significantly faster inference.

Regarding expressive music generation, early research was mostly focused on using statistics to construct musical sequences. Context models were used for this, which utilize a frequency database of musical sequences to compute the most likely continuation. This is shown by SONG/3, a model proposed by Conklin and Witten in 2003 [CW03]. With recent breakthroughs through neural approaches, these systems were significantly improved. Notable examples include Music Transformer [HVU+18], a model that significantly improved on capturing long-term dependencies in music, and MuseNet [PSR20], capable of generating coherent and more expressive music than its predecessors. Compared to these, the PTP model shows slightly lower performance in expressiveness, in favour of being built on a generalized architecture.

One of the first studies to contemplate the idea of using text-to-speech synthesis models for other purposes was published in 2016. Here, Oord et al. proposed WaveNet [ODZ+16], a text-to-speech synthesis model capable of generating speech directly from linguistic features. In the same paper, they also showed the model had viability in converting text to expressive music, showing examples of the generation of synthesized piano pieces. The idea was further explored in 2022, when Dong et al. [DZBKM22] showed how a deep learning model can be trained to convert sheet music to audio, using recent developments in the speech synthesis field. The paper proposed an encoder-decoder system with a mel-spectrogram as an intermediate output, with an architecture similar to what is seen in modern text-to-speech (TTS) systems. Additionally, it highlights how music conversion introduces issues with aligning notes to audio, due to the variation in possible note length. A separate model is required to convert the sheet music to its aligned counterpart.

4 Methodology

4.1 Audio prediction system

As mentioned in Section 3, traditional neural network-based approaches for converting text to speech are often composed of two major components. First, a model processes the input text and predicts some intermediate output. Often, this intermediate output is a low-level feature representation of the final audio output. Common examples include spectrograms [DZBKM22], audio codecs [WC+23] or other acoustic feature representations such as mel-frequency cepstral coefficients [ATKR20]. In the case of speech synthesis, audio samples are provided as additional input, so that the model can adapt the output to resemble the voice of the speaker. The output of the first model is then forwarded to a vocoder, which converts the intermediate output to a waveform, transforming it into actual audio.

By splitting the process into two separate tasks, each model can focus on performing a lower-level prediction task. This increases fidelity significantly compared to a system where one model predicts the audio directly from the input. Note that splitting the process into more than two parts would start decreasing performance again, as each added prediction task will add small inaccuracies to the output that add up over time. Therefore, the best practice is to split the high-level system into as few low-level tasks as possible[TQSL21].

4.1.1 Intermediate output: Mel spectrogram

As an intermediate output, the system will generate a mel-spectrogram. This is a low-level representation of the target audio that is efficiently computable. In addition, it emphasizes highlighting features that are important in human auditory perception, such as pitch, timbre and intonation [DM80].

To create a spectrogram from audio, sampled audio data is converted from the time domain to the frequency domain using a Short-Time Fourier Transform (STFT)[AR77]. Audio is segmented into overlapping windows, with the hop length determining the amount of overlap. Each window $x(t)$ is first multiplied by a window function $w(t)$, to minimize spectral leakage. The modified windows $x_w(t)$ are then converted to the frequency domain using a Discrete Fourier Transform (DFT), with N as the filter length:

$$x_w(t) = x(t) * w(t) \tag{1}$$

$$X(f) = \sum_{t=0}^{N-1} x_w(t) * exp(-i2\pi ft) \tag{2}$$

Note that this transform is not reversible, as the windowing process loses the phase information of the signal [AR77]. Therefore, we require a vocoder to convert a spectrogram back to a waveform, by predicting the lost phase information. After the DFT is calculated, the spectrogram is constructed by combining each of the transformed windows.

After the STFT, a mel-spectrogram can be generated by transforming the frequency axis of the traditional spectrogram from the Hertz scale to the Mel scale. This scale distances different pitches akin to how humans perceive the auditory distance of those pitches. More specifically, to convert a frequency f_h from the Hertz scale to a frequency f_m in the Mel scale, the following formula can be used [DM80]:

$$f_m = 2595 * \log_{10}(1 + \frac{f_h}{700}) \tag{3}$$

The logarithmic relation means that lower frequencies will be distanced further from each other in the Mel scale than the Hertz scale, and higher frequencies will be closer together. The most important acoustic details are apparent at lower frequencies, so emphasising these more is important. In speech, higher frequencies are often the result of noise, so losing accuracy there is not problematic. By emphasizing the acoustic elements important for humans, the system will be able to focus on improving accuracy in those parts, thereby generating more pleasing audio.

4.1.2 Acoustic model: Tacotron 2

The acoustic model used for the first part of the PTP process is trained on the Tacotron 2 text-to-speech synthesis architecture [SPW+18]. This is an encoder-decoder system with an attention layer, capable of converting input text to a mel-spectrogram representing the predicted speech.

The encoder layer of the system creates a sequence of hidden features from the input text. The character sequence obtained from the input text is converted to an embedding, to represent its syntactic information on a better level for further processing. The embedding is used as input for the first of three consecutive convolutional layers, each taking the output of the previous layer as input. The main function of this stack is to model the context of the input over a longer time frame. The output of the final convolutional layer is used by a single bi-directional LSTM layer, that generates the hidden feature sequence [SPW+18].

The attention layer creates a context vector from a weighted sum of the encoder outputs. This is done by using location-sensitive attention [CBS+15], which reduces the likelihood of the model repeating any sub-sequences. This behaviour needs to be avoided, as speech never repeats any characters or phonemes from previous parts of the sequence. Thus, the model should not look back at previous characters as much, whilst keeping the greater context relevant. This holds for music as well, as a performance follows sheet music similarly to how speech follows text. The context vector is forwarded to the decoder and is updated each time step using the output of previous decoder time steps [SPW+18].

The decoder layer consists of three main parts, with each time step of the decoder depending on the output of the previous time step. The first is dependent on the output of the attention layer. First, a pre-net of two fully connected feedforward layers processes the input, to aid in learning attention by acting as an information bottleneck. Combined with the output from the attention layer, two one-directional LSTM layers predict a representation of the current frame of the mel-spectrogram. By projecting this representation using a linear transform, the actual mel-spectrogram frame is generated. Finally, this frame is passed to a post-net consisting of 5 convolutional layers, predicting minor additions to the frame to improve its overall fidelity. The next decoder time step uses the output of the linear transform, without the post-net additions, as these minor details are unwanted information for future frame prediction [SPW+18].

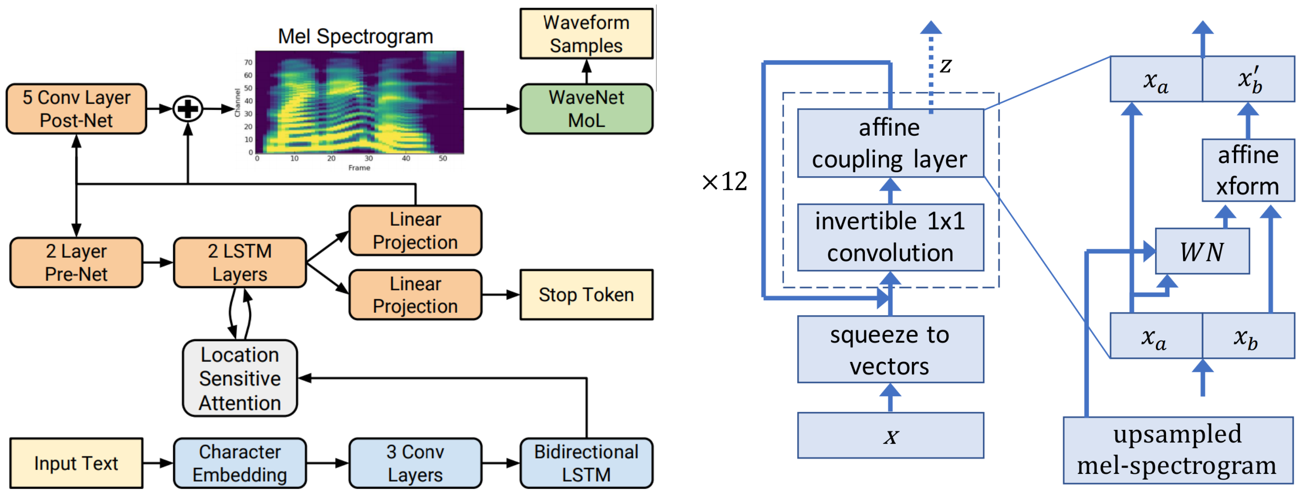
A diagram of the full Tacotron 2 architecture is shown in Figure 1a. To properly show the viability of generalizing this model to other types of audio, PTP training was done with the same hyperparameters used for the creation of an accurate speech prediction model [SPW+18]. This configuration can be found in Appendix A.

4.1.3 Vocoder: WaveGlow

After the full mel-spectrogram of the character sequence is predicted, it is forwarded to a vocoder to be converted to a waveform. For this thesis, a vocoder was trained on the WaveGlow architecture. The main idea of WaveGlow is to create a model of the audio distribution corresponding to the generated mel-spectrogram, by taking samples from a Gaussian distribution and transforming them to samples of the desired one using the model [PVC18].

Each forward pass through the network during inference is done using vectors of 8 audio samples, an operation depicted by the authors as squeezing. To map this set of vectors to the desired audio distribution, a series of invertible transformations defined as a flow are used. Each flow consists of a convolutional layer followed by an affine coupling layer. This process must be invertible, as the process for inference is the reverse of the training process, where samples are instead taken from the desired audio distribution and are mapped to a Gaussian.

The affine coupling layer splits the input into two parts. One half is put through a transformation conditioned on the mel-spectrogram. The transformed channel is then used to scale the other half of the input. By doing this, the model can learn a transformation that depends on the structure of the first half of the input, which helps the model learn more complex dependencies in the data. As channels in one half can not modify each other, a convolutional layer is added before the affine coupling layer to mix information between channels. This process is repeated a total of 12 times and generates an accurate representation of the audio given the initial mel-spectrogram. The full architecture is shown in Figure 1b. As with Tacotron 2, the configuration of the vocoder has been kept the same for the PTP process, to properly show the generalizability of the entire system. The configuration can be found in Appendix A.



(a) Tacotron 2 architecture

(b) Waveglow architecture

Figure 1: Full architecture of the text-to-speech synthesis system, consisting of Tacotron 2 as the acoustic model and WaveGlow as the vocoder.

4.2 Data preprocessing

To provide proper input to the system, the musical score needs to be converted to text, so that it can be processed by Tacotron 2. The sheet music used for this is stored digitally in MusicXML (MXL) files, that contain detailed information about each note in a piece. A parser was created to read MXL file data using Music21 and convert it to a character sequence. This sequence follows a custom syntax, to make the input clear and concise for the model. In addition, several other preprocessing tasks have to be taken into account.

4.2.1 PTXT notation

The "Partiture-TXT" (PTXT) format was created to convert all meaningful information of a piece to a text format as detailed as possible, in such a way that the input remains as simple as possible for encoder-decoder processing. Generally, this includes using a minimal number of characters to document a musical sequence and ensuring as many characters as possible have a unique definition, to avoid any unnecessary confusion for the model. PTXT follows the general structure of musical notation, making slight changes to reduce ambiguity.

PTXT can be read like a sequence of linguistic sentences. Similarly to a musical sequence, a PTXT sequence can be split into measures that are delineated by a closing square bracket, signifying the end of each "sentence". Each measure starts by signifying its BPM, using decimal digits. Elements, those being notes, chords and rests, are then noted by the order in which they appear in the sheet music. An element can be seen as a word in the sentence. It consists of a relative pitch, octave number and duration characters, Multiple elements are separated by a blank space. Chords are structured as a combination of notes, separated by a comma.

To reduce the number of characters used for relative pitch notation, PTXT avoids a system using accidentals. Instead, as there are 12 total relative pitches within an octave, the format uses the capital letters A-L to represent each unique relative pitch in order of frequency. The traditional notation with accidentals is then translated to this single-character system. The translation system can be seen in Appendix B. Note that multiple unique relative pitches in the traditional system can get translated to the same character in the PTXT notation. For instance, an E-flat (E&) and D-sharp (D#) would both be translated to a G, and an F-double-sharp (F##), A-double-flat (A&&) and G would all be translated to a K. Thus, this translation loses some musical information. However, this is not a problem for an encoder-decoder model, as all equivalent notes should produce the same sound. Therefore, they can be represented by the same character.

The octave number is a single digit, ranging from 1 to 9. These represent the octave ranges of a piano with 88 keys. Thus, the range starts at an A, as A1 is the lowest note on a piano. The octave number should typically never require double digits, as the octave reach on most instruments does not reach past 8. For violins, the lowest practical note is G3 (J3 in PTXT), and the highest one is E7 (H7 in PTXT). Theoretically, the number of characters could be reduced by omitting the octave number and instead labelling all pitches with a unique character, as there are a discrete number of practical pitches. However, the decision was made to include the octave number in PTXT, as the combination of pitch and octave allows for establishing connections between different pitches. The idea behind this was to potentially make it easier for the model to learn the relations of different pitches, which should help it convert an unseen combination with higher accuracy. The effect of this trade-off has not been tested further here. As a positive side-effect, it makes the PTXT format a lot more readable for humans as well.

Duration is handled through a dictionary, which maps note lengths to a single character. Theoretically, an infinite number of note lengths exist, as it is always possible to halve the length of the current note. However, in practice, the lowest note is a 256th note, as past this point notation becomes unclear and playing speed is impractical. Therefore, PTXT translates notes up to 256ths. The translation of duration-to-character can be seen in Appendix B. A dot is represented as a > character. In the event of ties, the character for the second note is printed directly after the first one. The characters used for duration are handled in such a way that the next element in the sequence starts playing after the duration of the first character is finished. To illustrate, consider the sequence "C4qhe D4h E4e". As the first duration character of the C4 corresponds to a quarter note, the D4 will be played a quarter note after the C4, while the C4 continues playing. Half a note later, the E4 starts playing, while the C4 is still held and the D4 stops.

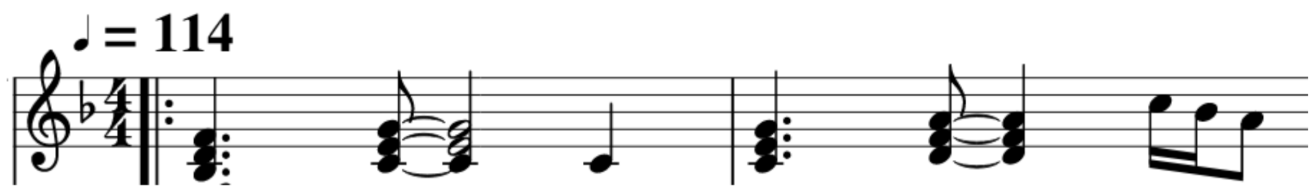


Figure 2: Example musical sequence. The PTXT equivalent of this sequence is "114 B4q>,F4q>,I4q> D4eq,H4eh,K4eh D4q] 114 D4q>,H4q>,K4q> F4eq,I4eq,A5eq D5s B5s A5e". Each chord can be seen as a word, with corresponding notes delineated by a comma and chords separated with a space.

Figure 2 shows an example of how the PTXT notation can be used to convert sheet music to text. Note that the time signature is not given in a PTXT sequence, as the cumulative duration of all elements in a measure is sufficient information to determine its corresponding time signature. The full PTXT notation can be written as a grammar in Extended Backus-Naur form, as seen in Listing 1.

Listing 1: PTXT grammar in Extended Backus-Naur Form.

```
sequence = measure, {"]", measure} ;
measure = number, " ", chord, {" ", chord} ;
chord = note, {",", note} ;
note = pitch, number, duration ;
pitch = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" |
    "K" ;
number = digit, {digit} ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
duration = length, dots, {length, dots} ;
length = "w" | "h" | "q" | "e" | "s" | "t" | "x" ;
dots = {">"} ;
```

4.2.2 PTXT parser

The PTXT parser was made to convert MXL sheet music data to the PTXT notation, using Music21. Both the MXL and PTXT formats store the elements of a musical score in order of occurence. Therefore, the parser is mostly able to read the elements in an MXL file and convert them sequentially to PTXT notation.

To determine the time when a note is played, as well as determine the duration of notes and rests, the PTXT parser uses a time step value as an index. The total number of time steps are determined by the smallest note duration played in the full piece. For instance, if the shortest played note is an eighth, a quarter note will have a length of two time steps. However, if a sixteenth exists somewhere in the piece, the quarter note has a length of four time steps.

The MXL format handles repeats by storing the time step of repetition brackets, similarly to the way it would appear in sheet music. To make repeats more comprehensible for an encoder-decoder model with an attention layer, PTXT does not provide designated repetition characters. Instead, the parser stores the sequence starting from the most recent opening bracket, or from the start if no opening bracket has been encountered yet. When a closing bracket is found, the stored sequence gets appended to the current sequence. This essentially unfolds the repeated sections inline within the sequence, allowing the attention layer to be less complex as it does not have to take possible repetition of a sequence explicitly into account. Adaptations of this technique are used to unfold repetitions with multiple endings and nested repetitions as well.

For musical scores with multiple parts, there exists a possibility of different parts playing a note with the same pitch on the same time step. When this occurs, the MXL file will store the same note twice, essentially storing a chord with two identical notes. As such a chord would sound similar to a single note, the parser combines all instances of identical notes on a given time step into one. This avoids introducing unnecessary complexity for the model.

4.2.3 Final preprocessing

In language, numerous words can be written in an alternative way, to either compress the text or make it more structurally comprehensive. When converting this text to speech, these alternative methods do not produce different audio from their original. For instance, converting the character "4" to speech should produce the same audio as the conversion of the expanded string "four". Other examples of speech equivalency include abbreviations ("jr." is equivalent to "junior"), capitalization ("Apple" is equivalent to "apple") and whitespaces (a space is equivalent to two spaces, a tab or a line break). To improve the accuracy of a speech synthesis model, text is often cleaned to only include one of each speech equivalency. In the case of Tacotron 2, the cleaning of input text includes converting all characters to lowercase, collapsing all whitespace to a single space and expanding numbers and abbreviations.

For traditional language, text cleaning improves the accuracy of an audio synthesis model significantly. However, applying the cleaning methods of Tacotron 2 to the PTXT notation has a notable adverse effect. The expansion of numbers and conversion to lowercase results in characters losing their unique meaning. As an example, the note "F5e" would be converted to "ffivee". In this new system, there is no clear delineation between pitch, duration and octave. Furthermore, the use of characters becomes ambiguous. Consequently, generating accurate audio from this sequence becomes much more difficult for a synthesis model. As the conversion from MXL to PTXT is consistent and PTXT aims to provide text in the most unambiguous and convenient form for AI prediction, additional cleaning of the input is not necessary.

As a final preprocessing step, the sequence of characters gets converted to a sequence of integers, before being forwarded to the model. Each character has a corresponding unique ID that is mapped through a dictionary. As the unique characters in PTXT is vastly different than the characters in normal text, the IDs for each symbol have to be redefined for PTP. The dictionary is generated based on all possible characters in PTXT, with ID's starting at 1 and counting up for each unique symbol. This process ensures the model receives a numerical representation of the text, which is necessary for performing mathematical operations on the text during model inference.

4.3 Text-Audio Alignment

The dataset solely contains sheet music and performances of full pieces and movements of violin music. To create practical training data from this, the sheet music must be split into sections of a few measures. However, doing this introduces the challenge of properly aligning the audio with the corresponding section in the score, to create accurate labels for training. The start and end times of each note have to be calculated so that the audio can be aligned sequentially. For the Bach Violin dataset, accurate note alignments have been estimated for all pieces using the method explained below [WC+23].

To get a general idea of the timings of each note, the MXL data of the sheet music can be utilized. With the combined information of the BPM and time signature, the expected length of each measure can be accurately calculated. As this is equivalent to the expected time of a whole note, the individual duration of each note can be calculated trivially, as each note is defined as a given fraction of a whole note. Then, the start time of a given note can be calculated by the sum of the previous notes, and the end time equates to the sum of the start time and its duration. Doing this chronologically for all notes in the sheet music results in a sequence of expected start and end times, corresponding to when a new note should be played.

The next step of the alignment process involves creating a digital audio representation of the sheet music. This is done by generating audio directly from the score, using a synthesizer with a digital violin sound font. In such a sound font, each pitch is mapped to a violin recording of that pitch, played in isolation. Audio conversion then becomes trivial, as one can simply sequence the appropriate audio samples according to the notes specified in the sheet music. This audio serves as a reference point for the timing of each note and is devoid of dynamics, articulation, tempo variations or other expressive elements. Because of this, the timings of each note are an accurate representation of the timings in the previously converted note sequence.

To perform a more accurate analysis of the relation between the digital reference audio signal and the expressive recording, both are converted to a spectrogram. For this audio signal processing task, it is preferred to calculate spectrograms using a constant Q transform, instead of the Short-Time Fourier transform used for creating mel-spectrograms. As previously mentioned in Section 2, transposing a pitch up an octave is equivalent to doubling its frequency. As each octave has 12 discrete pitches, the frequency difference between each pitch is exactly a factor of $\sqrt[12]{2}$. Therefore, transposing n pitches upwards displaces the frequency f by the following margin:

$$\Delta f = \sqrt[12]{2}^{n} = 2^{\frac{n}{12}} \tag{4}$$

$$n = 12 * \log_2(\Delta f) \tag{5}$$

Because of the logarithmic relation between frequency and pitch, higher pitches are more distant from each other in terms of frequency than lower pitches. However, as the Short-Time Fourier transform has a fixed resolution due to a consistent filter length, this results in varying frequency resolution between pitches. Specifically, the resolution diminishes as the difference from the centre pitch increases. A Constant Q Transform counteracts this, by using a logarithmic resolution depending on pitch to maintain a constant frequency resolution [Bro91]. Therefore, the resulting spectrogram has a much more accurate frequency representation than a Short-Time Fourier Transform would give, which aids in finding the exact timings of each played note in the audio.

With the Constant Q spectrograms of both the expressive audio and the digital reference, both sequences can be compared to each other. By calculating the similarity between each time step, a cost matrix can be constructed. This matrix represents the alignment cost between the corresponding time steps of the two spectrograms, with similar time steps costing less. A warping path can then be constructed, equivalent to the path with the lowest cost. The difference from this to the diagonal shows how to lengthen or shorten one of the spectrograms, such that both spectrograms align as accurately as possible throughout the whole piece.[Sen08]

As the note timings in the reference correspond to the previously converted note sequence, the found warping path can be applied to this sequence. This creates a new sequence, closely matching the timings of all played notes in the expressive audio. Finally, the sequence is split into parts with a length equal to the number of notes in the corresponding split text files. The starting time of the first note and ending time of the last note provide the respective domain in the full recording where the notes in the split files are played. Thus, splitting the recording at these times gives accurate audio labels for the split text files.

4.4 Experiment setup

4.4.1 Dataset

To train a system to be able to convert a musical score to audio, a dataset is required consisting of instrumental recordings and the sheet music each recording follows. However, as of writing, there are minimal extensive datasets of this kind available. This was noticed as well in 2022 by Dong et al. [DZBKM22], who collected approximately 6.5 hours of recordings of Bach violin sonatas with accompanying scores, to train the Deep Performer model proposed in the same paper. This "Bach Violin" dataset is the choice for training Tacotron 2 as well, as it is currently one of the most extensive open-source datasets available with recordings and corresponding scores. In addition, the dataset also provides the alignment files for all pairs in the dataset, generated using the method described in Section 4.3 [DZBKM22].

Note that training the vocoder solely requires audio labels, as the spectrograms used as input can be generated from the audio directly using an STFT. Thus, the number of available training datasets is significantly increased, allowing the vocoder to be theoretically trained on possibly more extensive data. However, most datasets with more than 6.5 hours of instrumental music are piano recordings. Should the vocoder be trained on this data, inference with spectrograms of violin music yields significantly less accurate results than a model trained on the Bach Violin dataset. Therefore, the same dataset will be used to train both models.

4.4.2 System Training

The Tacotron 2 audio synthesis model and the WaveGlow vocoder are both trained separately from scratch using the configurations shown in Table 4a and Table 4b, with training data provided by the Bach Violin dataset. The Tacotron 2 model was trained to estimate a mel-spectrogram from PTXT input, and WaveGlow was trained to generate audio from a mel-spectrogram. Models were trained until convergence, defined as the point where the validation loss does not significantly change over the course of five consecutive epochs. After a fixed number of iterations, a checkpoint of the model is created. The validation loss is calculated on these checkpoints. Both models use an Adam optimizer [KB17], with batch size and learning rate varying between the models. The model evaluations were done on a section of the Bach Violin dataset that was held back during training.

The training hyperparameters for both models are shown in Table 4d. The training data was split into sections of three measures for both Tacotron 2 and WaveGlow. Due to issues with working memory, Tacotron 2 could not be trained with a higher batch size than 2. Alternatively, the model would have to be trained with sections of a single measure to increase the batch size. However, splitting the audio like this loses crucial expressive information on the transition between measures and the construction of musical phrases by the performer. Thus, training the model on multiple measures with a lower batch size was decided to be preferred over training on a single measure with a higher batch size.

4.4.3 Model evaluation

To measure the performance of Tacotron 2, the model output will be compared to the ground truth spectrograms using mean squared error. Converting both spectrograms to matrices, the mean squared error E can be computed by averaging the accuracy of each pixel in the matrix, with N as the number of pixels and S_{truth} and S_{out} the spectral matrices of the ground truth and model output respectively:

$$E = \frac{1}{N}\sum_{i=1}^{N}(S_{truth}[i] - S_{out}[i])^2 \tag{6}$$

For this, both spectrograms need to be of equal length. Therefore, the error is calculated using a zero-padding appended to the end of the shortest of the two spectrograms. The final Tacotron 2 model will be compared to a similar model trained with linguistic cleaning, to show the effect on accuracy. Inference is done on a sequence of three measures to validate that the model implements expressive musical phrasing similar to the training data. Both models will be compared to both the ground truth, as well as digitally synthesized audio of the same score as a baseline.

The WaveGlow model audio output will be evaluated by a subjective listening test. Inference on the model will be done using spectrograms directly generated from the audio labels, and spectrograms generated from Tacotron 2, using the text corresponding to the audio labels as input. The former is done as a baseline, to test whether any apparent artefacts in the audio signal are caused by Tacotron 2 or WaveGlow. The listening test has solely been performed by the author of this thesis. Therefore, due to the minimal sample size, no remarks will be made on the human-like quality of the audio, often measured with a mean opinion score. Instead, the results will mainly focus on how the audio translates notes in the text input to actual pitch and duration, as well as the general intelligibility of the music.

5 Results

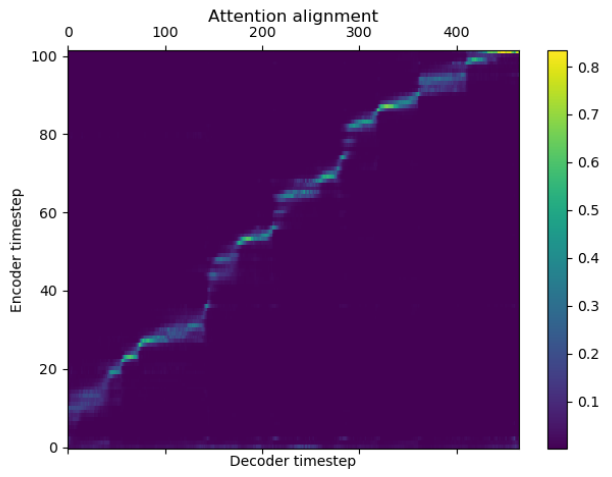
The acoustic model was trained on Tacotron 2 for 121,000 iterations, with the validation loss being computed every 1000 iterations. To show the effects of linguistic cleaning, another acoustic model was trained for 56,000 iterations on normalized text input. The vocoder was trained on the WaveGlow architecture for 220,000 iterations and evaluated every 2000 iterations. The learning curves of these models can be seen in Appendix D. Audio examples can be found in the repository attached to this thesis, linked in Appendix C. A table is provided here that categorizes samples according to which of the remarks made below they demonstrate most.

5.1 Audio output

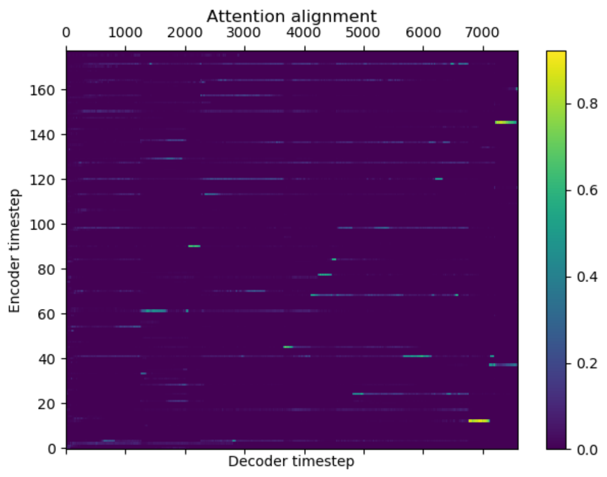
In general, the audio produced by the system does a good job of converting the notes in the input text to violin music. The pitches of each note are accurate, and notes are played for as long as they are expected to in the given tempo. In particular, instead of sounding like each note is recorded in isolation and then stitched together, the recording sounds more realistic and continuous. The transitions between notes are smooth, and there exists a natural flow to the music similar to live violin playing. With chords, multiple notes are properly played at the same time and rests correctly create a section of silence in the recording.

A notable difference from the target audio is that the model always chooses to play the input sequence fully legato, meaning transitions between notes are connected and contain no intervening silence. The performers of the target audio often switch to playing with slight staccato, where notes are played for less long and a short silence exists between two notes. This distinction was not explicitly specified during training, and holding a note for slightly longer within its duration window will not penalize the model significantly. Therefore, constantly playing legato is a consistent method of obtaining a decently similar result to the target audio.

Additionally, audio from the model often contains some amount of echoing. On the one hand, this increases its realism, as notes generated from a digital synthesizer often sound pre-recorded. However, the echo is not similar to the training data, instead being more muffled. This results in overall less pleasant audio. By converting the target audio to a mel-spectrogram directly and doing inference on the WaveGlow model with it, the echo persists. Thus, this is caused by the general translation from the mel-spectrogram to audio and is not a residue from Tacotron 2 attempting to predict the spectrogram.



(a) No normalization



(b) Normalization

Figure 3: Alignment plots of Tacotron 2, for input sequence ”90 D4e,H4e,A4e,A5e A4s C4s A4e A5e] D4e,H4e,A4e,H5e D5e C3e,L4e,F5e C4e] A3e,A4e,D5e I4e K4e,E5e H4e”

5.2 Text normalization

Comparing the performance of Tacotron 2 with the variant trained using linguistic cleaning on the PTXT input, the effect of properly structured input text becomes immediately apparent. The alignment plots of both models on the same input sequence are shown in Figure 3. The alignment plots visualise the area where the attention layer is focused on each decoder time step. Compared to the model with normalized text, not changing the PTXT format results in the attention layer focusing significantly more effectively. In the cleaned model, attention is much more spread out each time step, often switching focus to a completely different encoder time step seemingly randomly. As individual notes can often be processed and played sequentially, the model should only slightly emphasize the recent past to keep track of note duration, and only look further in the past when chords appear in the sequence.

This effect is reflected in the audio generated by both models. The cleaned model struggles to generate music in line with the prompted sequence. Often, the first note will be accurate in pitch, but its duration will be significantly off. After a short while, the model loses meaningful focus and either repeats the current note forever, or stops playing entirely. In addition, the audio is noticeably more muffled than the model without text normalization. Looking at the accuracy of both models regarding the generation of mel-spectrograms, the discrepancies become significantly more apparent. Figure 6 shows the generated mel-spectrograms of both models for an equal input sequence, compared to the ground truth mel-spectrogram. The accuracy of these predictions is shown in Table 1, with a 95% confidence interval. The model without text normalization appears as a version of the ground truth with much less noise, mimicking aspects such as pitch and duration highly accurately. In contrast, the normalized spectrogram is only able to hold a single note for a long time, but this note often has a completely different pitch and duration compared to the ground truth.

Acoustic model	**Ground Truth MSE**	**Digital MSE**
Digital sound font	6.09 ± 0.15	-
Tacotron 2	4.23 ± 0.18	3.30 ± 0.23
Tacotron 2 with text normalization	50.57 ± 1.46	48.13 ± 1.83

Table 1: MSE loss with 95% confidence interval of both Tacotron 2 spectrogram outputs, compared to the ground truth and spectrograms generated through the use of a digital soundfont.

5.3 Expressiveness

Compared to audio generated using a synthesizer with a digital sound font, the model output sounds notably more like a live recording. This is mostly due to the fact that the transition between notes happens more smoothly, instead of sounding like isolated notes stitched together. In addition, the audio contains several small techniques that make it sound more expressive. For instance, the duration of a note can deviate slightly from the actual length to increase its realism, mimicking the subtle timing variations a human performer might introduce. Notes are played slightly longer in places where it increases tension, or cut short when a section benefits from being played slightly faster. The output audio contains parts where dynamics are notably varied. This usually happens abruptly for long sections, similar to terrace dynamics in human performances. Finally, chords are sometimes played as an arpeggio, one note after each other instead of at the same time, without being explicitly prompted to do this. All these things help distance the model from digital synthesizers and increase expressiveness.

The accuracy of the model reflects these observations. Looking at Table 1, Tacotron 2 is more accurate to the ground truth than the digital sound font. As the objective features of played notes are equivalent for all models, most of the difference comes from the implementation of expressive variation. Nevertheless, the output still has more resemblance to a digital sound font than to a human recording. All expressive features are applied on a minimal level, significantly less than most professional performers would implement in their playing. Therefore, the generated mel-spectrograms remain similar to their digital counterparts.

6 Discussion

6.1 Model Limitations

The function to convert a spectrogram to the Mel-scale is logarithmic, meaning higher frequencies will lose resolution in favour for lower frequencies. This is useful for speech synthesis, as important frequencies in voices are mostly located on the lower end of the frequency spectrum. However, violins use significantly higher frequencies, which may result in a loss of accuracy when using these spectrograms for violin music generation. Using a different scale for the spectrogram might result in better performance compared to using a mel-spectrogram.

Another important remark to make is that both Tacotron 2 and WaveGlow were trained on a relatively small dataset, for a relatively short time and small batch size. To be able to generate spectrograms close to human quality, Tacotron 2 was trained on a dataset containing approximately 25 hours of audio from a single speaker. In addition, training was done for approximately 300,000 iterations [SPW+18]. Similarly, WaveGlow was trained for 580,000 iterations on the LJ Speech dataset, containing approximately 24 hours of short audio clips from a single speaker. In contrast, the model shown in this thesis was trained on much less data, as well as being recordings of the same pieces from different performers. The limited data also resulted in the training curve converging significantly faster. With a more extensive dataset, the fidelity of the model would notably increase. However, as of now, no larger violin music datasets are available for training.

In addition, the architectures used in this thesis are not the current state-of-the-art. Tacotron 2 was released in 2017 [SPW+18], and WaveGlow in 2018 [PVC18]. Since then, numerous models have been published that have achieved a higher performance. For instance, FastSpeech [RRT+19] and Transformer TTS [LLL+19] have both achieved a higher Mean-Opinion Score (MOS) than Tacotron 2. These have been shown to provide more expressiveness in their mel-spectrograms, with better length control and the ability to add breaks to improve prosody. Similarly, models such as Hifi-GAN [KKB20] can exceed the quality of audio generated by WaveGlow.

Finally, the PTXT notation could be expanded to contain more data about the notes being played. For instance, information on dynamics was not included in this version of the notation. Furthermore, a note could include whether it is meant to be played staccato, legato or otherwise. Expanding this idea further, additional documentation from the sheet music could be added as well. Adding more elements to the notation makes the text more informative, at the cost of introducing more characters for the model to process. In other words, providing extra information in the PTXT notation might make the model output more expressive audio. However, this audio could be less accurate, due to the sequence length increasing. Furthermore, this might cause an adverse effect, where the model receives too much information about how to play a note and struggles to add any creative expressiveness of its own.

6.2 Generalizability of TTS architecture

Despite its limitations, the trained model demonstrates the feasibility of generating natural-sounding violin music using an architecture specifically made for speech synthesis. Objective note features, such as pitch and duration, are translated with high accuracy, as can be seen from the similarity to the digital audio. Furthermore, the model shows signs of expressiveness, although more held back than often heard in human-recorded music. The observation of increased noise in the audio can be attributed to the conversion from a mel-spectrogram done by WaveGlow. As the spectrogram prediction step is responsible for generating the general features and expressiveness of the output, this noise does not limit the overall evaluation of the model's capabilities.

However, it is important to note that an existing model cannot simply be adapted to generate other types of audio. To be able to generate violin music for this thesis, a new PTXT grammar had to be defined, which the target input had to be properly represented in and converted to. As seen from the linguistic cleaning example, each grammar has separate normalization criteria, which had to be taken into account as well to generate proper audio. The original model was trained using a dataset of English language, and would not convert a different grammar to anything meaningful. Therefore, the PTXT grammar then had to be used to train a model using the Tacotron 2 architecture from scratch, to generate mel-spectrograms of violin music. However, violin music and speech have vastly different mel-spectrogram representations, and so, a vocoder trained in speech can not accurately generate audio from it. Consequently, WaveGlow had to be retrained from scratch as well.

Nevertheless, doing all this does generate the wanted audio from the architecture. Moreover, these steps can be seen as a general method for creating other sounds as well, as the architecture is likely to be adaptable to various types of audio. This only requires the audio to be convertible to an informative textual representation, as well as having an existing dataset to train on. As seen in this thesis, such a dataset does not have to be incredibly large; a few hours is sufficient to train a functional proof-of-concept model. This implies it is likely possible to train models for other applications as well, such as for prompted generation of natural ambience, animal sounds or white noise, among numerous others. All of these share a similarity with speech as music does: an audio frame of this type can be broken up into smaller components, each detailing the most fundamental parts of the sound. In essence, this similarity is what is required to convert the audio to a detailed textual representation, which can be learned by a speech synthesis model to generate expressive sequences.

This generalization is not limited to Tacotron 2 and WaveGlow. As mentioned in Section 6.1, numerous other architectures have been created that allow for more accurate text-to-speech synthesis. Despite these architectures being different in some cases, the processing of text to predict audio frames is done using similar methods. In addition, some of these architectures use a different intermediate output than a mel-spectrogram. An example of this is VALL-E [WC+23], which computes a matrix of audio codecs as intermediate output. Nevertheless, the type of output the audio prediction model generates is not relevant, as long as a vocoder exists to translate that type of output to audio. Given the fundamental similarities between music and speech, it is likely the same model creation steps work for most other architectures as well. With this, it becomes possible to utilize the strengths of different systems to process general audio similarly to how speech is produced. For instance, the VALL-E model excels in speech synthesis, being able to imitate the voice of a speaker accurately with approximately 3 seconds of audio samples [WC+23]. This could be applied to music as well, where a model can predict the interpretation a performer would give for a certain musical piece. Similarly, FastSpeech was designed as a speech synthesis model that significantly speeds up inference compared to Tacotron 2 [RRT+19], which can also be used for other types of audio.

7 Conclusion

This thesis has aimed to demonstrate the viability of generalizing speech synthesis models for generating expressive music. This has been accomplished by training a model on the Tacotron 2 encoder-decoder architecture to create mel-spectrograms from violin sheet music and training a vocoder on the WaveGlow architecture to convert these mel-spectrograms to a waveform.

To provide input to Tacotron 2, sheet music was converted to a textual representation by introducing the PTXT notation. This grammar encompasses all objective information about each note, chord and rest in the piece in as little characters as possible, to help the model convert essential features properly. The full system was trained on the Bach violin dataset, using dynamic time warping to obtain accurate audio labels corresponding to the text splits.

Findings showcase the model is capable of converting all essential features of notes accurately to audio and can add some slight expressiveness to it as well, such as variations in tempo or dynamics. A comparison of the generated mel-spectrograms with both the ground truth and a spectrogram generated with a digital sound font shows the model output is closer to digital audio than human recording. Nevertheless, it is also significantly more expressive than its digital counterpart. Combined with the fact that other types of audio show similarities with music and speech regarding their features, it is thereby concluded that the generalization of speech synthesis models to other expressive audio types is not only viable but also promising for future applications.

7.1 Further research

Given the possibility of applying the described methods to other text-to-speech architectures, the question arises which one provides the best performance for generating other types of audio. Future research can study this, by training models on other architectures and for varying audio types, to attempt to figure out which one is most plausible for high-accuracy general audio synthesis, or looking into what architectures perform the best for each audio type separately.

As the listening test was only conducted by the author of this thesis, the sample size was too minimal to compute any accurate metrics on more subjective aspects of the audio output, such as relative sound quality or similarity to human recordings. Looking forward, similar studies can investigate how well a model trained on a speech synthesis architecture compares in these aspects to a model from an architecture specifically made for its type. This can be done by, for example, calculating the mean opinion score of each model based on the results of an outsourced survey [WSRS+17].

Additionally, Tacotron 2 was built solely for speech generation, and each component is therefore vital for that process specifically. Each part fulfils a separate role to generate speech with the most fidelity possible. When looking at music generation, the importance of each component will likely shift. For instance, the attention mechanism, which aligns input text with output audio in a speech model, might play a more important role in music, to handle its inherent rhythmic and harmonic structures. Future research could expand on this, such as by comparing the performances of both speech and music models in an ablation study. This shows how music and speech are related, and what parts of the model are most vital for both applications.

Looking forward, further research on this topic might eventually culminate in the construction of a model that can convert text to any type of audio with high accuracy and expressiveness. Such an advancement would not only enhance the entirety of the speech synthesis field but the audio signal processing field as a whole. In the future, these enhancements can lead to numerous applications in various fields, such as interactive media, virtual reality and assistive or medical technologies. As research progresses, exploring the adaptability and scalability of different architectures for diverse audio types will be crucial for realizing these transformative applications, and help push the boundaries of audio signal processing in both technical and creative domains.

References

[AR77] Jont B Allen and Lawrence R Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

[ATKR20] Shalbbya Ali, Safdar Tanweer, Syed Sibtain Khalid, and Naseem Rao. Mel frequency cepstral coefficient: a review. *ICIDSSD*, 2020.

[Bro91] Judith Brown. Calculation of a constant q spectral transform. *Journal of the Acoustical Society of America*, 89:425–, 01 1991.

[CBS+15] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[CW03] Darrell Conklin and Ian Witten. Multiple viewpoint systems for music prediction. *J. New Music Res*, 24, 06 2003.

[DM80] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

[DZBKM22] Hao-Wen Dong, Cong Zhou, Taylor Berg-Kirkpatrick, and Julian McAuley. Deep performer: Score-to-audio music performance synthesis. *arXiv preprint arXiv:2202.06034*, 2022.

[GL84] Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on acoustics, speech, and signal processing*, 32(2):236–243, 1984.

[HVU+18] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[KC90] Roger A Kendall and Edward C Carterette. The communication of musical expression. *Music perception*, 8(2):129–163, 1990.

[Kin03] Tomi Kinnunen. Spectral features for automatic text-independent speaker recognition. *Licentiate's thesis*, 2003.

[KKB20] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *arXiv preprint arXiv:2010.05646*, 2020.

[LLL+19] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with transformer network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6706–6713, 2019.

[ODZ$^+$16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[PSR20] Abhilash Pal, Sourav Saha, and Anita Ramalingam. Musenet : Music generation using abstractive and generative methods. *International Journal of Innovative Technology and Exploring Engineering*, 9:784–788, 04 2020.

[PVC18] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. *arXiv preprint arXiv:1811.00002*, 2018.

[RRT$^+$19] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech: Fast, robust and controllable text to speech. *Advances in neural information processing systems*, 32, 2019.

[Sen08] Pavel Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

[SPW$^+$18] Jonathan Shen, Ruoming Pang, Ron J Weiss, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4779–4783. IEEE, 2018.

[TQSL21] Xu Tan, Tao Qin, Frank Soong, and Tie-Yan Liu. A survey on neural speech synthesis. *arXiv preprint arXiv:2106.15561*, 2021.

[WC$^+$23] Chengyi Wang, Sanyuan Chen, et al. Neural codec language models are zero-shot text to speech synthesizers. *arXiv preprint arXiv:2301.02111*, 2023.

[WSRS$^+$17] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

A Configuration details

Hyperparameter	Value
Sequence embedding size	512
Encoder kernel size	5
Encoder convolutional layers	3
Encoder embedding size	512
Decoder RNN layer size	1024
Pre-net layer size	256
Decoder gate threshold	0.5
Decoder dropout	0.1
Attention dropout	0.1
Attention RNN layer size	1024
Attention filters	32
Attention kernel size	31
Post-net embedding size	512
Postnet kernel size	5
Postnet convolutional layers	5

(a) Configuration of Tacotron 2 audio synthesis model

Hyperparameter	Value
Mel channels	80
Flows	12
Samples per passthrough	8
Affine coupling transform layers	8
Affine coupling transform channels	256
Affine coupling transform kernel size	3

(b) Configuration of WaveGlow vocoder model

Hyperparameter	Value
Audio channels	1 (Mono)
Amplitude maximum	32768
Sampling rate	22050
STFT filter length	1024
STFT hop length	256
STFT window length	1024
Mel bands	80
Frequency minimum (Hz)	0
Frequency maximum (Hz)	8000
Measures per label	3

(c) Audio Preprocessing configuration

Hyperparameter	Tacotron 2	WaveGlow
Batch size	2	8
Learning rate	10^{-3}	10^{-4}
Iterations per checkpoint	1000	2000
Precision	Single	Single
Gradient clipping threshold	1.0	-
Weight decay	10^{-6}	-

(d) Training hyperparameters

Figure 4: Configurations and hyperparameters of Tacotron 2 and WaveGlow models.

B PTXT translation scheme

Musical notation	PTXT
A, G##, B&&	A
A#, B&	B
B, A##, C&	C
C, B#, D&&	D
C#, D&	E
D, C##, E&&	F
D#, E&	G
E, D##, F&	H
F, E#, G&&	I
F#, G&	J
G, F##, A&&	K
G#, A&	L

(a) Pitch conversion

Note type	PTXT
Whole	w
Half	h
Quarter	q
Eighth	e
16th	s
32nd	t
64th	x
128th	o
256th	p

(b) Duration conversion

Table 2: Translation scheme of the PTXT notation. Sharps are represented as #, and flats as &.

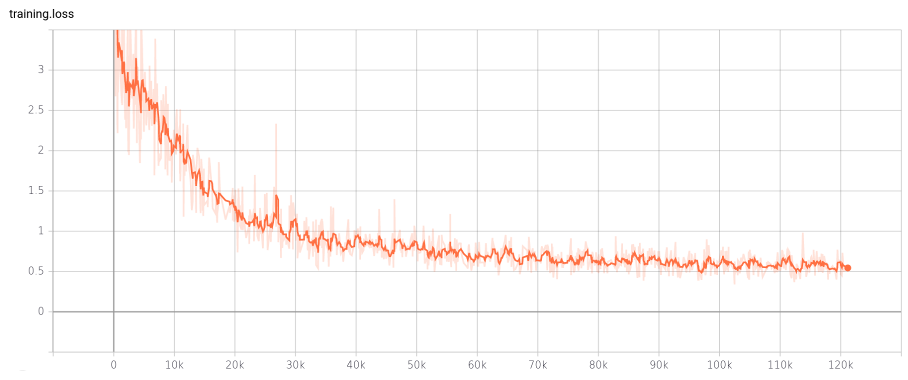
C Repository

Accompanying this thesis is a repository that implements the discussed PTXT parser and provides additional scripts that aid preprocessing and inference. Additionally, it contains examples of the final model output. The repository can be accessed at https://github.com/Stannie04/ptp. Table 3 contains direct links to samples in the repository showcasing some of the discussed features in Section 5.

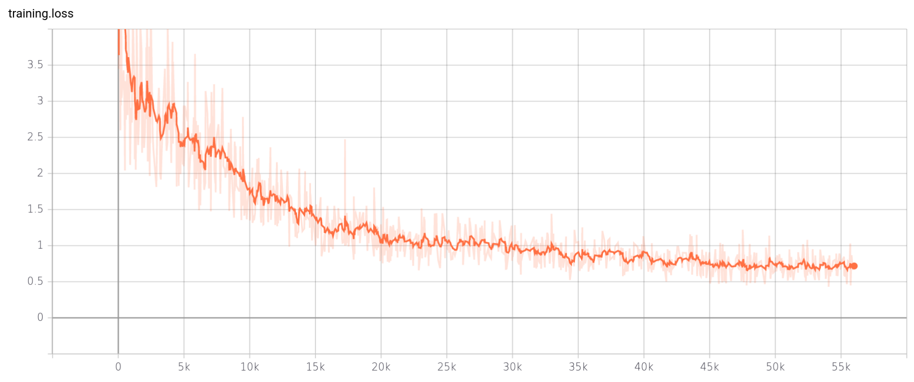
Description	Link
Demonstration of objective features (pitch, tempo)	https://github.com/Stannie04/ptp/tree/main/examples/objective
Demonstration of expressiveness (dynamics/tempo variation, arpeggios)	https://github.com/Stannie04/ptp/tree/main/examples/expressiveness
Inaccuracies caused by using an speech text normalization on PTXT	https://github.com/Stannie04/ptp/tree/main/examples/bad_normalization
Echo generated directly from the ground truth	https://github.com/Stannie04/ptp/tree/main/examples/direct_audio

Table 3: Links to sample references on audio results.

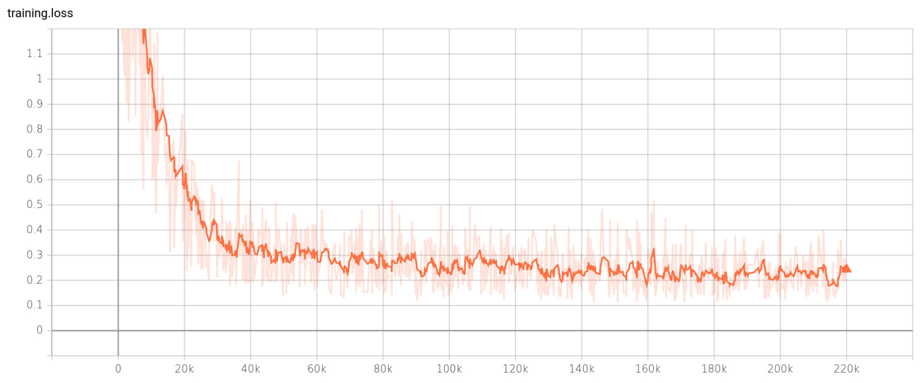
D Training results



(a) Tacotron 2

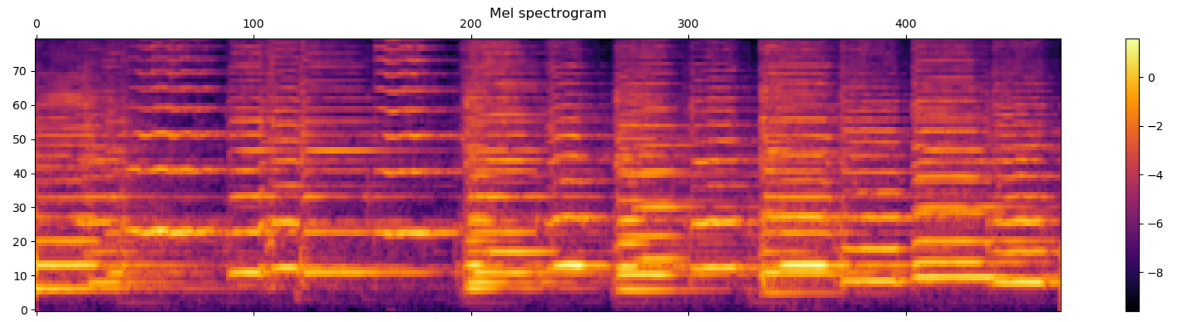


(b) Tacotron 2 with linguistic cleaning on PTXT

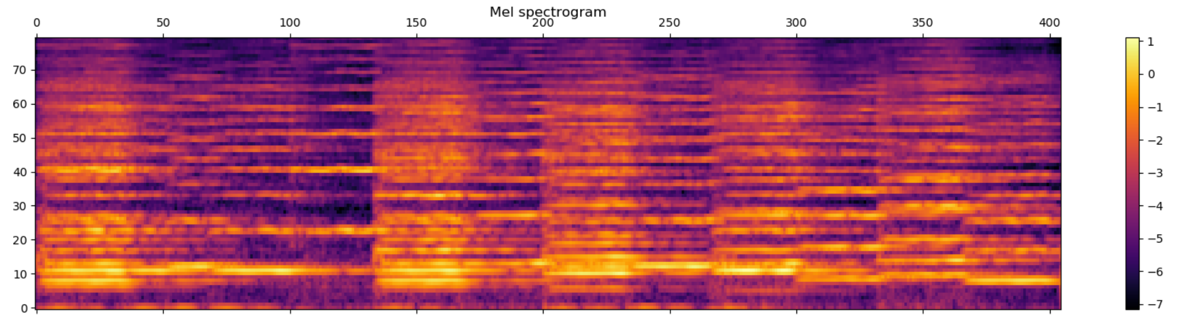


(c) WaveGlow

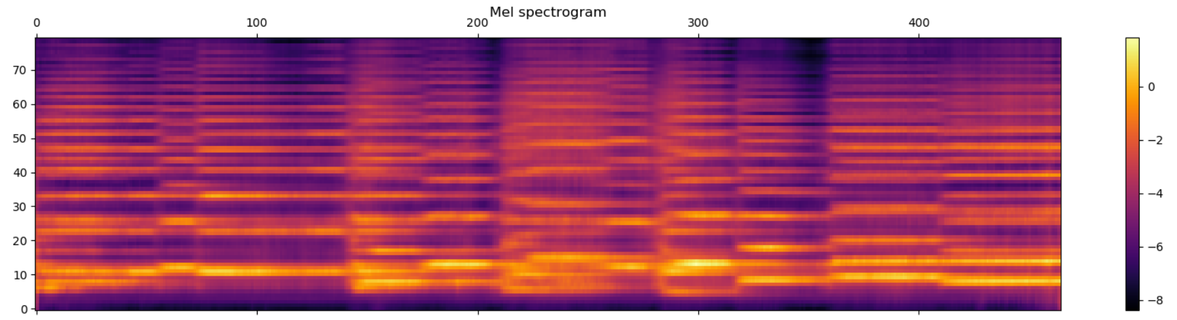
Figure 5: Learning curves of trained models, as provided by Tensorboard.



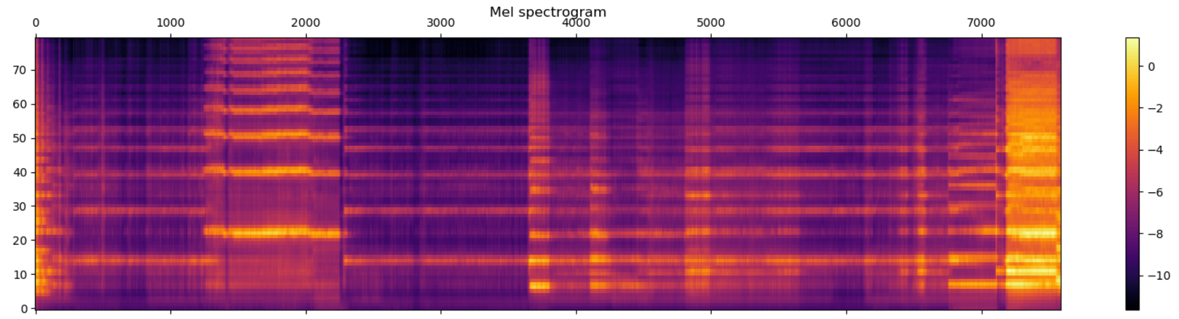
(a) Ground truth



(b) Digital sound font



(c) Tacotron 2



(d) Tacotron 2 with linguistic cleaning on PTXT

Figure 6: Mel-spectrograms of the ground truth, both Tacotron 2 models and a representation using a digital sound font, by stitching together notes recorded in isolation. All spectrograms were generated using the input ”90 D4e,H4e,A4e,A5e A4s C4s A4e A5e] D4e,H4e,A4e,H5e D5e C3e,L4e,F5e C4e] A3e,A4e,D5e I4e K4e,E5e H4e”