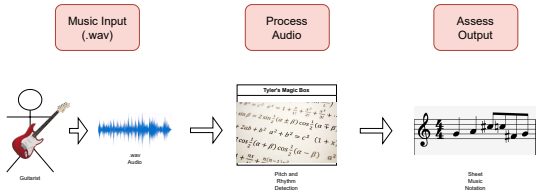
Technical Report v3

Tyler Jones

November 2025

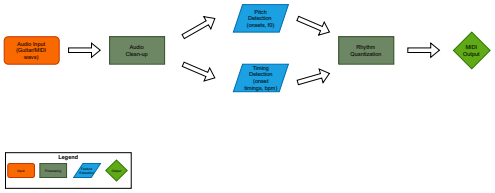
1 Graphical Abstract



2 Abstract

Automatic Music Transcription (AMT) has long been a complex and evolving challenge. Beginning with James A. Moorer's research in the 1970s [5], research has progressed from early polyphonic analysis to modern Transformer-based neural architectures, which achieve remarkable accuracy in rhythm and pitch detection [1]. In contrast, this work primarily focuses on monophonic transcription, with an emphasis on processing distorted guitar input. Building upon classic techniques like onset detection, frame-by-frame pitch detection, pre and post processing, along with more modern techniques like convolutional neural networks built on detecting monophonic pitch [2], this project aims to create a streamlined system that takes .wav audio input, detects note pitch and rhythm quality, and outputs a symbolic representation of the original input as either MIDI and/or sheet music. Leveraging Python libraries like `librosa` for onset detections, `crepe` for model-based monophonic pitch detection, and `music21` for MIDI representation, I've outlined a process for generating MIDI files that can be input into notation software like MuseScore and evaluated through visual and auditory inspection. This structure has shown promising results in both MIDI-created distorted guitar input and real distorted guitar recorded through a Digital Audio Workstation (DAW), both of which were exported as .wav files.

3 Data Architecture Diagram



4 Introduction

Automatic Music Transcription (AMT) is the process of converting an audio recording into a symbolic representation such as sheet music, MIDI (a language designed to store musical information), or tablature (a form of musical notation that provides finger positioning along with note pitch).

Even in monophonic cases, AMT presents challenges. Recreating the human ability to listen to and transcribe music with a computer requires it to handle many complex operations on a dynamic scale. While programs have been able to reliably detect pitch and rhythm of static notes, adding variations to those notes (the human element) drastically decreases accuracy. Musical techniques like vibrato, bends, slides, syncopation, staccato, and accents– essential tools for incorporating dynamics and phras-

ing into music–cannot be reliably dictated through a computer.

Recent advances in modeling and deep learning have significantly improved transcription accuracy. Models like Onsets and Frames [1] and Convolutional Representation For Pitch Estimation (CREPE) [2]–the `CREPE` model being packaged into a Python library for ease of use–have substantially enhanced the accuracy of pitch detection for both polyphonic and monophonic music, respectively. More recently, transformer-based modeling has been shown to be very promising, with the ability to transcribe polyphonic music featuring multiple instruments, melodies, and timbres with an accuracy that is almost too good to believe [4].

This project aims to design low-level, but accurate, AMT software for distorted, monophonic guitar input. Using Python libraries such as `librosa` for general audio analysis, `CREPE` for model-based, monophonic pitch detection, and `music21` for MIDI representation, the program outputs a MIDI file from the input .wav file. This, in turn, can be opened in a free music notation software, such as MuseScore, my preferred method, for both visual and auditory analysis.

5 Related Work

5.1 Classical Approaches to Music Transcription

Early work in AMT relied heavily on signal processing techniques such as the short-time Fourier transform (STFT), constant-Q transform (CQT), and harmonic-percussive source separation (HPSS) to detect note onsets and estimate fundamental frequencies [3]. These classical methods worked reasonably well for monophonic audio or isolated instrument recordings, but work as effectively for polyphonic input.

5.2 Deep Learning in AMT

The introduction of deep learning marked a significant turning point in AMT research. Neural networks enabled models to learn hierarchical representations of audio signals, eliminating the need for hand-crafted features. The Onsets and Frames model [1] proposed a dual-objective architecture that separates onset detection from frame-wise pitch tracking, significantly improving transcription accuracy for piano recordings.

Transformer-based models, such as those developed under Google's Magenta project, have shown strong performance on structured musical passages by modeling music as a sequence of tokens [4]. These models are capable of attending to long-range dependencies and have been shown to work well across different instruments and styles.

5.3 Self-Supervised and Unsupervised Learning

Labeling large datasets of transcribed music is both time-consuming and expensive. To address this, recent work has explored self-supervised frameworks for learning that enable models to learn from unlabeled audio. Techniques such as BYOL (Bootstrap Your Own Latent) and contrastive predictive coding have shown promising results [6]. These methods provide a foundation for building more robust transcription systems that are less dependent on labeled training data.

5.4 MIDI and Symbolic Output

A key component of AMT is converting detected pitch and timing information into meaningful symbolic representations. Libraries like `music21` have made it easier to programmatically generate and manipulate Western music notation and MIDI files [**?**]. However, aligning these symbolic outputs with expressive performance elements (e.g., tempo rubato, articulation) remains an active area of research.

6 Methods

AMT has advanced significantly since its computational inception in the 1970s. However, due to the scale of my project and technical limitations still plaguing AMT as a whole, music in its most natural form cannot be transcribed accurately with my methods.

To achieve an acceptable level of transcription, the input music must be tailored to the software's strengths and away from its weaknesses.

6.1 Onset Detection Inaccuracy

Accurate onset detection is essential for identifying note events. If `librosa` fails to properly detect an onset due to any variety of obstacles, transcription output may be misaligned or incomplete.

Method: I will employ a post-onset detection form of musical quantization (the process of adjusting notes to a predefined grid). This ensures that each note's onset will be placed at a predictable and logical location based on the tempo of the input audio. This, however, would not solve the issue of note durations being shorter or longer than the intended note.

An additional step in fixing unaligned onsets would be to remove the rhythmic reliance on onsets altogether. If the tempo (bpm) of the input audio can be provided to the program, it can calculate the durations of note qualities–whole note, quarter note, eighth note, etc, etc–and estimate the quality of each note based on the time between its onset and offset. This is an attractive and seemingly simple solution to the aforementioned problem, but it can lead to exponential inaccuracy if note qualities are misinterpreted.

6.2 Ambiguity in Pitch Estimation

Even in monophonic audio, estimating fundamental frequencies can result in octave errors or spurious harmonics, especially in non-piano instruments or heavily reverberant recordings.

Even in monophonic audio, pitch estimation still presents a challenge. Octave errors, harmonic overtones, vibrato in the note, musical effects applied to the instrument (such as reverb, chorus, delay, and distortion), and even extraneous noise from the recording software itself all interfere with the software's ability to detect pitch.

Method: I will incorporate a variety of audio filtering techniques along with program-level pre and post-processing of the audio and its elements. The `librosa` library offers a built-in normalize utility, which smooths out any audio level spikes and dips, effectively "normalizing" the overall volume of the audio input.

I can also use the `scipy` library to create a band-pass filter function, which allows for only a specific range of frequencies to pass through. This method greatly reduces the amount of harmonics mistaken for the desired pitch.

When running `CREPE`'s pitch detection, it returns a list of variables for each measured frame. One of those variables is the confidence level in the estimation. This allowed me to create a function with a confidence level floor, which enabled me to eliminate any estimations with confidence levels lower than the floor.

6.3 Difficulty in Evaluating Transcription Quality

Evaluating the accuracy of computer-based transcription is best done with an audio input and respective ground truth annotations (known, correct data representations of the audio input). However, without access to existing datasets, evaluating the program's accuracy requires manual annotation if I wish to avoid manually evaluating the results of each test.

Method: I will create a small dataset with audio inputs accompanied by annotations. I have already created several audio examples, and due to the limitations of my program's transcriptions, the audio inputs are relatively simple and would be easy to manually annotate. I can then create and run automated tests that test various variable combinations while logging the best-performing transcriptions.

7 Results

7.1 Pitch Detection

The majority of pitch detection tuning–so far–has been done with MIDI distortion guitar converted to .wav audio files. A free, web-based application `https://signalmidi.app/` was used to create the MIDI files. This method was quick and efficient, allowing for the almost instantaneous creation of different techniques to test.

7.1.1 Quarter-Note Pitch Detection (120 BPM)

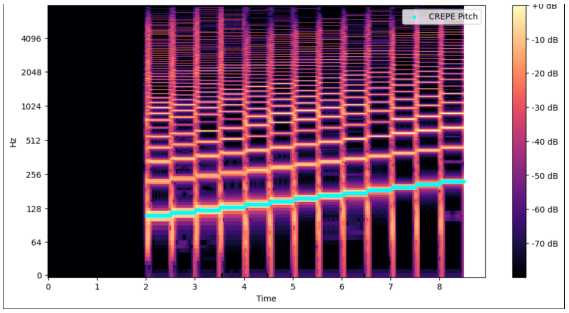


Figure 1: MIDI Example One (A2-A3, 120bpm, Quarter Notes)

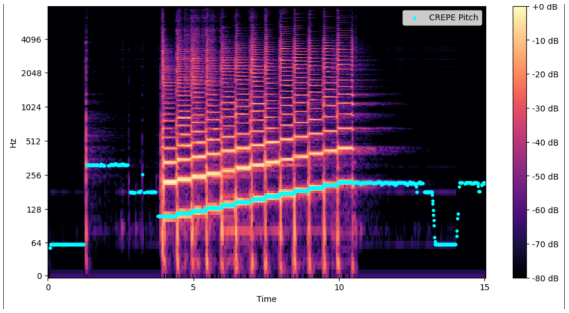


Figure 2: Real Overdriven Guitar Example One (A2-A3, 120bpm, Quarter Notes)

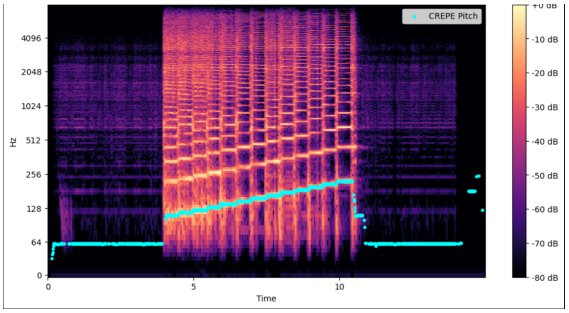


Figure 3: Real Distortion Guitar Example One (A2-A3, 120bpm, Quarter Notes)

Pitch detection at 120 BPM using quarter-note input is consistently accurate across all three test cases (Figures 1-3). These examples were intentionally designed to align with the program's strengths: small intervals, uniform note spacing, and tight note events. While these inputs do not reflect natural performance conditions, they serve as a controlled proof of concept for baseline program accuracy.

Figure 1 (MIDI input) demonstrates the highest accuracy, as expected. With virtually no ambient noise caused by the recording software or performance mistakes, the program can track pitch cleanly and reliably.

In contrast, Figures 2 and 3 introduce the challenges associated with real guitar recordings. These signals contain substantial extraneous noise, including hand movement across strings, mechanical bumps, pickup interference, overtones, and general inconsistencies expected from human playing. Much of this noise can be filtered out through preprocessing–such as normalization, band-pass filtering, and discarding low-confidence estimates provided by CREPE—but they still create a noisier environment than the MIDI baseline.

Despite this, the system maintains strong pitch-tracking performance in both real overdriven and real distortion guitar recordings. Harmonic filtering effectively suppresses overtones that might otherwise be mistaken for fundamental pitches. Additionally, the implementation of an average-pitch smoothing algorithm helps stabilize the pitch of the note for the duration of its event. Examples of the pitch smoothing–appearing as perfectly horizontal contours–are visible in Figures 2 and 3, constrasted by the notes the program fails to smooth.

7.1.2 Eighth-Note Pitch Detection (120 BPM)

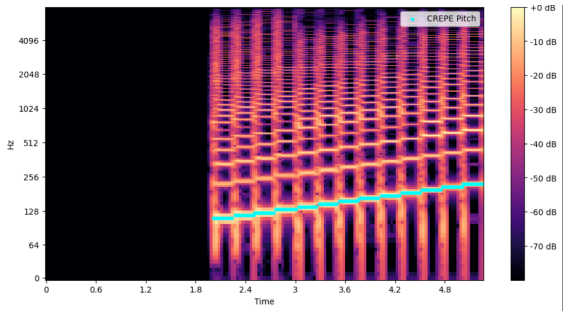


Figure 4: MIDI Example Two (A2-A3, 120bpm, Eighth Notes)

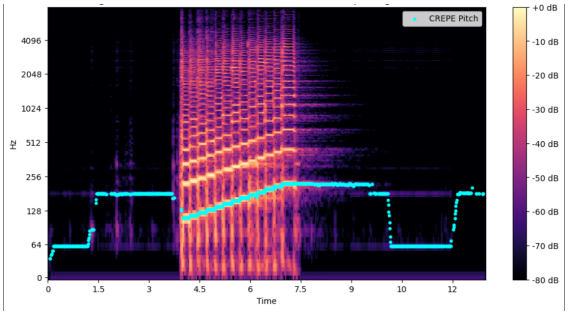


Figure 5: Real Overdriven Guitar Example Two (A2-A3, 120bpm, Eighth Notes)

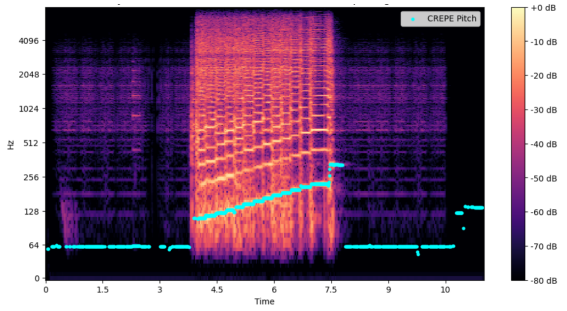


Figure 6: Real Distortion Guitar Example Two (A2-A3, 120bpm, Eighth Notes)

Doubling the speed to eighth notes introduces additional challenges for the pitch-detection program, but the overall trends remain consistent with the quarter-note results.

Figure 4 (MIDI) shows nearly identical accuracy to its quarter-note counterpart, as the clean synthetic input provides little opportunity for error.

For real guitar recordings, the increased event rate amplifies the effects of noise and timing variability. In Figure 5 (overdriven guitar), pitch accuracy remains strong, though the shorter note durations make the estimated note durations less stable and prone to slight fluctuations. Figure 6 (distortion guitar) experiences the most decline in performance. The smoothing algorithm fails more frequently, note boundaries become less consistent, and noise interacts more strongly with the denser sequence of events. Despite that, the predicted pitch trajectories generally follow the correct contour, and the intended notes remain relatively identifiable.

7.2 Rhythm Detection

For clarity and presentation, rhythm analysis includes screenshots of the output MIDI imported into MuseScore. This shows both the rhythm quality and pitch of the notes–using the same test audio from the previous results section.

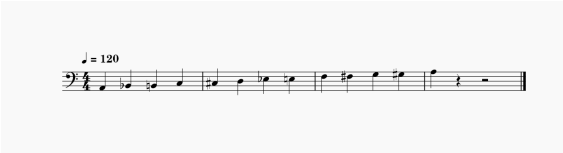


Figure 7: MIDI Rhythm Example One (A2-A3, 120bpm, Quarter Notes)

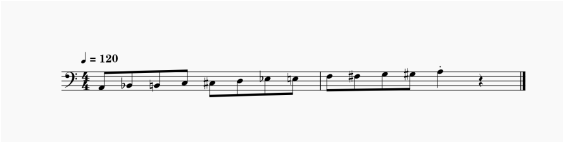


Figure 8: MIDI Rhythm Example Two (A2-A3, 120bpm, Eighth Notes)

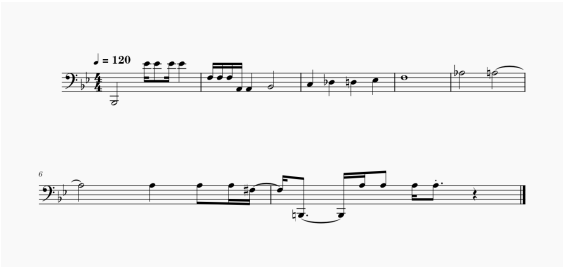


Figure 9: Real Overdriven Guitar Rhythm Example One (A2-A3, 120bpm, Quarter Notes)

Figure 10: Real Distortion Guitar Rhythm Example One (A2-A3, 120bpm, Quarter Notes)

Rhythm detection at 120 BPM using quarter-note and eighth-note input for MIDI is highly accurate, as expected (Figures 7-8). For both outputs, the rhythm– and pitch–notation viewed through MuseScore is one-to-one with the original audio inputs–with occasional unconventional accidental choices, though these still represent the correct pitches. With exact note onsets, offsets, and durations, this is the ideal test case for rhythmic transcription. There is little-to-no need for quantization, which also reduces the amount of algorithm-made errors. However, both examples of real guitar struggle to achieve a consistent level of rhythm detection, displaying errors in note spacing and timing.

In contrast, Figures 9 and 10 introduce the challenges associated with real guitar recordings. Miscalculations earlier on in transcription can lead to a cumulative timinf drift. Additionally, limitations in the program's algorithms to normalize audio, smooth pitch, and detect natural starting and stopping points for the notes also make it more difficult to achieve accurate rhythmic transcription. Much of the extraneous noise is also picked up and transcribed into notes. This can be seen in both Figures 9 and 10 as the clusters of notes with irregular rhythms and pitches that don't align with an ascending scale.

Despite the program's shortcomings with rhythm detection, it is still possible to see–and hear–contours of the initial input audio amongst the barrage of noise. More rigorous noise filtering, better pitch-smoothing, and note duration quantization can result in the real guitar accuracy being similar to that of the MIDI.

References

[1] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Onsets and frames: Dual-objective piano transcription. *arXiv preprint arXiv:1802.06182*, 2018.

[2] Peter Li Juan Pablo Bello Jong Wook Kim, Justin Salamon. Crepe: A convolutional representation for pitch estimation. *Music and Audio Research Laboratory, New York University, Center for Urban Science and Progress, New York University*, 2018.

[3] Anssi P. Klapuri. Automatic music transcription as we know it today. *Journal of New Music Research*, 2004.

[4] Google Magenta. Transcription with transformers, 2021.

[5] James A. Moorer. On the transcription of musical sound by computer. *The MIT Press*, 1977.

[6] Daisuke Niizumi, Daiki Takeuchi, and Yuki Mitsufuji. Byol for audio: Self-supervised learning for general-purpose audio representation. *arXiv preprint arXiv:2110.14449*, 2021.