Write On Cue

Shivi Jindal, Grace Li, and Deeya Patel

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—**A system capable of automating the process of transcribing flute performances into a music score that is available to music composers, students, and hobbyists to utilize on a web application. The web application takes in a recorded flute signal and invokes a backend pipeline that filters and segments the signal by note (using Fast Fourier Transforms, Short Time Energy, and RMS), performs audio and pitch detection (with frequency mapping and note onset/offset detection), encodes this information into a MIDI file, and converts the MIDI file to sheet music.

***Index Terms*—Butterworth Filtering, Fast Fourier Transform, Generative AI, Harmonic peak detection, MIDI conversion, Music Information Retrieval (MIR), Short-Time Energy, Spectral analysis**

I. Introduction

Music transcription is a crucial tool for musicians who want to document their compositions, analyze performances, or share their work with others. However, manual transcription can be time-consuming and requires significant expertise. Our capstone project aims to bridge this gap by developing a software-based music composition system that transcribes flute performances into sheet music in real time.

This system is designed to help musicians capture their improvisations, rehearsals, and live performances without the need for manual transcription. The core of our implementation is a signal processing pipeline that analyzes flute audio, extracting pitch and rhythm based on a user-defined BPM. The processed data is then converted into digital sheet music, providing an intuitive and accessible way for musicians to view their transcriptions.

To ensure high transcription accuracy, our approach will focus on precise frequency and pitch analysis of the flute signal. The system will be packaged as a web application, offering a user-friendly interface where musicians can play their flute and instantly see the generated notation. This software-based approach also provides flexibility, allowing us to experiment with different signal processing techniques to refine accuracy.

As a stretch goal, we plan to incorporate a generative AI feature that suggests potential next notes to assist composers in expanding their musical ideas. This feature would help musicians develop compositions interactively, making our tool not just a transcription system, but also a creative aid for music composition.

With this project, we hope to create a powerful and accessible tool that enhances the way musicians engage with their music, whether for practice, composition, or performance analysis.

II. Use-Case Requirements

Write on Cue should be a seamless and intuitive experience where musicians, students, and educators can find the transcription process effortless and reliable. To achieve this, it must consistently and accurately transcribe flute performances into sheet music with minimal effort from the user. Given that manual transcription is time-consuming and requires specialized skills, it must provide an accessible and efficient alternative by ensuring high accuracy, low latency, and adaptability across various musical styles and environments.

A. Accuracy

This application should accurately determine the rhythm and pitch of notes played, using an inputted BPM and time signature, with at least 95% accuracy under standard playing conditions.

The system's accuracy will be tested under various conditions, including different flute types, articulation styles, and background noise levels that simulate environments flutists play in. Testing will take place in both controlled lab environments and real-world settings. To enhance accuracy, we are collaborating with three flutists and faculty from the School of Music, including Professor Dueck and Professor Almarza, to gather real-world flute recordings and expert feedback on our system performance.

The system must handle variations in recording quality, instrument tone, and playing techniques while maintaining accuracy. It should also be adaptable to different music genres, ensuring broad applicability across user needs.

B. Latency

For an efficient user experience, the system must generate the first transcribed note within 3 seconds of receiving the audio input. This ensures minimal delay in feedback, allowing users to interact with the transcription in near real-time.

C. Output Format

The application must generate a digital music file (MIDI) based on the transcribed notes and then convert the MIDI data into a readable music score within the web application using MuseScore. This dual-format output ensures compatibility with a range of music editing and notation tools.

D. Public Health, Safety, and Welfare Considerations

Our flute transcription system enhances public health by supporting creative expression through music learning. With an attachable microphone and a software-only pipeline, it is affordable and safe to use. Our system also promotes welfare

by lowering barriers to music production, as it can be made accessible online for musicians, students, and educators to use.

E. Global, Cultural, Social, Environmental, and Economic Factors

Write on Cue aims to make music transcription more accessible to diverse musical communities, including amateur musicians, educators, composers, and students from various cultural and social backgrounds. This benefits people who may not have the technical skills or resources to manually transcribe music and allows individuals to better engage with music across a variety of cultural contexts. For example, in communities where formal music education is less accessible, our project can provide a more equitable way for musicians to preserve and share traditional flute music, irrespective of whether they are classically trained. Additionally, socially, this allows musicians from different backgrounds to contribute their musical expressions and allows for easier preservation of musical heritage.

By streamlining the transcription process, our project reduces the dependency on costly manual transcription services, which lowers the overall cost of producing sheet music. Also, we are designing our project on a web app, which maximizes accessibility and encourages a cheaper and more widespread music education. By offering an affordable or free version, the application ensures inclusivity, supporting musicians across various economic backgrounds.

By meeting these requirements, Write on Cue will serve as a powerful transcription tool that enhances the workflow of musicians, students, and educators while considering broader social, cultural, and environmental impacts.

III. Architecture and Principle of Operation

Our system has four major subsystems that integrate into an overall pipeline for transcribing flute recordings into sheet music: preprocessing & calibrating the input signal, segmenting the audio, applying rhythm and pitch detection, and generating the sheet music onto our web application. We are leveraging principles of signal processing, Fourier analysis, and web application development to ensure accurate and efficient transcription.

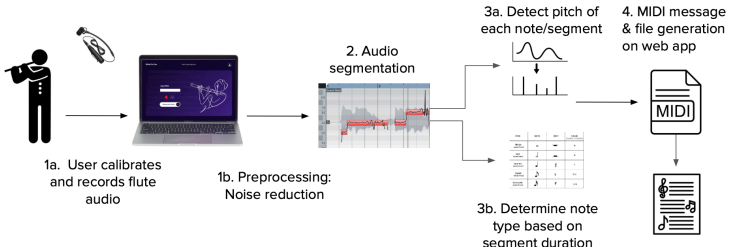
Our overall physical system is described in (Fig. 1a). At the start of the user's practice session, the user will attach the Behringer CB100 Gooseneck Condenser Microphone (Fig. 1b) using a clamp on the end of the flute and plug the microphone into the laptop using the XLR to USB C adapter. This specific microphone will make sure that there will be a clear and high-fidelity audio input that our system can use and analyze. The user will then create a new account or log into an existing account in our Write On Cue web application.

Using the microphone, the user will capture two audio signals - the background audio and the flute audio. As the user is playing and recording their flute audio, they have an option to listen to a metronome set at a default of 60 BPM and adjust it accordingly. The user will then be able to click the 'Generate Button' on the web application once the flute audio, background audio, and BPM they played at is recorded into our web application. The background noise audio is used for calibrating the flute audio, which then gets preprocessed. In our preprocessing step we are using engineering principles like bandpass filtering to retain flute-relevant frequencies, and spectral subtraction and adaptive filtering techniques to remove the background noise from the flute audio.

We then segment the flute recording into distinct notes using a sliding window root mean square (RMS) approach with a 10 ms window. A note onset is detected when there is a steep increase in amplitude (≥5 dB) lasting at least 100 ms. Also, we do frequency domain analysis via Fourier transforms to help refine note transitions.

For our pitch detection system a Fast Fourier Transform (FFT) is applied to each segmented note to extract the dominant frequency. The highest-amplitude frequency bin determines the fundamental frequency, which is then mapped to a MIDI note. To classify note durations (whole, half, quarter, eighth), we analyze amplitude stability. A note is considered active if its amplitude remains above 20% of the segment's maximum. The duration of active segments, combined with the BPM input from the user, determines note lengths.

The processed notes (pitch and duration) are encoded into a MIDI file using the MIDO library. The MIDI data is sent to MuseScore via an API to generate sheet music, which is displayed on the web application's front-end and stored in an SQL database. On the web application the user is also able to view their past transcriptions stored on the website. When the user is done practicing, they will be able to compactly pack up our system by detaching the microphone and adapter and logging out of their profile on the web application.



(a)

(b)

Fig. 1. Physical system separated into four major subsystems (a) overall system. (b) Behringer CB100 Gooseneck Condenser Microphone that records the flute and background audio and can be plugged into a laptop.

The subsystems work in parallel to be able to provide the user with a transcription in almost real time. Our overall high-level block diagram illustrating the complete architecture can be found at the end of this document (Fig. 10). It breaks down each of the four subsystems in greater detail and is explained thoroughly in Section VI.

IV. Design Requirements

Overall, our system will be following a pipeline of calibration, filtering out any noise, performing pitch and rhythm algorithms, writing the information into a MIDI file, and then sending it back to the web application. To first ensure that we have a clear signal, we need to execute a calibration step to ensure that the flautist is playing at a loud enough volume. To do so, we will require the flautist to record the surrounding environment noise to determine the baseline decibels. Afterwards, we will be making sure the flautist is consistently playing at least 20 decibels (dB) louder than the outside noise. If the flautist is playing at a volume less than this threshold difference of 20 dB, then the web application will signal to the flautist to play louder. To test this, we will practice playing the flute at three different relative volumes: quiet, medium, and loud. We will be testing these three different volumes in three different environments as well, similarly being quiet, medium, and loud. To test if the web app is accurately detecting the difference correctly, we will be using an outside library in python to determine the dB of both the environment and of the flute recording, from the microphone, and see if the web application output is within +/- five dB from the outside library measurement. We are ensuring that we have a clear signal to exclude any noise being included in the transcription of the recording.

After determining that there is a relatively clear signal from the flute recording, we will be filtering noise by utilizing a butterworth high pass filter. Then this filtered signal will be used to conduct pitch detection on the audio file by performing fourier transforms with python functions like SciPy. To test this feature, we will be playing sheet music with basic notes, like a scale, and comparing our application's output with the notes the flautist played. We want our outputted pitches to match the sheet music with at least 95% accuracy. We are aiming for a high accuracy to improve user experience as incorrect transcriptions will not satisfy our use cases of creating a seamless experience.

Next, we want to perform audio segmentation when a new note has been played. We know a note has changed when either the frequency has changed, signaling that the pitch has changed, or there is a sudden increase in amplitude, signifying that a new note, with the same pitch, has likely been played. Using this information, we can conduct audio segmentation by determining when a frequency has changed in time, reusing the results from pitch detection, or check if the peak of the signal has changed by at least five decibels and if it surpasses this threshold, then a new note has been played. To further reduce any noise, we will also be implementing a threshold of 20% of the maximum amplitude to be identified as one note. After reviewing some preliminary results, we noticed that the max amplitude tends to be right when the flute begins before stabilizing at around 20% of the maximum. To be more accurate, we will utilize this stable value as our minimum threshold to determine the continuous length of one note. To test if we are accurately segmenting the audio, we'll be manually determining when a note has changed in time and comparing the output of our system to this manually determined time. Ideally, our system output should be within +/- 0.1 seconds of the correct time. We want our output to match the sheet music by at least 95%. As aforementioned, we are aiming for a high accuracy to improve user experience as incorrect transcriptions will not satisfy our use cases of creating a seamless experience.

V. Design Trade Studies

When designing our flute transcription system, we considered various implementation approaches, balancing factors such as performance, efficiency, scalability, accuracy, and user experience. The major design tradeoffs involved deciding between a hardware- vs. software-based signal processing approach, determining the best way to filter/calibrate the flute signal, automatically detecting the input signal's tempo vs. integrating a metronome for rhythm consistency, and using a sliding window vs. audio segmentation technique for pitch and rhythm detection.

A. Signal Processing: Software vs. Hardware-Software Approach

Initially, we considered implementing a combined hardware-software pipeline, in which a hardware bandpass filter would handle removal of frequencies outside of the flute's frequency range; and a microcontroller (such as an Arduino or Raspberry Pi) would perform the ADC conversion,

use a thread-based approach to perform a Fast Fourier Transform for frequency analysis, and send necessary information about the signal to our computer via UART serial transmission. We also explored designing a custom PCB for taking in the flute signal and preprocessing before ADC conversion on a microcontroller. This would have involved designing and soldering the circuit with an IC microphone and bandpass filter. However, due to the computation-heavy nature of signal processing algorithms such as adaptive filtering and FFT, running these processes on a microcontroller would introduce significant performance bottlenecks and prevent us from achieving our goal of a 3-second latency between the user's submission of the audio input and sheet music generation. Additionally, the process of manufacturing a custom PCB would be complex, more costly, and make it more difficult to iterate upon the design without much improvement to our transcription accuracy.

A purely-software based approach allows us to leverage computational power from a laptop CPU, which is much more efficient for both filtering and FFT. Moreover, this approach provides us with more flexibility in terms of additional filtering approaches, such as adaptive filtering, on top of the Butterworth filter and spectral subtraction. Specifically, we will be able to easily integrate our system with a variety of open-source software tools, such as:

- SciPy: Access to robust filtering and FFT tools, allowing for efficient and accurate spectral analysis.
- Mido: Python library for handling MIDI file generation after rhythm/pitch detection is complete.

Overall, the pure software approach is inherently more scalable and accessible, as it eliminates the need for additional physical components (besides a microphone) and can be distributed as a standalone application.

B. Rhythm Detection: Tempo Detection vs. Metronome

To meet our design requirement of achieving 95% rhythm detection accuracy, we needed a way of tracking the tempo of the audio. We looked into detecting the tempo via beat tracking, but a metronome ensures a steady tempo. Though this constrains musicians who vary their tempo throughout a piece, a system without the metronome would need more sophisticated rhythm detection (such as beat-tracking) to handle tempo fluctuations, especially if the fluctuations were not intentional. Including a metronome also reduces cognitive load on musicians by providing a reference tempo, which enables us to more easily classify the duration/type of a particular note.

C. Audio Segmentation vs. Sliding Window FFT

Originally, we intended to transcribe the audio in real-time. However, this is no longer a priority– after speaking with musicians from the School of Music, we found that it is instead more practical for a user to be able to upload an audio file and receive a transcription. Given this change, we have opted for an approach where the recording is processed at once after the musician finishes playing. Instead of using a sliding window FFT, which may introduce boundary issues (in detecting note onset/offset), we will first segment the audio and process each segment independently for rhythm and pitch detection.To do so, we will apply a sliding window STE (Short-Time Energy) for note segmentation. This is useful for ensuring that each segment contains a full note before performing FFT and avoids splitting the notes across windows. Applying sliding window FFT prior to segmentation can split notes across windows, which makes it more difficult to detect clean frequency peaks. Instead, we will perform the FFT on multiple segments in parallel, which still allows us to detect the pitch in an efficient and more clean way. Though STE-based segmentation first requires some extra processing, an improved pitch detection accuracy outweighs the cost of some additional processing time.

VI. System Implementation

Our system comprises four major subsystems that integrate together into one overall pipeline as mentioned in Section III. In this section, we provide a summary of each subsystem– namely, the signal preprocessing and calibration, audio segmentation, rhythm detection and pitch detection, and web application/user interface. The first stage of the pipeline requires the user to log into our web application and record a flute signal through a Gooseneck microphone, which then triggers the signal preprocessing steps– bandpass filtering and spectral subtraction– to obtain a clean signal for audio segmentation. Next, we focus on audio segmentation, where we use a sliding window approach and Short Time Energy to detect energy spikes in the signal to detect note onset/offset. Once we have obtained our segmented audio, we will use a multithreaded approach to process multiple segments in parallel. For each segment, we will then perform rhythm detection to classify the note (whole, half, quarter, eighth, etc.) based on its duration and BPM rate. Each segment will also undergo pitch detection via a Fast Fourier Transform (FFT), which is then converted into a MIDI number and encoded as part of a MIDI file that aggregates all the MIDI notes. The MIDI note is then saved into the web application's database, which then calls the MuseScore API in order to translate the MIDI encodings into a sheet music score to be displayed on the web app's frontend. The web application enables the user to view the most recently generated music score along with all previous ones they have generated.

A. Subsystem A - Signal Recording, Calibration, and Preprocessing

Firstly, we decided to use the Behringer CB100 Gooseneck Condenser Microphone to capture a clear recording of the flute signal by attaching it to the end of the flute using a clamp. The microphone will then connect to a laptop to transfer the audio directly to the laptop to be uploaded to our website for any necessary preprocessing before being transcripted. We decided to go with a gooseneck microphone as it allows us flexibility in moving the microphone in conjunction with the flute– too close may result in picking up a distorted signal, and too far may be lost by noise.

For our pipeline, we require the user to first upload a recording of their background noise (η) in addition to a recording of the flute signal with the same background ($\eta + \epsilon$). These recordings pass through a calibration step triggered by the web app, which allows the user to proceed if the flute signal, ϵ, is at least 20 dB higher in amplitude than the noise-only (η) recording. As they record, they are also able to play a metronome (set to 60 BPM by default), which they then have the option to adjust in real-time via a slider on the user interface.

After passing this calibration step, we then pass the recording of the flute signal through a Butterworth bandpass filter, which allows us to preserve all frequencies between 260-2096 Hz for a cleaner signal. Depending on how well we are able to detect the Short Time Energy peaks, we may also need to implement spectral subtraction (allowing us to subtract background noise from the recording) and adaptive filtering (to detect and eliminate background noise levels that vary throughout the audio).

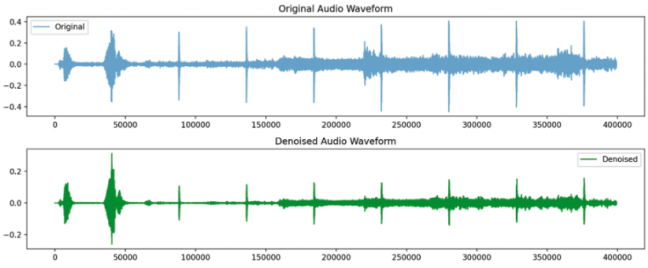


Fig. 4 Diagram demonstrating the original audio waveform before and after applying a Butterworth bandpass filter.

B. Subsystem B - Audio Segmentation System

The audio segmentation system will be an interface implemented within software. The intention of this subsystem is to divide the audio signal into segments, where each segment will denote a new note played by the flautist. This additionally will allow us to transcribe notes in parallel, improving the latency. To implement this, we plan on incorporating a sliding window of the root mean square of every 10 ms within python. This will be utilized to find steep increases in energy, or when the amplitude of the wave has increased quickly. To prevent any mischaracterization of multiple notes when it is actually one note, we will be having a threshold where 100 ms has had to elapsed so one steep increase in energy, defined as an increase of five dB, is not transcribed into multiple different note changes. After determining the times where there has been a steep increase in the energy or regions of interest, we will divide the audio into segments between these time intervals. Additionally, we will be simultaneously using times the frequency has changed, which can be determined when viewing the signal with a fourier transform using python libraries like SciPy, and using those to determine note changes as well since this can be used to determine if a note has changed in pitch.

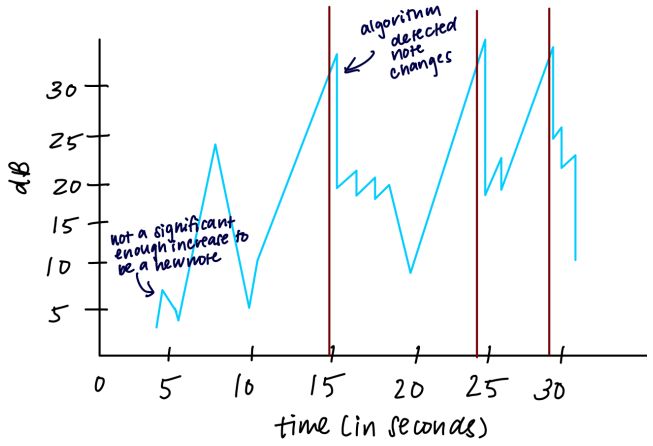


Fig. 2. Diagram demonstrating how we will be detecting a new segment or a new note change. Additionally, it depicts peaks that are not steep enough, i.e. less than a 5 dB change, that would not count as a new note.

C. Subsystem C - Pitch Detection System

The flute has a range of three octaves, which spans over C_4 to C_7. In order to determine which frequency each note corresponds to, we will be performing a Fast Fourier Transform on each of the note segments, where we process multiple segments in parallel using threading in Python. For each segment, we perform a Fast Fourier Transform (FFT) that converts the signal from time to frequency domain, producing a sequence of frequency "bins". Each bin represents a small frequency range (~10Hz), and our algorithm scans the bins to find the one with the highest amplitude. Once the strongest frequency is identified, it is converted into a MIDI number using the following formula:

$$m = 12 \times log_2(f_m/440\,Hz) + 69$$

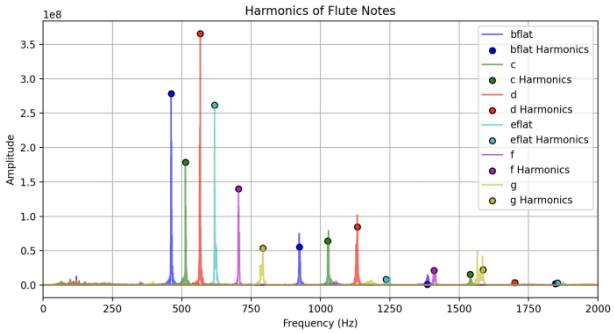


Fig. 3. Diagram demonstrating resulting harmonics from performing Fourier Transforms from individually recorded notes. The fundamental frequency of each note will allow us to distinguish between the notes and assign them a MIDI number for the MIDI file encoding.

Algorithm 1 Single Thread of Pitch Detection

Require: $b_{width} = \frac{f_s}{size_{FFT}}$
$b_{FFT} \leftarrow FFT(signal, size_{FFT})$ ▷ Compute FFT and get all FFT bins
$m \leftarrow |b_{FFT}|$ ▷ Get magnitudes of frequency components
$A_{max} \leftarrow 0$
$f_{best} \leftarrow 0$
for each bin i in b_{FFT} **do**
  $a \leftarrow m[i]$ ▷ Get amplitude of current bin
  if $a > A_{max}$ **then** ▷ Update max amplitude and frequency
    $A_{max} \leftarrow a$
    $f_{best} \leftarrow i \times b_{width}$
$n_{midi} \leftarrow 69 + 12 \times \log_2(\frac{f_{best}}{440})$ ▷ Convert to MIDI Number
return m

Fig. 4. Pseudocode for a single thread of pitch detection using FFT.

D. Subsystem D - Rhythm Detection System

To determine the length of the notes at a basic level would be to check the length of the segmentation. However, since the segmentations could include rest times as well, to accurately transcribe the rhythm or note length, we play on first calculating what the maximum amplitude of the signal in the segmentation, from part D, is. Since we are performing preprocessing and filtering any noise or anomalies, we can likely assume here that the signal will stabilize after a certain point in time. After initial experimentation, this stabilization point looks to be around 25% of the maximum amplitude, which typically occurs when the flute has just begun to play a new note. As such, our calculations will only consider a note to have continued playing if it maintains an amplitude above 20% of the maximum. We will then calculate the duration of the signal that is above the threshold to determine the length of a single note. This will then be used in conjunction with the pitch to be sent to the web application via a MIDI file. These calculations will be made with basic equations in code and using libraries to read signals, like SciPy.

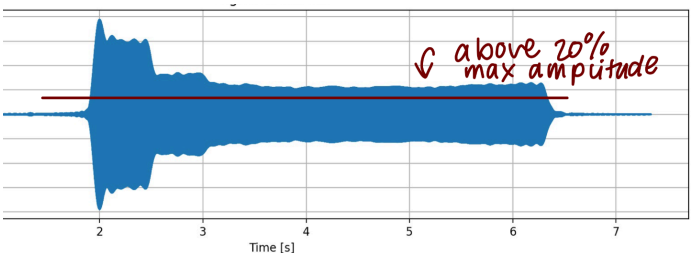


Fig. 5. Graph of the signal and depiction of the threshold for how we are determining the length of one note.

Algorithm 2 Rhythm Detection

for peak in segment *iteration* $= 1, 2, \ldots$ **do**
  Find max amplitude
end for
Calculate threshold to be 20% of max amplitude
for time in segment $1, 2 \ldots$ **do**
  Check if dB is above the threshold
  Stop if dB dips below threshold, signifies end of note
end for
Return total amount of time the signal was above threshold

Fig. 6. Pseudocode detailing the rhythm detection algorithm we will be implementing in the system.

E. Subsystem E - Writing to the Web Application System

The system overall will label the segments based on where they come in time. For instance, the first system will be known as *segment 1* and increase until there are no more new notes. The code will then cycle through the segments, in numerical order i.e. the order in which they are played, and take the pitch and rhythm, which was calculated for all segments at once in parallel and stored within the code, to write this into the MIDI file using the python package MIDO. The code takes in the BPM the user inputted and utilizes this tempo to encode when a note is “on” and when a note is “off” or when the note has begun and ended respectively. From the pitch detection system, we include the determined pitch, alongside the rhythm, into a class known as a message. The message is then attached to a header, which will all be combined to make the MIDI file. This information will then be uploaded to MuseScore to be converted into easily viewable sheet music. The website utilizes this MuseScore API to display this information on our website and store the transcription within our SQL database.

Algorithm 3 Writing to the Web App

import MIDO library
Create MIDI file
Create MIDI Track
Append header
for segment $= 1, 2 \ldots$ **do**
  Write notes message, including note tempo (BPM), velocity (length), and start time
end for
Save File
Send to MuseScore API

Fig. 7. Pseudocode detailing how the algorithm will write into a MIDI file and send the note information to the web application. .

F. Subsystem F - Web Application System

The architecture for web application is structured to handle and facilitate real-time audio processing, transcription, and user interaction while ensuring efficiency and scalability.

The user interface is developed using Django, integrating HTML, CSS, and JavaScript. Users can upload flute recordings, view past transcriptions, and interact with various features such as BPM adjustment and being able to view a Help Page. Once all the audio files are uploaded the user is able to view the transcription of the audio as well as optional new generated notes from our model. The backend is powered by a SQL database that stores user data, audio files, and transcription results, making sure there is efficient data management and retrieval.

A key feature of the system is the real-time adjustable BPM metronome, which runs as a separate thread to prevent interference with flute notes. This functionality is enabled through WebSockets, allowing smooth synchronization and user control over tempo adjustments. Additionally, the system includes an OAuth-based user authentication module, ensuring secure access and personalized experiences for multiple users. The architecture is designed to support multiple users simultaneously.

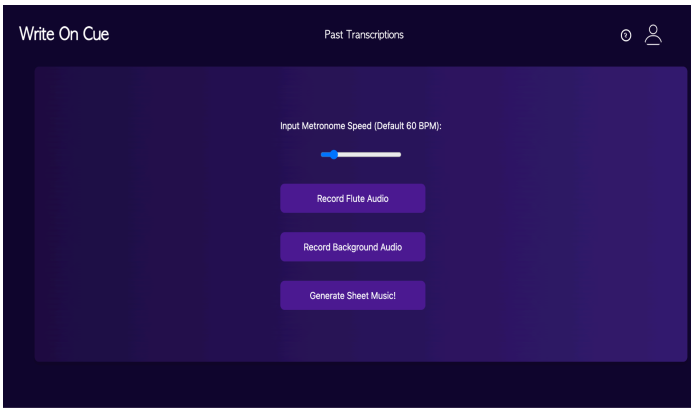


Fig. 8. User Interface of Write On Cue, featuring audio uploads, metronome control, and profile settings.

G. Subsystem G - Generative AI

Model Architecture

Our generative AI system will attempt to generate flute music by utilizing a Transformer-based generative model trained on MIDI sequences. We are planning on using a Transformer model because of its effectiveness in capturing long-term dependencies in sequential data, making it particularly useful for music generation. Music has intricate patterns that include harmony, rhythm, pitch variations, and melodic structure, which the self-attention mechanism of the Transformer is highly capable at modeling.

Data Preprocessing

The flute music is represented using a token-based system derived from MIDI sequences. MIDI provides a precise representation of musical elements, such as pitch, duration, and velocity, making it an ideal format for training a generative model. The MIDI files are converted into a series of tokens that encode musical notes, timing, and dynamics, which can then be processed by the Transformer model.

Feature Extraction: Prior to feeding the MIDI data into the model, we will need to perform several preprocessing steps. First, we extract individual notes, velocities, and durations from the MIDI sequences. Pitch normalization is applied to ensure that the generated output remains within the range of a typical flute. Rhythm encoding is another critical step, where we represent the timing of each note relative to a fixed BPM.

Model Training

Dataset: The dataset will need to consist of a combination of MIDI sequences from flute recordings. The MIDI sequences will be derived from public databases, while the flute recordings will either be sourced from the flutists we are working with or also from public databases like Classical Archives. The format of the dataset will need to follow the format of the MAESTRO dataset of piano music. The diversity of the dataset will make sure that the model learns a wide range of musical styles, tempos, and articulations.

Training Objective: The model will be trained using a next-token prediction approach. Given a sequence of notes and rhythms, the model is tasked with predicting the subsequent note or event in the sequence. This objective is useful for music generation, as it allows the model to learn the flow of musical ideas over time. We will also be using relative positional encodings, an enhancement over traditional absolute positional encoding. This allows the model to more accurately account for the relative timing between notes, rather than relying solely on their absolute positions in the sequence.

Loss Function: The loss function we will use in training is cross-entropy, which is commonly used for sequence prediction tasks.

VII. Test, Verification and Validation

A. Calibration and Noise Filtering

We will test our calibration and filters by testing the system in various environments with varying playing volumes. We plan on testing the recordings in noisy and loud; medium; and quiet surroundings with varying flute volumes of loud, medium and quiet, doing nine different conditions. For instance, testing a loud dynamic recording of the flute within a quiet environment, like a studio. This will help determine if our noise filters algorithm will effectively filter out the noise of the environment without affecting the flute recording.

These tests will verify that our design will not record notes that the flautist did not play by ensuring that noise does not affect the notes recorded.

Additionally, to help ensure that the flautist is playing at a loud enough volume after the calibration step, the system will be measuring the decibels of the environment, from the calibration recording, and the decibels of the test flute recording. To test that the decibels are accurately measured, we will be testing our results against an outside library function in python and ensuring that our system's results are within five dB of the outside library's to help maintain consistency. These tests will further ensure that noise is not transcribed into the generated sheet music.

B. Pitch Detection

Starting out, to test pitch detection we will be playing scales. Our system will then output sheet music and we will compare the scale being played to see how accurate our transcription is. For instance, when playing a simple C major scale, we would expect to see the ascending notes in the correct order. When we compare our generated sheet music to the scale, we want the notes' pitch to match the scale with a 95% accuracy. We desire a high accuracy to improve user experience and to ensure a simplified experience for new users when composing music.

Afterwards, we will be further testing the system by using more difficult flute compositions of known songs such as *Twinkle Twinkle Little Star* and comparing the application's generated music with the sheet music from music applications like MuseScore, still requiring 95% accuracy. These tests aim to satisfy accurate sheet music generation.

C. Rhythm Detection

As a simple test, we will first be playing three different length notes: a quarter note, a half note, and a whole note. Our system will then output sheet music and we will compare the results to the intended lengths played for the recording, needing at least 95% accuracy over multiple tests when identifying the lengths of notes. To further test the application, we will be conducting these tests at varying beats per minute (BPMs) to accurately identify note length and prevent aliasing. Similar to the pitch detection, we desire a high accuracy to improve user experience and to ensure a simplified experience for new users when composing music.

Afterwards, we will be testing the system with known flute compositions of songs such as *Twinkle Twinkle Little Star* and comparing the sheet music, from music applications such as MuseScore, to our generated transcription. When comparing the two, we are still aiming for 95% correct identification of note length. We expect some difficulty in terms of aliasing. For instance, if we identify two quarter notes as a half note instead of the two separate notes. From preliminary testing, we noticed that when a new note begins, there tends to be a spike in amplitude, so we hope to use a minimum difference threshold to accurately identify when a note has changed but has the same pitch. These tests aim to satisfy accurate sheet music generation, especially with the high accuracy measure of 95%.

D. Overall Transcription

We will be testing the overall transcription by recording and transcribing known sheet music to various songs of different genres and comparing the sheet music to our generated one. When comparing the pitch and rhythm, we want to ensure that it has a 95% accuracy in both matching the intended pitch as well as the intended volume. Furthermore, to ensure positive user experience, we aim to have the latency of transcription be less than three seconds. Thus from when the user is finished recording the flute audio to when they see the finalized, generated sheet music, this should ideally take, at most, three seconds. To test this, we will be testing compositions of varying lengths, such as a one minute song and a three minute song, and timing how long it takes to transcribe the final music. As we are planning on running the identification in parallel, we imagine that the transcription time will be significantly shorter than if we attempted it in serial.

We are aiming for this short latency as a long transcription time could hinder user experience as waiting to see results might cause individuals to lose inspiration or interest in creating the next measures.

Additionally, to gain user feedback, we will be testing this in common areas where flautists would be utilizing the device, like in a home environment or a studio, with flautists from the Carnegie Mellon School of Music flute studio, in collaboration with the School of Music.

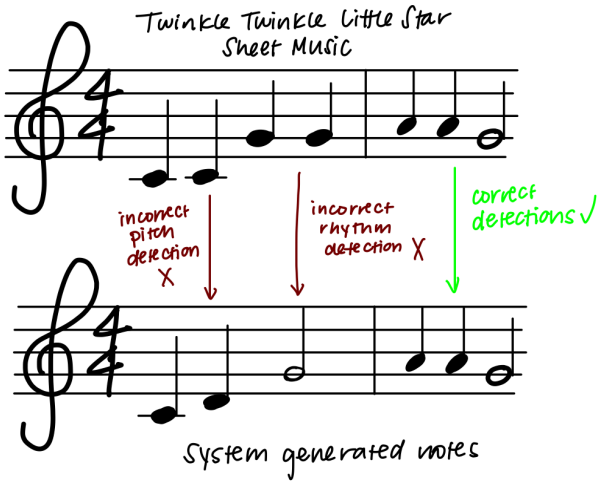


Fig. 9. Diagram demonstrating what the testing process will look like as it depicts what an incorrect rhythm and an incorrect pitch detection would be.

VIII. Project Management

A. Schedule

We have included the GANTT chart to outline our schedule. We are currently on track as we are in the process of implementing the individual subsystems of our project. We have the basics of website framework down, allowing individuals to log in with OAuth, upload files, and input a BPM. In terms of the transcription, we are currently experimenting with audio segmentation and using the information obtained from the fourier transforms and the amplitude of the signal to determine when there has been a note change. The main milestone for the web application will be having an easy to use website be able to take in a recording and create a transcription for the user. For the transcription, the main milestone will be to analyze the signal and convert it into the individual notes, with accurate pitch and rhythm. See end of document for our full schedule in Figure 11, which includes more detailed time frames of testing, implementation and design, as well as the final integration timeline.

B. Team Member Responsibilities

Each team member is responsible for a subsystem of the design. Shivi and Grace will be working mostly with the signal processing as they oversee the audio segmentation calculations and the detection of rhythm and pitch from the segmentations. We have overlapped these subsystems with team members because we believe that the two subsystems are extremely integrated and the collaboration between the two calculations help with smoother and faster detection times. Deeya will oversee the front end of the web application and establish authentication and security within the site. During this time, since Grace and Shivi's project is more frontloaded will be software support as Grace will help with the generation of the sheet music and Shivi will aid in the integration.

Audio Segmentation	Shivi + Grace
Noise Filtering	Shivi
Rhythm Detection	Grace
Pitch Detection	Shivi
Web Application Framework + Front End	Deeya
Integration of Signal Processing	Shivi + Grace
Gen AI	Deeya
Integration of Signal to Website	Shivi + Grace + Deeya

C. Bill of Materials and Budget

Our bill of materials can be viewed in Table II at the end of this document in Table I. We project that our system will be fairly inexpensive at a total cost of $50.29 currently.

D. TechSpark Use Plan

We do not plan on using TechSpark for our project.

E. Risk Mitigation Plans

The primary risk for this project is the audio segmentation step. When looking at the audio waves, there tends to be a lot of variation within the waves amplitude. This could be the result of outside noise or inconsistent tone with the flute. In turn, all these various peaks could interfere with calculating the audio segments, or when a new note has occurred. To mitigate these, we plan on using a sliding root mean square (RMS) equation to look for the energy of the wave, rather than solely relying on the amplitude. This helps reduce the impact of noise in the signal.

Another risk is the latency. We want to make this experience enjoyable for all users and musicians. Having long wait times in between recordings and the transcription visualization would hurt the overall experience, thus we are aiming for a less than three second wait time between the recording upload and the result loading. To mitigate a long latency, we plan on running our audio segmentations in parallel, reducing the amount of time long compositions would take to transcribe into sheet music.

IX. Related Work

This project shares similarities with MuseScore where individuals are able to convert MIDI files into sheet music. As such, we will be using this API for preliminary stages before developing our own function to read a MIDI file and generate sheet music within our own website.

Furthermore, there have been many past transcription projects for various other instruments in the past, but the project that is most similar to ours is *Transcriber* in which the group created an application to transcribe piano recordings into sheet music, using a combination of hardware and software. Our projects are similar in that they both take in musical recordings to generate visual sheet music but they different in the components, as we are solely using software to do our calculations, as well as the instrument, since we are focusing on the flute. We believe that this will help with complexity of the basic framework as the flute has a limited range of pitches it can produce.

X. Summary

Our website is designed to make the process of composing music easier and more accessible for all individuals by eliminating the tedious process of copying down the notes. To do so, we will be creating a web application that will take in a flute recording and transcribe the corresponding sheet music by analyzing the rhythm and pitch of the audio signal through various python libraries. Some upcoming challenges include correctly identifying where a note begins in audio segmentation and integration where we connect our signal processing methods to the web applications while simultaneously ensuring that our system meets the accuracy and latency requirements outlined in the use-cases.

Glossary of Acronyms.

BPM – Beats Per Minute
dB – Decibels
MIDI - Musical Instrument Digital Interface

References

[1] Ruffley, Wang, and Tarczynski, *Transcriber,* Accessed on Feb 26, 2025, [Online]. Available: https://course.ece.cmu.edu/~ece500/projects/f23-teamb5/

[2] Von Seggern, Ian. "Note Recognition in Python." *Medium*, Accessed on Feb 26, 2025 [Online]. Available: medium.com/@ianvonseggern/note-recognition-in-python-c2020d0dae24.

[3] "About MIDI — Mido 1.3.4.Dev6+Ga0158ff Documentation." Readthedocs.io, 2025, mido.readthedocs.io/en/latest/about_midi.html. Accessed 28 Feb. 2025.

[4] "Signal Processing¶." Signal Processing - SciPy Cookbook Documentation, scipy-cookbook.readthedocs.io/items/idx_signal_processing.html. Accessed 28 Feb. 2025.

[5] Classicalarchives.com, 2020, www.classicalarchives.com/newca/#.

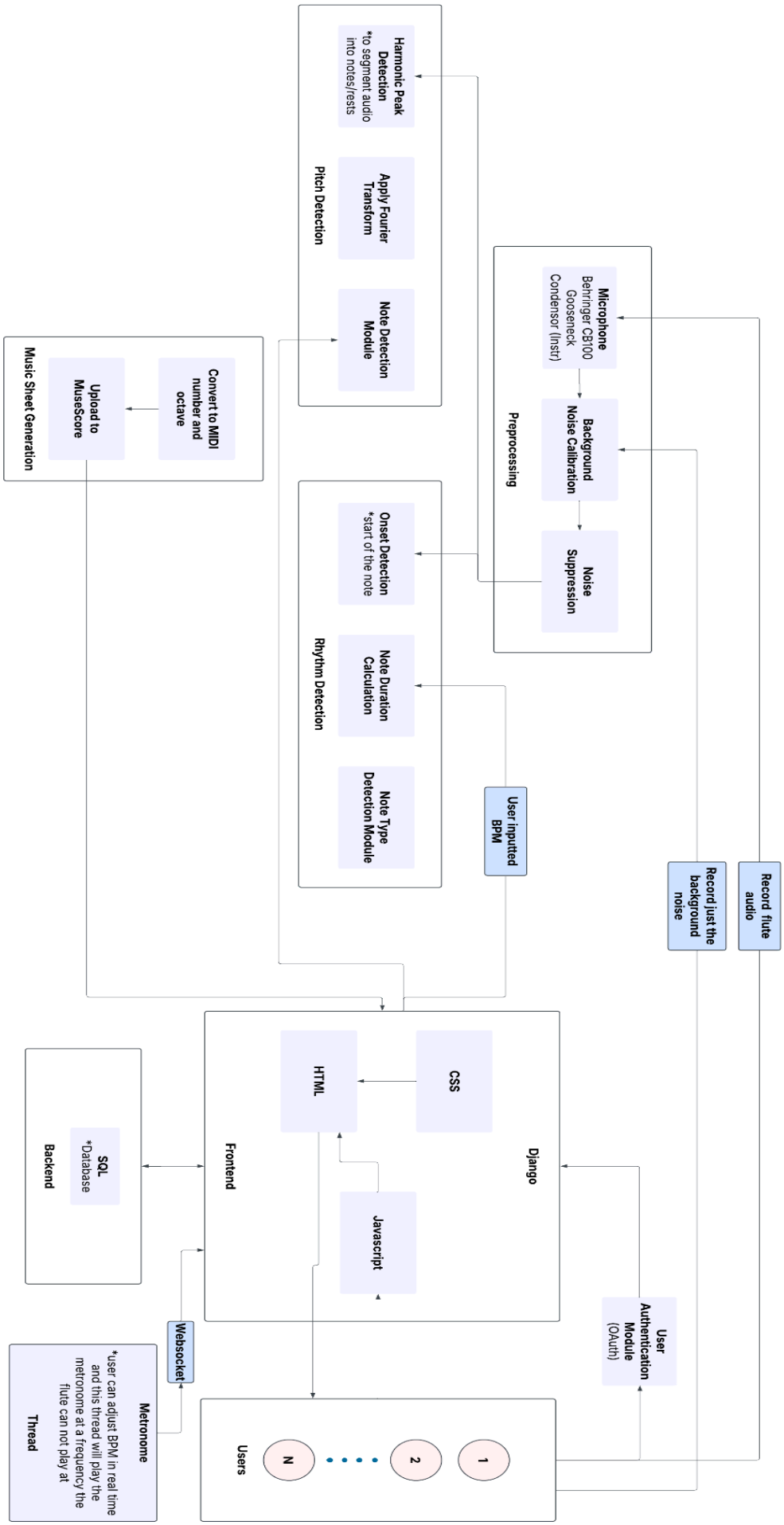


Figure 10: A full-page version of the system block diagram mentioned in Section III.

TABLE I: BILL OF MATERIALS

Description	Model #	Manufacturer	Quantity	Cost @	Total
Universal condenser microphone	CB 101	Music Tribe US	1	35	35
XLR to USB C Adapter	GOWENICvzt5g	GOWENIC	1	15.29	15.29
				Grand Total	50.29

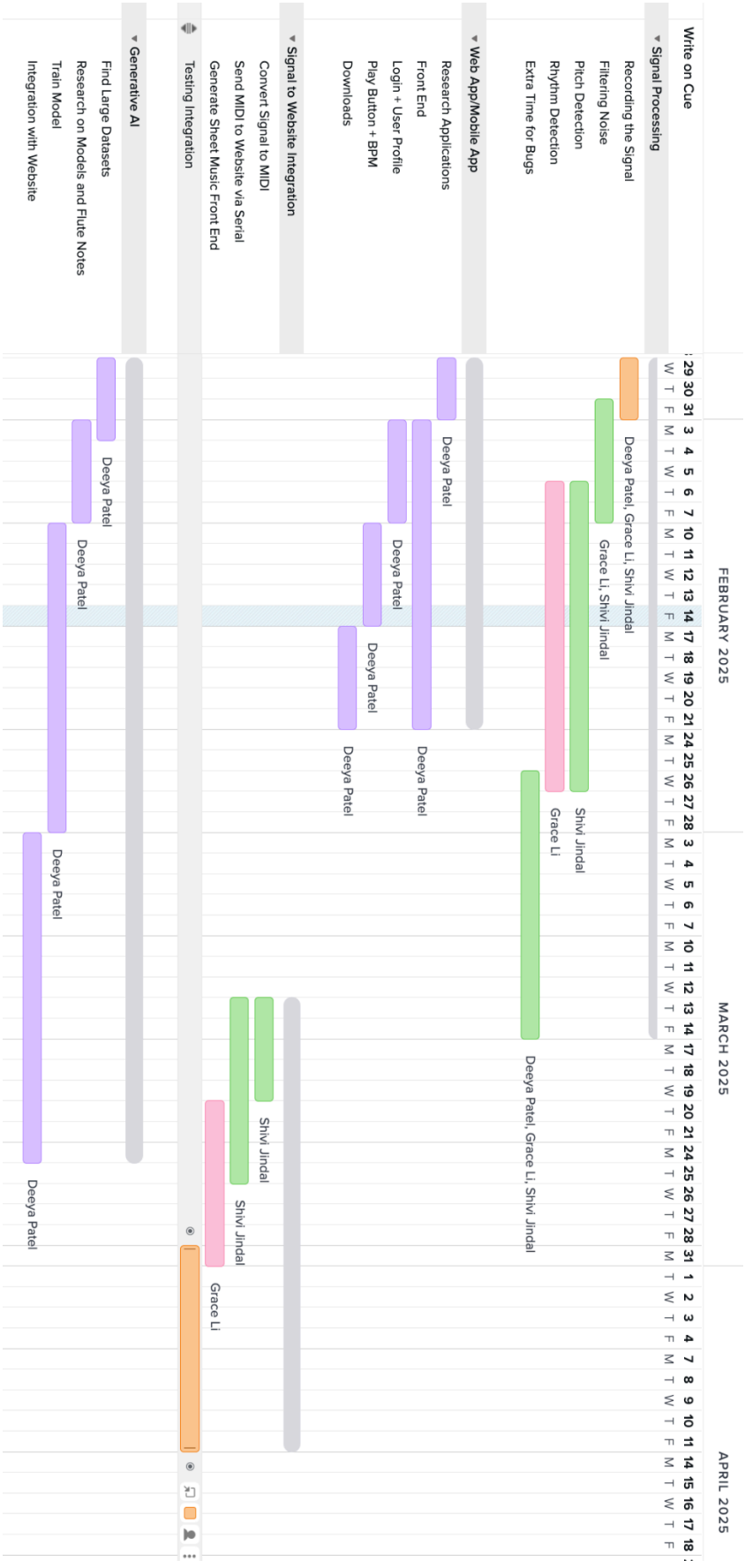


Figure 11: Schedule