ECE 445 FINAL REPORT: DYNAMIC VIOLIN FINGERBOARD ATTACHMENT

By

Adrian Ignaci

Kamil Waz

Sophia Wilhelm

Final Report for ECE 445, Senior Design, Spring 2026

TA: Manvi Jha

6 May 2026

Project No. 93

Abstract

The dynamic violin fingerboard aids self-taught violinists in learning correct finger placements and rhythms. Users upload a MIDI file for a piece they would like to learn and configure the piece settings. A dynamic LED display laid on top of the fingerboard then illuminates the locations where the user should place their fingers at the appropriate times. Potentiometers measure the position of a finger placed along any of the 4 paths (strings) on the fingerboard. These measurements are turned into an accuracy metric that is provided to the user via an LCD display. This allows us to collect information on how accurate the placement is, rather than a simple yes or no as to whether they play the right note.

Contents

1. Introduction

1.1 Problem

Most people would like to learn an instrument; however, not only are the instruments expensive, but the lessons are just as (if not more) costly. This also assumes lessons are even available where they live. For this reason, many people try to teach themselves how to play, either through experimentation or online resources. However, this path has a distinct lack of feedback that would help correct poor habits or otherwise incorrect playing.

1.2 Solution

Our project seeks to give those self-learning a violin an extra source of feedback with respect to finger placement (creating the notes) as well as the rhythm played. A dynamic LED display laid on top of the fingerboard would allow learners to better understand proper finger placement in addition to its relation to the specific note's duration.

Furthermore, by using linear/membrane potentiometers we measure the position of a finger placed along any of the 4 paths (strings) on the fingerboard. This allows us to collect information on how accurate the placement is, rather than a simple yes or no as to whether they play the right note.

To encourage building good habits and continuous practice, we would like to allow users to upload pieces they would like to learn. Thus, users will be allowed to upload files (MIDI) that can then be used on the fingerboard along with an adjustable tempo. This, paired with individual settings for full piece playthroughs and learning (only playing the next note after the user plays it) will help encourage good, accurate playing whilst making it fun.

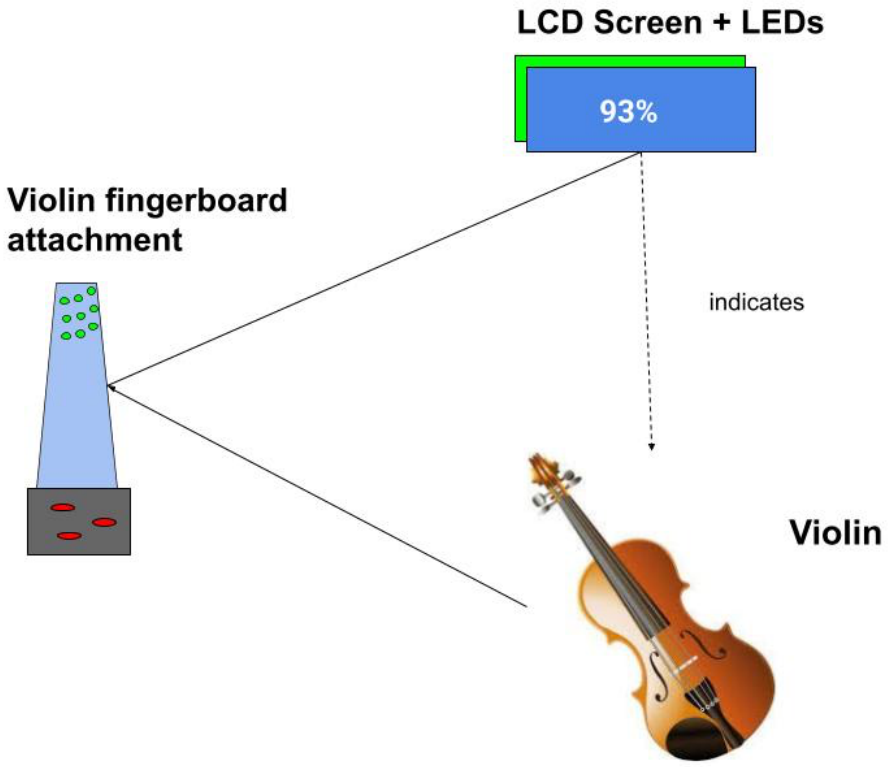

Figure 1: Basic Product Interaction

1.3 Requirements

This project will be considered successful if:

- The LEDs on the fingerboard turn on independently from each other in response to a signal from the control unit.
- A MIDI file less than 1 MB can be uploaded to the control unit and sets the control sequence for the fingerboard LEDs.
- Potentiometer membrane can detect finger placement within 2 mm and measurements are converted to an accuracy score that is provided to the user at the end of the piece.

If time permits, the following features will be added:

- Real-time accuracy feedback.
- Multiple songs can be uploaded to the controller at once and the user can choose which to play.

2. Design

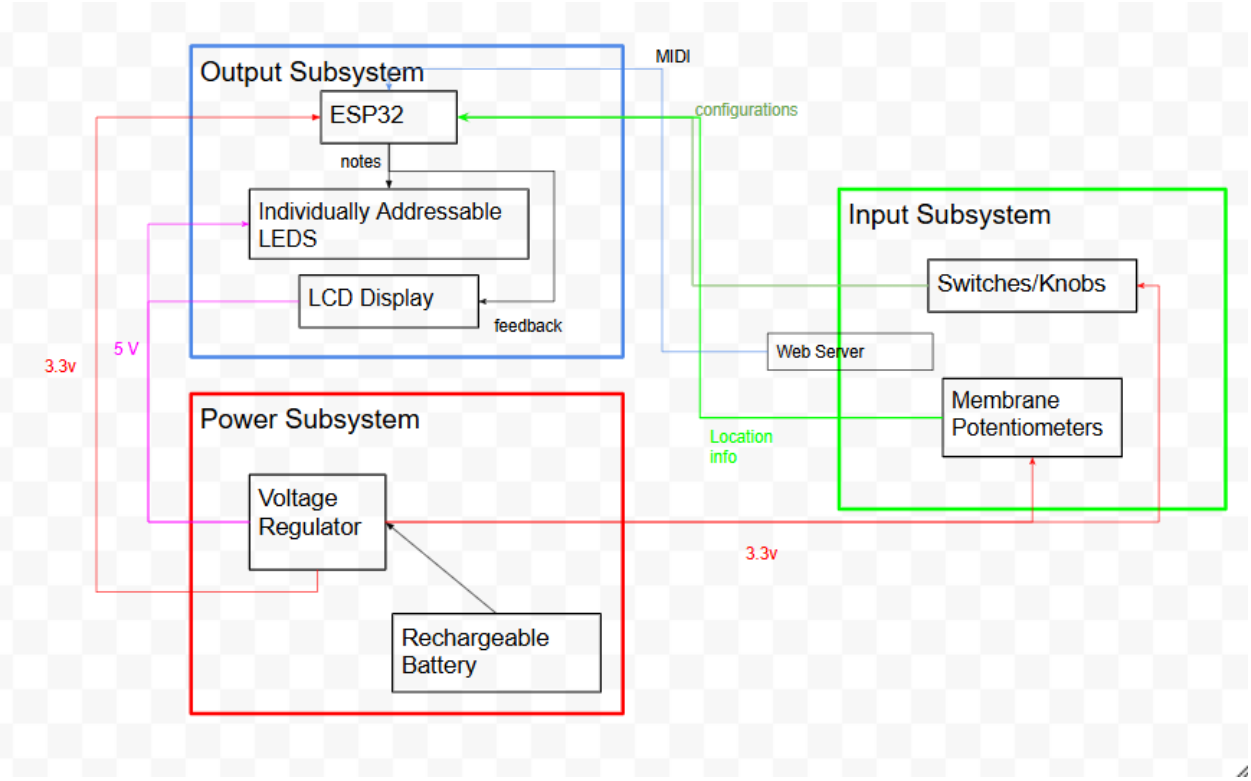

Figure 2: Block diagram of project's three major subsystems

Figure 3: Device placement on a full size violin using an elastic strap. Highlighting indicates the sensor strips. The box shown on the back is the main module enclosure (approximately 6" x 6" x 3"). Original photo from Amro Music [7]

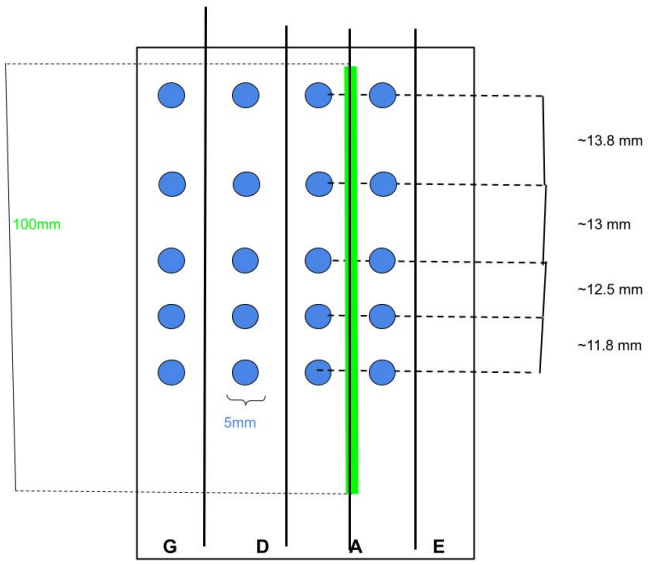

Figure 4: Sensor and LED placement on the fingerboard. Each string will have a sensor (green) and accompanying LEDs (blue). Note the logarithmic scaling of relative note distances.

The high level design consists of three major systems. The power system, as the name implies, is responsible for delivering and regulating the flow of power to the components in each subsystem. This is complicated by the existence of multiple separate, but interdependent, sub-circuits with different power needs. The output system encompasses the microprocessor and is how we provide feedback to the end user. Similarly, the input subsystem allows the user to interact meaningfully with the device. Without it, there would be no way to understand what the user is doing relative to our expectations or, as with the case of file uploads, what those expectations should even be.

The total weight of the entire project must not exceed 1lb to avoid disrupting the balance of the instrument or physically straining the user.

2.1 Power subsystem

The power subsystem will provide necessary power to all other subsystems and components.

It will provide the microcontroller with up to 500mA at 3.3V (as the ESP can spike current draw when running web servers), and the LEDs and LCD with 500mA at 5V. A voltage regulator will constrain the provided voltage to within the safety tolerances of the individual parts as needed. Due to the differing voltage requirements for each subcircuit, they must remain relatively isolated. So, an increase in power consumption from one subcircuit should not cause the voltage supplied to the microcontroller to fall below 3 volts at any point (Espressif Systems, 2019). Similarly, an increased consumption should never allow the voltage of the LEDs to fall below 3 volts (Feztek).

The battery pack provides the overall power; the voltage regulator helps isolate the two separate subcircuits (as they have different voltage requirements) while allowing for consistent current flow. Initially we desired to use a set of AA batteries due to the accessibility of them to consumers, however during breadboard tests we saw that it could not provide the necessary amount of power; we would use it to power on the ESP32 and see the voltage drop from 3.5v to approximately 1.2v. Thus we pivoted to a large rechargeable battery pack capable of supplying 5v and up to 5000mA.

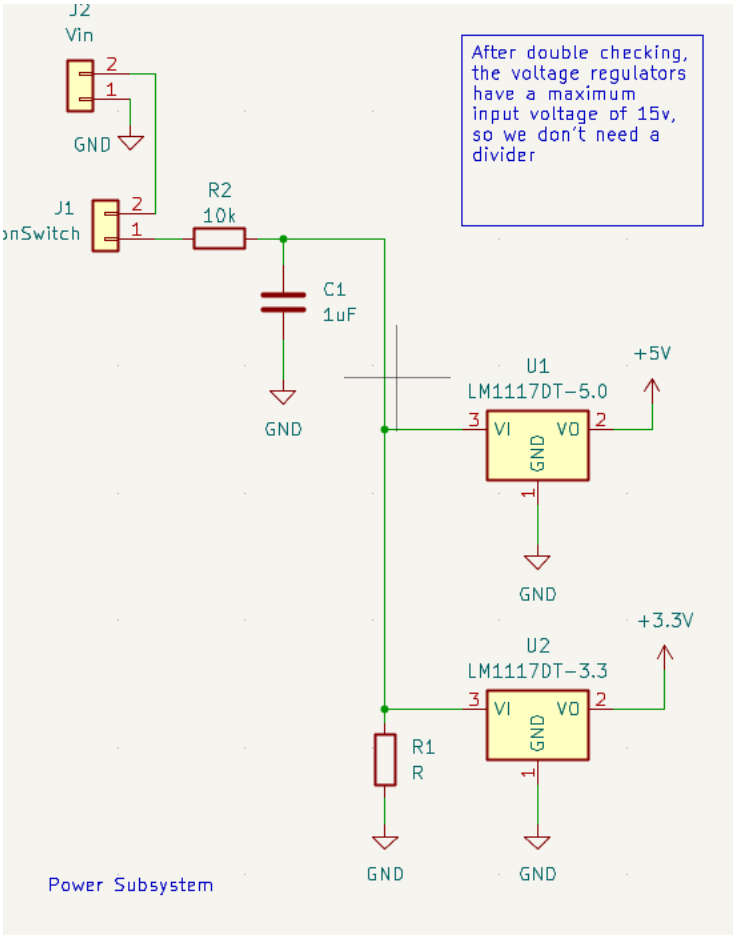

Figure 5: Power Subsystem Schematic

2.2 Input subsystem

The input subsystem is responsible for managing user input. This includes the buttons and switches located on the main-body unit that the user uses to control configuration settings. Also included are the membrane potentiometers mounted on the fingerboard that measure user accuracy. The transfer of information will be routed through a microcontroller.

The input subsystem must be able to detect the finger placement of the user within 2mm and convert this as part of an accuracy score. The ESP32, while part of the output subsystem, does take input via the web server. Thus, it should download a provided file within 10 seconds.

The switches and knobs will provide the output subsystem, specifically the ESP32 controller with the proper context under which to operate. It will allow the user to specify not only the tempo, but also what manner of play (progressive or run-through) to proceed with. Meanwhile, the membrane potentiometers will also interface with the ESP32 to allow data tracking and, in the case of progressive play, continue the piece when sufficient accuracy is achieved.

It is important to note that the membrane potentiometers must have pulldown resistors on the wiper/output pin. They are physically built such that the entire outer shell is a resistive material (to prevent shorting) and the less resistant wiper is suspended between the membrane without making physical contact until pressed. Thus, it will float unless grounded. It would be better to have a physical grounding pin on each, but to avoid soldering issues we opted to use the ESP32's internal pull down resistors.

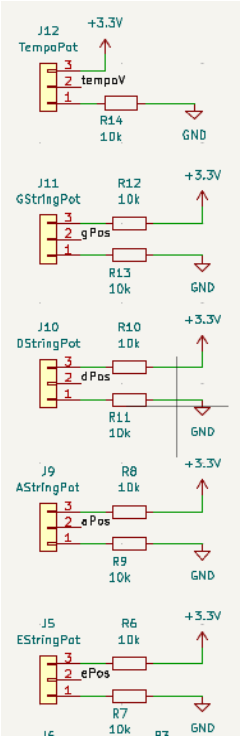

Figure 6: Schematic of external connections to sensors for each string and tempo control. Note that the ribbon sensors (membrane potentiometers) must be placed on the fingerboard and thus cannot be soldered on the main PCB

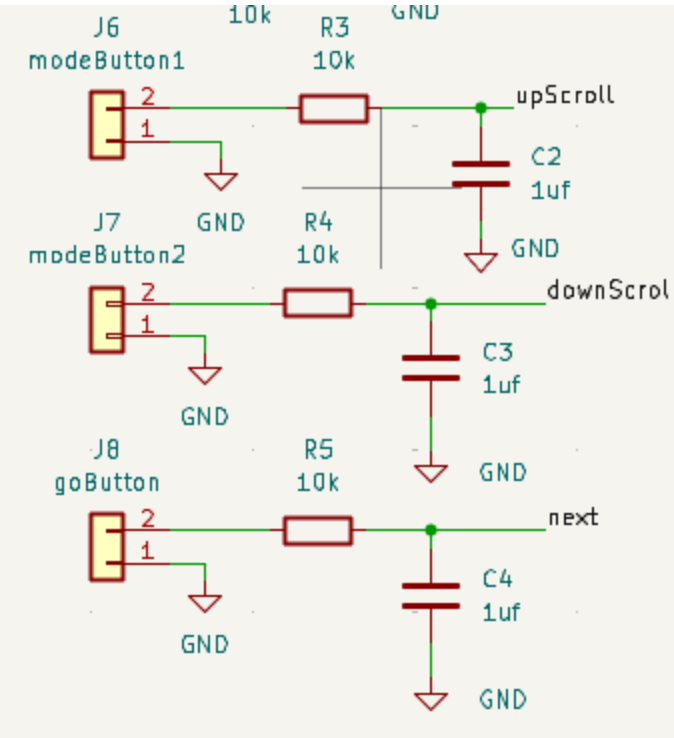

Figure 7: External connections to buttons used for configuring the device before use. Note the 10ms debounce circuit.

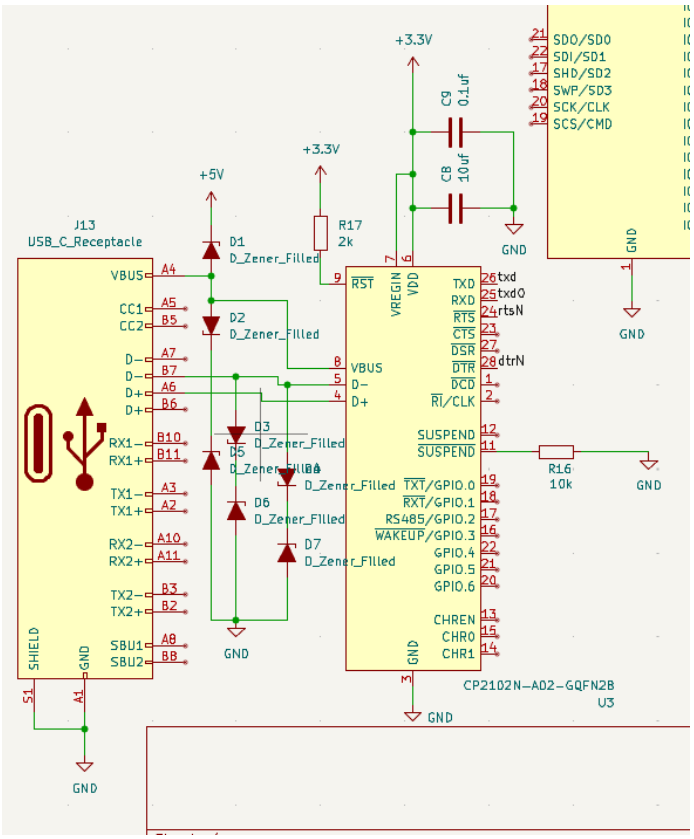

Figure 8: Custom USB-C compatible programming and file upload interface utilizing a UART-USB master bridge

2.3 Output subsystem

The output system is responsible for providing visual responses to the user. Individually addressable LEDs on the fingerboard guide the user through learning: 20 LEDs will be divided between the four strings to allow guidance for 5 notes per string (see figure 4). A Liquid Crystal Display makes setting configurations more user-friendly and shows statistical information taken from a playing analysis computed by the microcontroller.

In the context of musical performance, it is important to have a consistent tempo. Therefore, we want the latency between the microcontroller and the LEDs to be self-consistent, within a variation tolerance of 7ms. Changes to the user-input settings will be reflected on the LCD display in under 1s. Final stats will be shown on the display within 10s of the piece finishing.

The ESP32 is the brain of the project and will analyze all the data to be communicated back to the user. This includes reading MIDI files and telling the user what input to provide. For this reason, it will control the LEDs via custom mapping to show finger positioning and give feedback on the LCD when a piece is over.

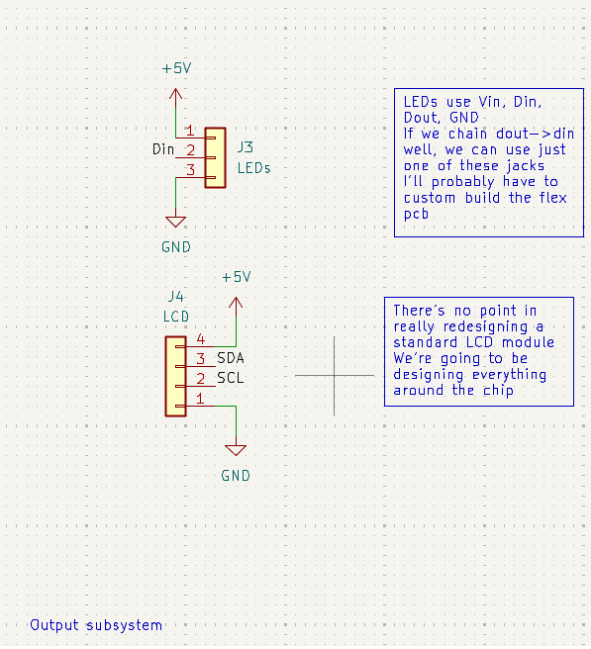

Figure 9: External pin connections to allow for LED and LCD control. Note that the LEDs will be on the fingerboard and thus cannot be mounted directly on the main PCB

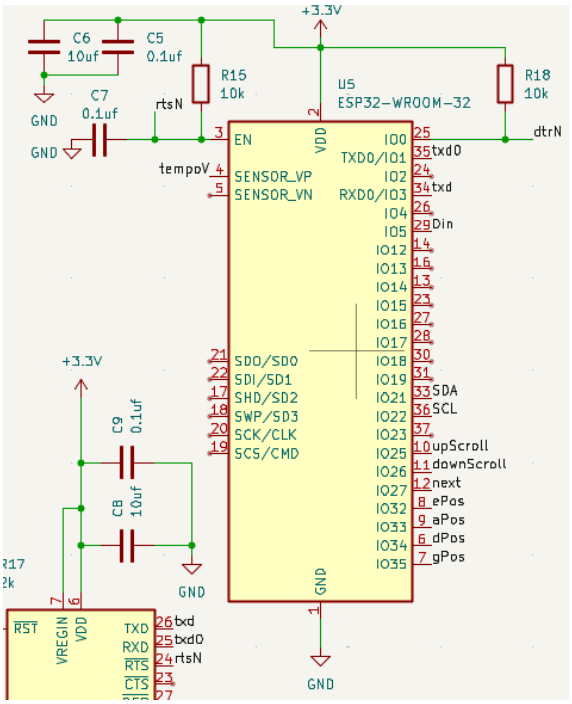

Figure 10: Connections to the ESP32 chip. Note the capacitors used to filter noise from the enable signal.

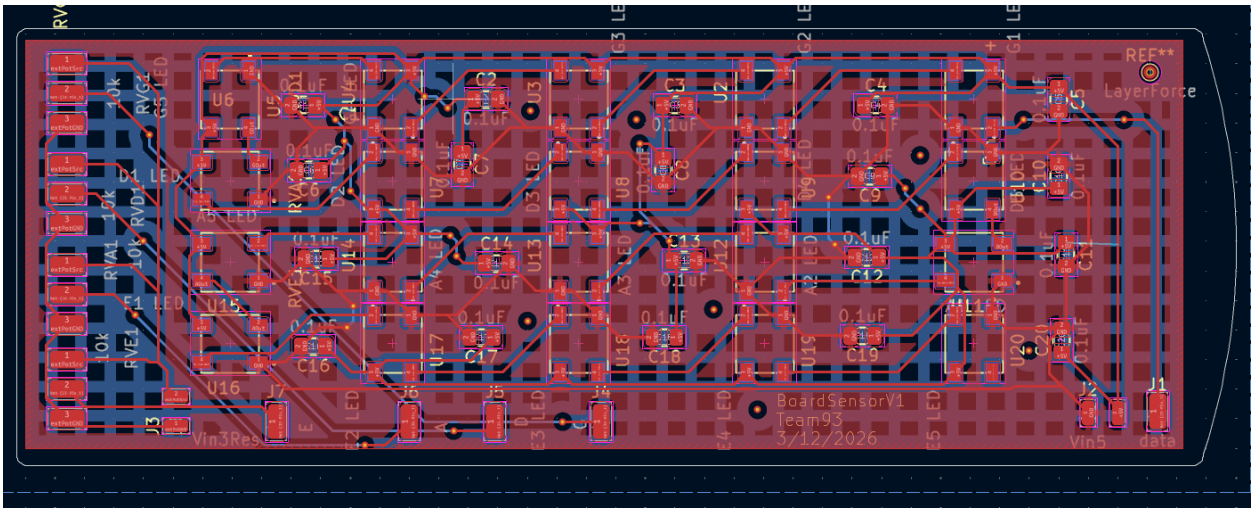

Figure 11: PCB Layout of flexible LED array

2.4 Software

The software processes all the inputs and thus had to be carefully constructed. Modifying the structure in any area could cause the entire system to stop functioning. The Arduino IDE, for which we used the supported C++, uses a setup and loop format. To keep the entire system running even after completing a piece, it was necessary to write much of the code in the looping section, but this also created many problems as moving to unique states based on user input became more complex.

2.4.1 State Movement

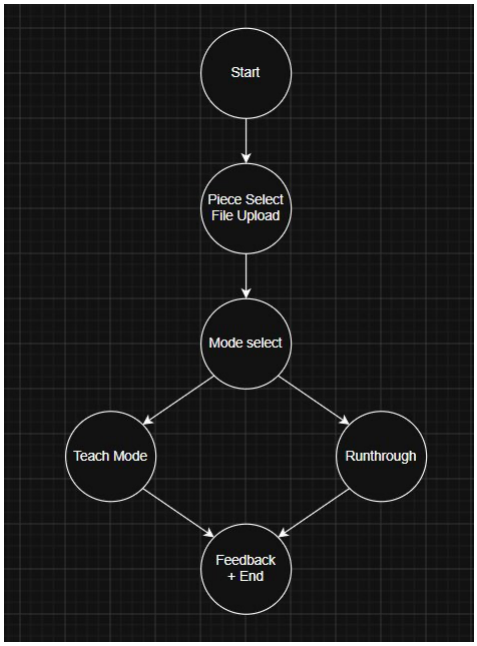

Figure 12: Finite-state machine for the operation of the program

For maintaining organization and properly moving within the looped code. Representative of the different points in operation, as shown in the FSM figure above, the following states were defined: START_MENU, SELECT_PIECE, SELECT_MODE, READY, PLAYING, RESULTS. An external button, denoted as SELECT, was used to determine when to use current variable values to transition.

2.4.2 Input Processing

The input processing, in this case, refers to receiving files via the web server and the measurement of values from the touch sensors, then calculating accuracy based off those measurements.

The gathering of location information was bounded by the physical limits of the sensors as they could only output values between 1/3 and 2/3 volts from the wiper. When read by the analog input pins, this ended up being a bit value between 750 and 1600, with 750 being at the bottom edge of the sensor. Anything outside of this range should be disregarded as a floating pin. Thus, we mapped the position using a linear function:

$$Position = (voltage - 750) \times (100/(1600 - 750))$$

Given that the active length of the potentiometer is 100mm. This returns the position detect in mm.

The accuracy code works based off a time delay. We decided on a 2 second delay for a user to play the desired note for it to be counted as correct. Thus, we increment in the following method

```
1. if (playMode == WAIT_MODE) {
2.         uint32_t noteStart = millis();
3.         waitForCorrectFinger(note);
4.         uint32_t waitTime = millis() - noteStart;
5.
```

```
 6.          startTime += waitTime;
 7.
 8.          if (waitTime <= WAIT_TIME) {
 9.            correctNotes++;
10.          }
11.
```

Figure 13: Code for how program determines correct or incorrect notes

As the piece progresses, the total notes are counted and at the end, the total accuracy is simply calculated via

$$\frac{correctNotes}{totalNotes} \times 100 = Accuracy$$

Which is displayed at the end of the piece being played.

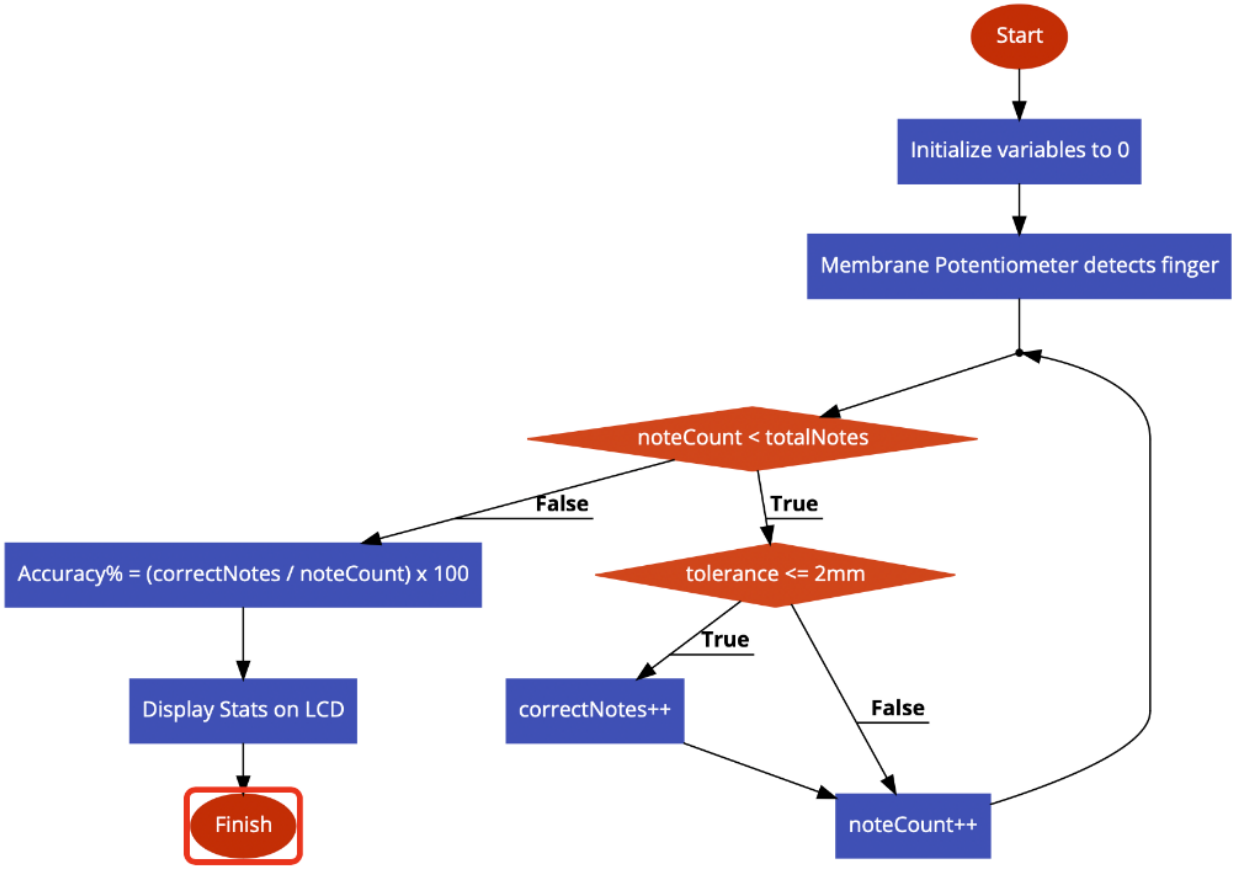

Figure 14: Flowchart for determining accuracy score

2.4.3 LED Control

The crux of the project is dependent on the mapping of the LEDs in the software. Unlike a normal LED strip, we cannot simply map blocks of notes to blocks of LEDS, because the snaking structure used to

save space and simplify the design. Thus, we linearly assign the LEDs of the G string and A string and reverse the order for the D and E string. As seen in the code snippet below:

```
void setupMapping() {
  for (int i = 0; i < 128; i++) noteToLED[i] = -1;

  // G string (LEDs 0–4)
  noteToLED[56] = 0;
  noteToLED[57] = 1;
  noteToLED[58] = 2;
  noteToLED[59] = 3;
  noteToLED[60] = 4;

  // D string (LEDs 5–9)
  noteToLED[63] = 5;
  noteToLED[64] = 6;
  noteToLED[65] = 7;
  noteToLED[66] = 8;
  noteToLED[67] = 9;

  // A string (LEDs 10–14)
  noteToLED[70] = 10;
  noteToLED[71] = 11;
  noteToLED[72] = 12;
  noteToLED[73] = 13;
  noteToLED[74] = 14;

  // E string (LEDs 15–19)
  noteToLED[77] = 15;
  noteToLED[78] = 16;
  noteToLED[79] = 17;
  noteToLED[80] = 18;
  noteToLED[81] = 19;
}

```

Figure 15: Code snippet for note-to-LED mapping

3. Verification

We tested the latency on the output subsystem, the upload speeds on the input subsystem, and the power subsystem.

3.1 Output Subsystem

For testing the output subsystem latency, we used a digital oscilloscope to measure the times that data packets were arriving. These data packets represented the note information going to the LEDs and it was vital for the spacing between any two notes to have a difference of less than 7ms keep a consistent tempo.

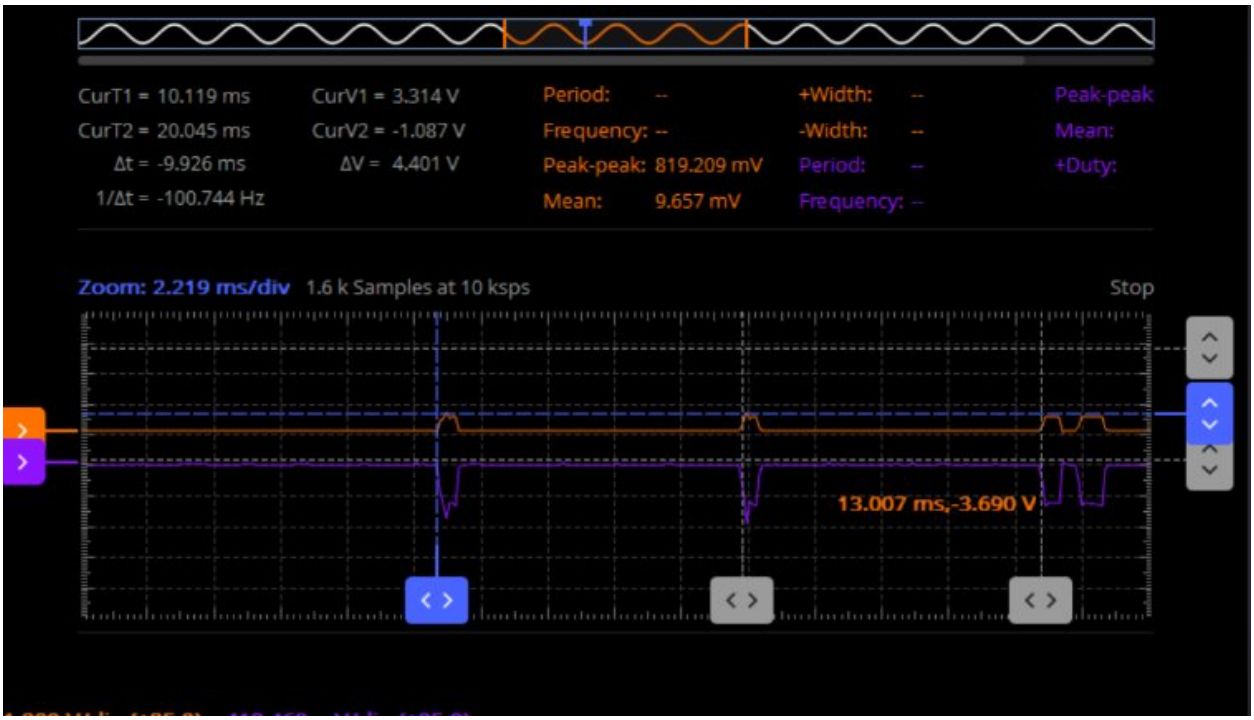

Figure 16: Digital oscilloscope showing the data packets as spikes in voltage.

We placed time markers for each rising edge of the data packet. The rising edge of the first data packet was set to 0s, the second data packet arrived 10.119ms after the first as shown by the CurT1 value above. The CurT2 value represents the time that the third packet arrives which is 20.045 seconds. That means that the third packet arrived 9.926ms after the second one. As a result, the difference in times between packets 1 and 2 and packets 2 and 3 is 0.193ms which is much less than the requirement of 7ms.

3.2 Input Subsystem

To fulfill this requirement, we used a network analysis tool to upload files of various sizes to the ESP32 and see the upload times it took to complete.

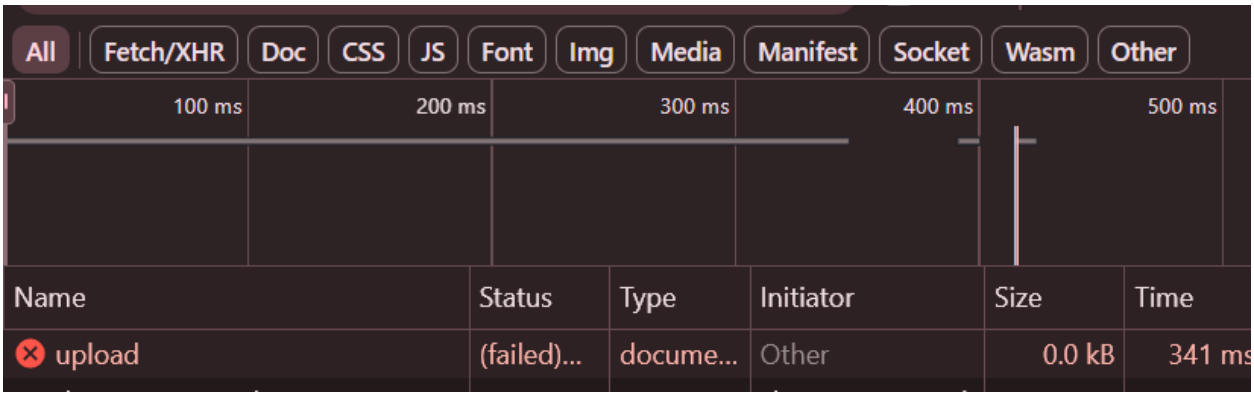

Figure 17: Upload speed of 1 KB file

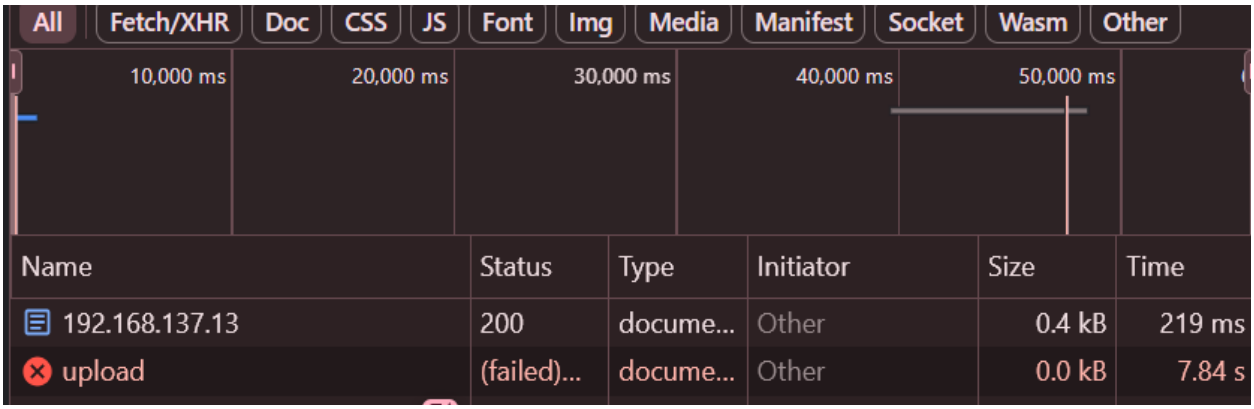

Figure 18: Upload speed of 1 MB file

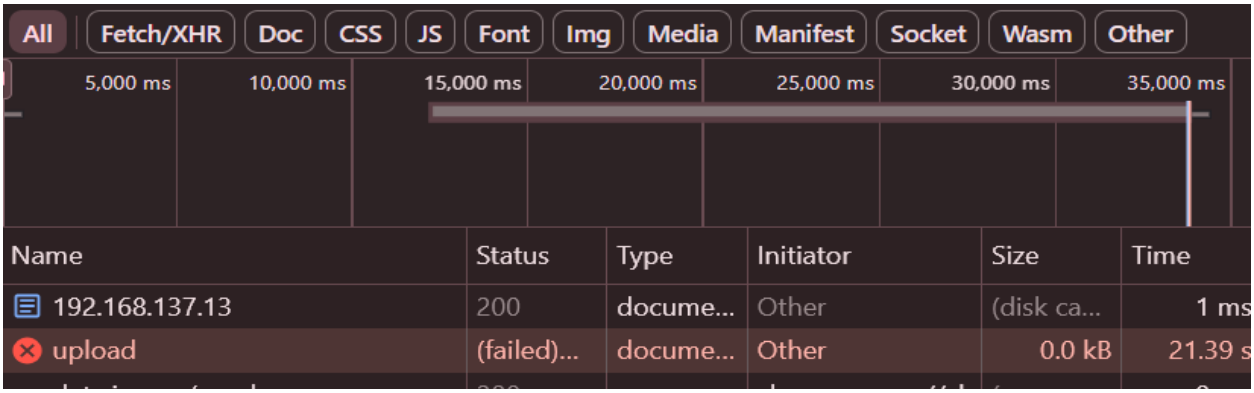

Figure 19: Upload speed of 1.2 MB file

The three file sizes we used were 1 KB, 1 MB, and 1.2 MB for the upload tests. The most important one for this requirement was the 1 MB file which took 7.84 seconds to upload. This is 2.16 seconds lower than the maximum time. Therefore, this requirement was fulfilled. We also used larger file sizes to test at what file size would exceed that limit so we wouldn't upload any files near that size.

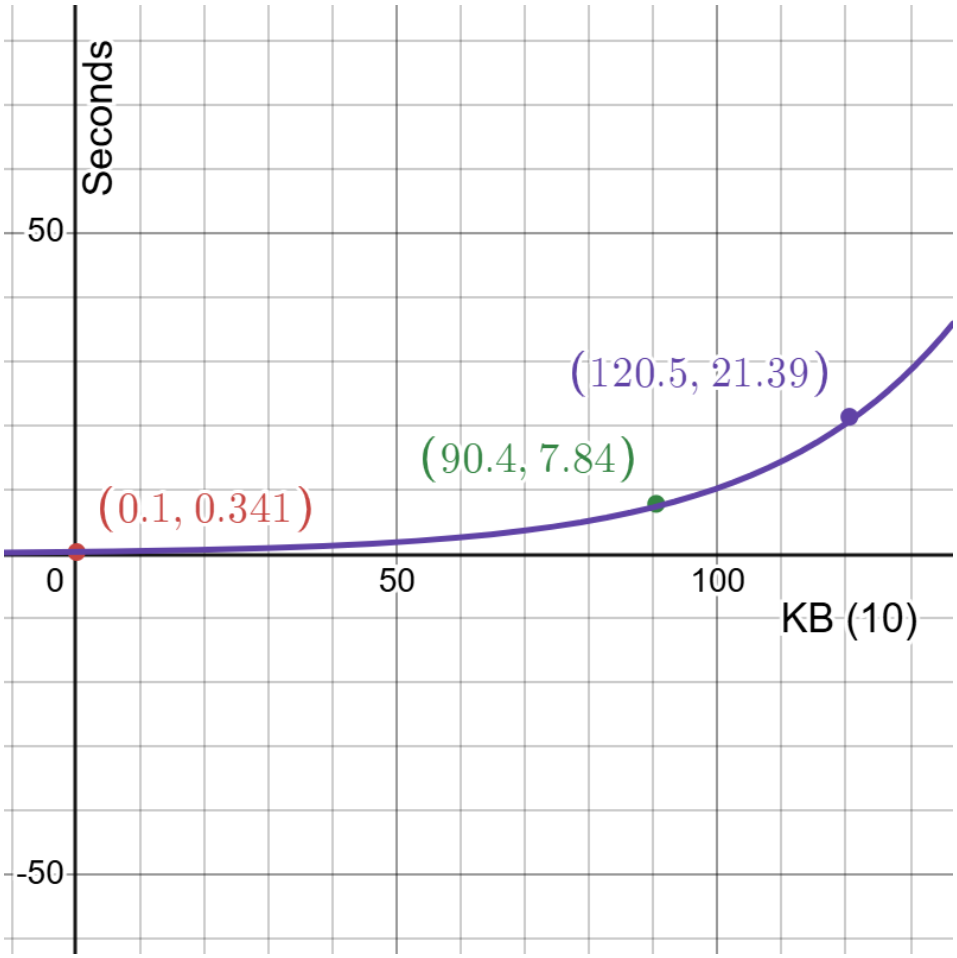

Figure 20: Graph created from three test values relating file size in 10's of KB to upload times in seconds

When the three test points were plotted, it created an exponential function. It started increasing significantly above 10 seconds after the file size exceeded 1 MB.

3.3 Power Subsystem

When we tested the power subsystem, we faced and documented quite a few issues.

- **Inner LEDs are not getting enough power**
- **(Individual test)**
  - **Power bank output 6.4 volts (>5)**
  - **3.3 voltage regulator output 4.2 (>3.3)**
- **(Multi-system test)**
  - **Power button pin, ground at ground of 3.3 voltage regulator**
  - **Vin got 1.36, and Vout got 0**

Figure 21: Documentation on power subsystem issues

For the flexible PCB, we used a multimeter to check the voltage running through each LED. We noticed that only one of the outer rows of LEDs was getting 5V and the others were not. We suspect that bridging was the cause, which means that excess solder created an accidental connection between two different components short circuiting the system.

For the main PCB, we wanted to test the voltage regulators individually to see if they were getting any input voltage. We noticed that they either got no input voltage or a 1.36V which was an insufficient amount. The root cause of this was that there was a button switch pad that was between the input voltage pad and voltage regulator this was most likely preventing current to flow to the voltage regulator regardless of whether there was a button or not. We also tested the performance of the 3.3V regulator directly. What we noticed is that the output voltage was 4.2V which was much higher than the tolerance for the ESP32 microcontroller. The cause was that the power source we were using was 6.4V instead 5V. The voltage regulator should output 3.3V regardless of the input voltage. Given the ratios between the output to the input voltages being the same however, the regulator seemed to be acting more like a resistor rather than an actual regulator, so it is likely that this component was faulty.

4. Cost and Pricing

4.1 Cost Analysis

4.1.1 Labor

We estimate our labor assuming three major roles are present: a PCB Design Engineer, Embedded Software Engineer, and a general Electrical Engineer (so that we account for safety testing, verifications, and any other miscellaneous work or assistance provided in the other areas).

A typical ECE graduate from UIUC makes between $90,000 (for EE) and $100,000 (for CompE) annually. Furthermore, a PCB designer makes on average $75,000 per year. An embedded software engineer on average makes approximately $111,000 per year. This is approximately equivalent to:

$$\frac{\$75000/year}{52weeks/year * 40hrs/week} \approx \$36/hr$$

$$\frac{\$111000/year}{52weeks/year * 40hrs/week} \approx \$53/hr$$

$$\frac{\$90000/year}{52weeks/year * 40hrs/week} \approx \$43/hr$$

Estimating the cost of labor thus is as follows (noting that once the product is designed and at sold at scale, this will not be the sale price):

$$\frac{\$36}{hr} * 80hr = \$1440$$

$$\frac{\$53}{hr} * 60hr = \$3180$$

$$\frac{\$43}{hr} * 60hr = \$2580$$

$$(\$1440 + \$3180 + \$2580) \times 2.5 = \$18000$$

Additionally, we will use an approximate 10 labor hours from the machine shop. Assuming a rate of $20 per hour, this adds another $200 of cost to the project. This yields:

$$Total\ Labor\ Cost = \$18200$$

4.1.2 Parts

Table 1: Parts List

Description	Manufacturer	Part #	Quantity	Cost/pc	Source
Membrane Potentiometer	Spectra Symbol	SP-L-0100-103-3-ST	4	$7.95	Adafruit
USB-C Gen 2 Receptacle	Amphenol ICC	10152803-00011LF	1	$1.08	Digikey
Voltage Regulator 3.3v	UMW	LD1117-3.3	1	$0.30	Digikey
Voltage Regulator 5v	Texas Instruments	LM1117DT-5.0/NOPB	1	$1.71	Digikey
16x2 LCD Display Module	Hosyond	B0BWTFN9WF	1 (comes in packs of 3)	$4.33	Amazon
Individually Addressable LEDs	Feztek	FZ2812-5050	20	$0.63	Digi key
ESP32 Microcontroller	Espressif Systems	ESP32-WROOM-32E-N4	1	$4.84	Digikey
USB-UART Bridge	Silicon Labs	CP2102N-A02-GQFN28	1	$4.32	Digikey
Knob/Rotary Potentiometer	Panasonic Electrical Components	EWV-YKXL16B14	1	$3.02	Digikey
On/Off Rocker Switch	E-Switch	RA1113112R	1	$0.64	Digikey
Buttons	Adafruit	1445 (Prod ID)	3	$0.95	Adafruit

4.1.3 Totals

Thus we can calculate the total cost of parts:

$$Total\ Parts\ Cost = (4 \times \$7.95) + (3 \times \$0.95) + (20 \times \$0.63)$$

$$+(\$4.84 + \$4.32 + \$3.02 + \$1.08 + \$0.30 + \$1.71 + \$0.64) = \$63.16$$

We end up with a grand total of:

$$Total\ Labor\ Cost + Total\ Parts\ Cost = \$18200 + \$63.16 = \$18263.16$$

5. Conclusion

5.1 Accomplishments

In the end we were able to successfully implement the entire system on the breadboard. We implemented not only the web server, but also a selection menu for users to pick between different modes and pieces that had been uploaded to the device. We were able to accurately determine the timing and placement of fingers in relation to the potentiometer zero aka the nut of the violin. Additionally, it did fit within the confines of a physical violin, particularly under the strings, which had an especially low clearance.

5.2 Uncertainties

Other than the power subsystem issues mentioned earlier, there were a few other issues we faced. The flexible PCB was very delicate to the point that any weak solder joints would cause the component and the pad to come off. This is what happened with the power/ground pads which made us workaround the issue by soldering wires directly to one of the LEDs.

We also faced issues with a lack of components. This included the lack of surface mount Zener diodes as well as capacitors. We had 10 capacitors soldered on and we were missing 10 more. This wasn't a major issue as they were used for debouncing and didn't directly affect the functionality of the circuit. We also couldn't find a 3.3V regulator for two weeks as the ECE supply and self-service shops were out-of-stock. This was a major setback because we needed this component to start testing the ESP32 microcontroller.

We also couldn't mount the USB port because of the difficulty soldering it onto the pad. A workaround was found where we piggybacked off the already programmed ESP32 on the development board.

5.3 Future Work

In the future, we would like to add modifications to the software that would improve user-friendliness. The current code only tells the user whether the note they played was correct or not and does not give the user enough feedback on what they are doing wrong. We would like to display the note that the user is supposed to play and what note they did play if the user was incorrect. We did also consider using open-source optical recognition software that could read sheet music and convert it into a MIDI or binary file. While not 100% accurate, it would be easier for the user instead of having to manually create a sequence of notes on a MIDI file manually. In terms of hardware, we would like to have a more polished enclosure that is detachable and a slimmer flexible PCB with thinner LEDs. Finally, we would like to add detachable sensor arrays and expand software support for more instruments.

5.4 Ethical Considerations

Due to the nature of electrical systems, it can be inherently dangerous to keep a battery-operated device near people and on expensive equipment, as a fault in equipment could cause combustion leading to unjust harm to an individual and/or personal property, as outlined by the ACM (Association for Computing Machinery, 2026) 1.2 and IEEE (Institute of Electrical and Electronics Engineers, 2026) I.1.

Thus, should there be a failure in the design or degradation over time, it is possible that the player or their instrument be harmed or severely injured.

As the designers, we are obligated to add redundancy both to prevent such accidents, as well as ways to mitigate accidents should they occur. This would include appropriately strong housing in case of any blown components and methods for heat dissipation to prevent overheating (which can cause warping in wood). The industry standard for electronics is generally plastic due to its electrical isolation and heat-resistant properties (AIP Precision Machining, 2025).

Furthermore, the ability for users to upload MIDI files without any sort of licensing or verification can enable, or otherwise encourage, the improper acquisition of creative works – specifically music. According to the US Copyright Office, derivative works made without permission from the original owner constitute copyright infringement (US Copyright Office, 2026). This can include MIDI files as they are generally treated as another medium expressing the same creative work. Sort of like uploading a digital copy of a physical book or translating it into a different language.

There seemingly isn't very much we can do to stop this without entirely removing the customizable uploads. The open-ended, almost open-source, aspect of this project rests on the proper conduct of the consumer when it comes to copyrighted works.

The self-sufficient nature of the product can potentially lead to decreased interest in receiving formal musical training. However, this product is not meant to be a replacement and treating it as would lead to overall worse musical performance across society.

References

[1] AIP Precision Machining. (2025, August 5). *Plastic for Electronic Enclosures*. Retrieved from AIP Precision Machining: https://aipprecision.com/plastic-for-electronic-enclosure-expert-guide-to-maximum-protection-standards/

[2] Association for Computing Machinery. (2026). *Code of Ethics*. Retrieved from Association for Computing Machinery: https://www.acm.org/code-of-ethics

[3] Institute of Electrical and Electronics Engineers. (2026). *IEEE Code of Ethics*. Retrieved from IEEE: https://www.ieee.org/about/corporate/governance/p7-8

[4] US Copyright Office. (2026). *Definitions*. Retrieved from U.S. Copyright Office: https://www.copyright.gov/help/faq/faq-definitions.html#:~:text=What%20is%20copyright%20infringement%3F,%2Dpeer%20(P2P)%20networking%3F

[5] Espressif Systems. (2019). *ESP32 Series Datasheet Version 5.2*. Retrieved from Espressif Systems: https://documentation.espressif.com/esp32_datasheet_en.pdf

[6] Feztek, *FZ2812-5050 Datasheet.* Retrieved from Feztek: https://www.feztek.com/uploads/1/2/6/3/126369845/fz2812-5050_datasheet.pdf

[7] Amro Music, *Violin Diagram*, Retrieved from Amro Music: https://www.amromusic.com/violin

Appendix A. Requirements & Verification Tables

Table 2: Requirements and Verification for Power Subsystem

Requirements	Verification
Must supply a consistent voltage to the ESP32 microcontroller that is within 3.3V +- 5% and a current of 250mA	• Connect voltage supply to another subsystem • Connect positive lead of multimeter to the microcontroller and the negative lead of the multimeter to ground • Check the multimeter to make sure that the voltage that goes through the microcontroller deviates at most 5% from 3.3V • Increase the voltage supply to the other subsystem and make sure that the voltage that goes through the microcontroller still says within 5% of 3.3V
Must supply a consistent voltage to the LEDs and LCD display that is within 5V +- 5% and a current of 500mA	• Connect voltage supply to another subsystem • Connect positive lead of multimeter to the LCD display and LEDs and the negative lead of the multimeter to ground • Check the multimeter to make sure that the voltage that goes through the LCD display and LEDs deviates at most 5% from 5V • Increase the voltage supply to the other subsystem and make sure that the voltage that goes through the LCD display and LEDs still says within 5% of 5V

Table 3: Requirements and Verification for Input Subsystem

Requirements	Verification
The input subsystem should be able to detect finger placement of the user under 2mm +- 0.1mm	• Mark correct positions with tape; measure deviation from tape when correct note detected
ESP32 should take input via a web server and download desired binary file within 10 seconds +- 0.5 seconds	• Use a network analysis tool to determine the delay between sending a file to the web server and receiving a response

Table 4: Requirements and Verification for Output Subsystem

Requirements	Verification
Latency between microcontroller and LEDs must be self-consistent within a variation tolerance of 7ms +- 1ms	• This will be very difficult to measure with a stopwatch so we cannot use completely quantitative measurements • Humans recognize latency between notes when it is above 7ms so as long as there is a consistent tempo between successive notes and we cannot notice any tempo changes then it meets the requirement • Measurement of signal timings via Scopy