DigiBraille

Zeynep Ozkaya, Joshna Iyengar, Becky Button

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—** We are designing a solenoid/stepper motor driven system controlled by a Raspberry Pi capable of embossing braille that can be directly accessed from a smartphone. Our device will connect to a web-app that allows users to input text or product instructions to print. It is >3x cheaper than the average commercial embosser and can be accessed with a phone unlike commercial embossers.

Index Terms— Braille, Cache, CSV, Database, Python, Raspberry Pi, Solenoid, Web Scraping

I. Introduction

Nearly 1 million Americans suffer from serious vision loss and blindness [9]. Due to limited amenities for disabled people, those who suffer from vision loss can lose their ability to lead independent lives much faster than their peers. Simple tasks for sighted people, such as reading a recipe or jotting down a quick note can become a complex undertaking for a visually impaired person, even with the aid of existing technologies.

Current technologies that provide blind people this access include braille printers, electronic braille readers, and braille writing slates. However, these technologies are also limited. Braille printers are very expensive, with costs ranging from $1500-$5000 [1]. In addition, assistive technologies are much more accessible through phones than computers [2]. However, braille printers can only be accessed through a computer. Electronic braille readers provide portable access to users and can be easily accessed through the phone. However, these devices can only display a limited number of characters and can cost up to $15,000 [3], which limits the user to where they can take the written information (for example, a user may not want to risk spilling something on their expensive device while reading a recipe or carry it with them in the store while reading a grocery list). Braille writing slates overcome the limitations of cost, but also display a limited amount of braille and can take a very long time to fill out.

To address these shortcomings, we propose a device that will take user input from a web-app that can be directly accessed from the phone and send it wirelessly to a custom braille printer to deliver an embossed braille sheet to the user. A key component of our project is accessibility from the phone, which is a very fast and usable interface for visually impaired people. From reading recipes for packaged food to quickly jotting down a grocery list, it is important to give blind people quick access to information at their fingertips in their preferred format for information.

II. Use-Case Requirements

Our device is intended for blind users who can read braille and require easy access to printed braille for everyday use. Thus, our requirements include fast delivery of braille, accuracy in embossing and translation of braille, completeness of information presented to the user, and an accessible interface. The device must produce results at a comparable speed to state-of-the-art devices and must have similar levels of accuracy. The information presented to the user must be complete, and the interface must be accessible to the user, which includes learning and everyday use of the device.

Our speed requirement is driven by the need to provide our users with fast, convenient access to printed braille. We are not attempting to re-invent braille printers, but rather provide users with a more convenient way of accessing braille than existing technologies (i.e., braille writing slates or electronic braille readers). As a result, we require our device to work faster than the time it takes to use a braille writing slate. From user interviews, we have determined that it takes a user about 15 minutes to write 3 lines of braille on a braille writing slate. We require the braille printing process of our device to take at most 10 minutes. We also require our device to be accurate, which includes correctly translating recipes to a braille format and taking the encoded braille format and embossing it with standard braille specs to ensure readability. We require 95% accuracy in translation because there are some differences in braille between users that don't impact readability (ex: contracted vs uncontracted). Our accuracy requirements also include 100% readability of braille. We require 95% completeness of web-scraping, as we do not expect our algorithm to produce results 100% of the time due to the vast number of potential recipes and information available on the web.

Our final requirement is accessibility, which encompasses ease of use with the UI and braille embosser. We require our web-app to be able to interface with existing accessibility features on phones and that it takes no more than 1 second to display results. The braille printer itself will have tactile cues to guide the user. We will implement raised edges to guide the user's paper placement. We will also include a speaker that will inform the user about updates in printing. This includes status updates every 2 minutes and updates about when printing is finished. The user should also be able to learn how to use the device in less than 10 minutes.

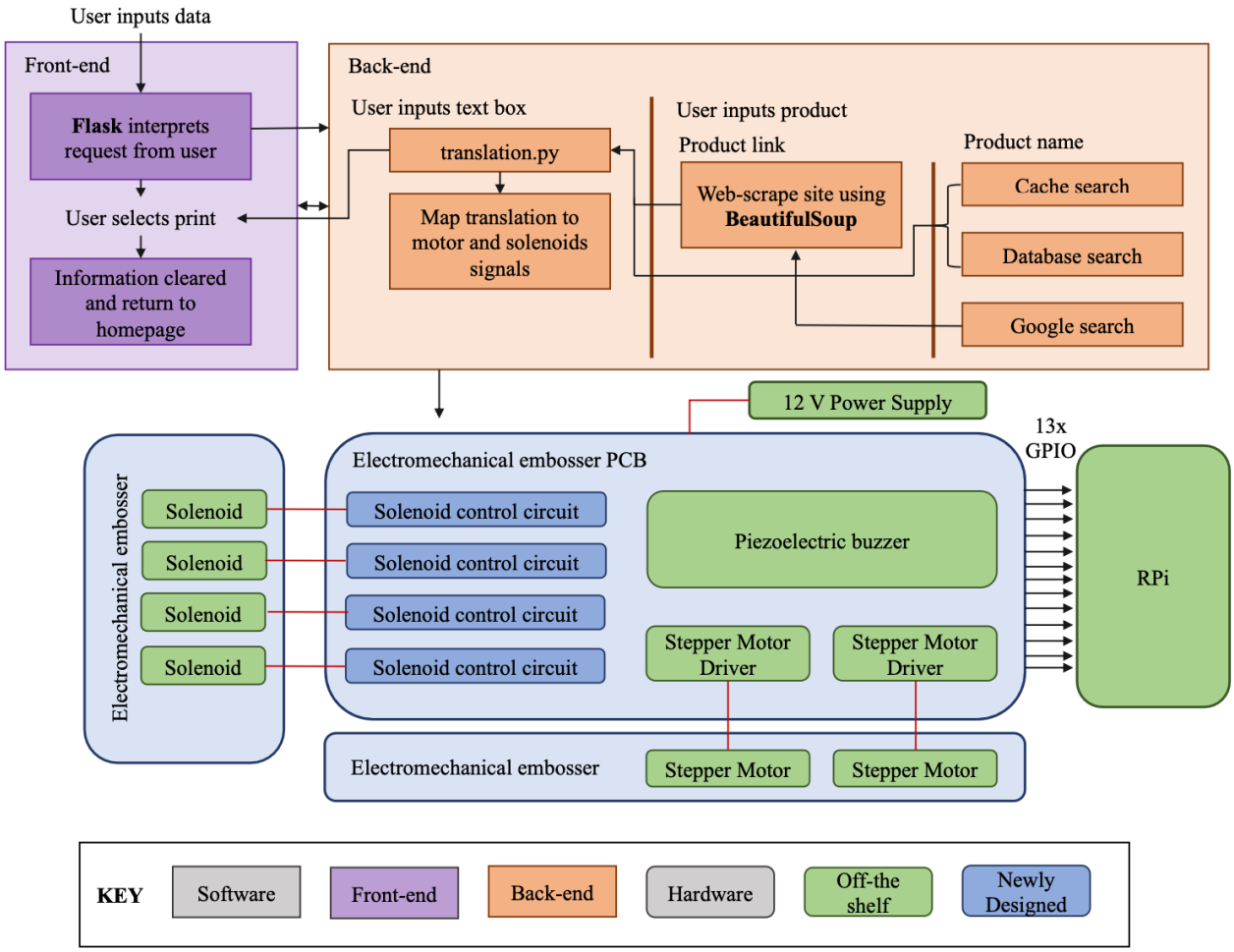


Figure 1. Full system block-diagram

III. Architecture and/or Principle of Operation

Figure 1 outlines our full system block diagram. Our device has two primary components: an embosser and a web-app. The braille embosser will be 30x15 cm in area and enclosed in a laser-cut acrylic encasing. The braille embosser will be implemented similar to traditional printers using a roller driven y-axis to move the paper through, and a lead screw attached to a gantry holding the embossing tool, both axes will be actuated by stepper motors. The embossing will be accomplished using four push-pull solenoids. The solenoid arm will move along the z-axis, and when activated will push on the paper to emboss a single dot of a braille cell. The solenoids will be attached to a carriage that moves along the x-axis. Tactile cues will guide the user as they insert the paper into the braille embosser and auditory cues will inform the user about status updates as their paper embosses. The stepper motors and solenoids will both be controlled with signals sent through an RPi.

On the software side, our web-app will be hosted on the RPi and allows for wireless access from a smartphone to the braille embosser. The web-app allows the user to input a chunk of text or product name for instructions that they want printed on a physical sheet of braille. The user can input information by either using an electronic braille reader that is compatible with their smart phone or through our dictation feature. We included both options to ensure usability for braille users that may not have access to electronic braille readers. If the user input is a product name, this product is searched for in our product database using hash functions. This database is created through offline web-scraping on multiple sites, and then is formatted into a readable format before converting to braille in our back-end text-to-braille conversion algorithm, which translates the American English text to contracted North American Braille Alphabet. If the user enters a chunk of text to print, this text is directly sent to the text-to-braille algorithm. Once the text is converted to braille, we will map this braille to the solenoids and motors to send signals through our RPi. An array of four binary signals will represent the voltages sent to the solenoids, which will be controlled by a single N-MOS transistor and a flyback diode. The RPi will also send motor control signals to our stepper motors, and the motor driver circuitry will be integrated into a custom PCB. The user will hit a print button on the web-app to inform the device that they are ready to print their selected text box. We incorporated auditory signals through a speaker, such as status updates every 2 minutes as printing commences through a buzzer, updates for when printing has begun, and when printing has ended to provide feedback to the user throughout the printing process. The user will receive a portable sheet of braille as the result of our product.

III. Design Requirements

Our design requirements are driven by our use-case requirements: speed, accuracy, accessibility, completeness.

Speed. We require our embosser to take less than 10 minutes to emboss a full sheet of braille. This limit was determined by the time it takes a blind user to transcribe the same amount of text on a braille writing slate. To achieve this, we require a stepper motor that rotates from 15-300 MMS and push-pull solenoids that acuate at <.5 seconds. The electromechanical system must emboss a single braille cell in at most .5 second. On the software side, we require our web-app respond to button presses <250 ms and to return properly formatted product instructions in response to user input, which is comparable to the latency of a Google Search [4]. We require <3s to convert from American English to North American Braille based on the time it takes for existing online translators, and 2ms/braille cell to map output to the solenoids and stepper motor signals. Once the user selects the "print" option on our web-app, we require the signal be sent to the hardware through our RPi in 30 ms. The web-app should return to next site/text within 1 second of button press given time of typical web scraping and time before people question if the site is still loading.

Accuracy. Our accuracy requirement is driven by accuracy of the solenoid embossing and accuracy of the text-to-braille conversion algorithm. To ensure successful embossing, we require at least 6 Newtons of force from solenoids, which was determined through literature survey [5][6]. Our solenoids provide 20 N, and the level of this force can be modulated by varying the amount of voltage we apply. To maintain accuracy, the solenoids must also be adequately powered. We will be using 4 12 V solenoids that draw at most 2 amps of current. Thus, we require a power supply that can supply 12 V and at least 8 amps of current. We also require our embossing to be within standard braille specs: the diameters of adjacent dots must be within 2.3-2.5 mm of each other, the diameter of a single pin must be between 1.5-1.6 mm, adjacent braille cells in the same line must be within 6.0-7.0 mm of each other, and adjacent braille cells in the two different lines must be within 10.0-11.0 mm of each other as demonstrated by Figure 2 [7]. These tolerances will be controlled for through our programming of stepper motors. We also require accuracy in our software. The text-to-braille conversion must be 95% accurate, which is acceptable the same way slight grammatical differences are acceptable in American English.

Accessibility. To achieve our accessibility requirements, the user must be able to acquaint themselves with the device and web-app in less than 10 minutes, which includes easy to use interface and consistent feedback to the user. To ensure accessibility with the UI, we require our hit target buttons to be at least 44x44 pixels with centers at least 60 pixels apart [8]. The website font should be 16 pixels according to standard large font practice, and all text should be in the same style. We will be interfacing with existing accessibility features for the web-app portion of our project but will implement our own auditory cues on the braille embosser. We require the buzzer to alert the user that printing has begun at most 3 seconds after the user hits print, which is the average time it takes standard printers to begin printing. We require the speaker to provide consistent updates as well. The braille embosser system must be able to be picked up (30x15 cm) and weigh <5 kg. The embosser must also include tactile cues to help the user input paper through raised edges around the corner of the printer bed. These edges will be raised by 1 cm and the paper will fit into these corners.

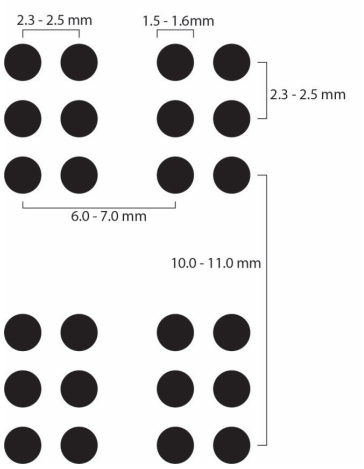


Figure 2. Braille size specifications

Completeness. This design requirement is met by our web-scraping design. We are implementing a database with results from web-scraping done offline. This database should web-scrape from at least 3 websites that compile at least 500k products total.

IV. Design Trade Studies

A. Solenoid System

The choices made for our solenoid system were informed by our requirements of speed, accuracy, and limiting cost. The solenoids are essentially used to provide a linear motion in the z-direction. Other methods could have also been employed to provide this motion such as stepper motors. However, stepper motors are slower than solenoids, and while they provide more control over motion, we need only a single stroke in the z-direction, so stepper motors would overcomplicate our system. Solenoids have a simple control circuit and move very quickly, which satisfies our need for speed. After deciding to use solenoids, the next trade-off was deciding how many solenoids to include. Table 1 demonstrates the calculations we did to determine the amount of time embossing would take for a given number of solenoids. The more solenoids we included, the faster the device would be. However, this is a trade-off between cost, power management, and size. We calculated the timing for 1-4 solenoids, with 4 fitting best into our user and design needs. Our results are shown in Figure 3. We also had to decide the optimal way to emboss. Instead of embossing one cell at a time, we decided to emboss by line of braille dot, which produces the least amount of stepper motor motions in the x and y direction.

TABLE I. WORST CASE PRINT TIME FOR FOUR SOLENOIDS

Motion	Timing
Solenoid actuation time	.5 seconds
Stepper motor MMS	15
Time to move to next dot in a cell	.167 seconds
Time to move to next cell	.291 seconds
Time to move to next line of dots	.359 seconds
Time to move to beginning of line of dots	11.06 seconds
Time to emboss one line of braille	16.6 seconds
Time to emboss full sheet of braille	**6.6 minutes**

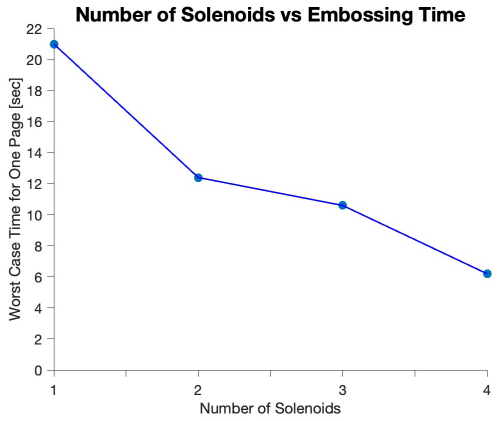


Figure 3. Number of solenoids vs. embossing

B. *User Interface*

The trade-offs we considered when designing our user-interface include the choice between a web-app, mobile application, or text-message service, which frameworks to use when creating the web-application, and how to design a user-friendly interface. We first decided to use a web-app instead of a mobile application because it provides a wider range of access for users with different types of phones. A text message service would provide easier access for users. However, it is very easy to accidentally hit the send button when using a text-message service, so we opted for a web-app hosted on a RPi because it has multiple capabilities such as hosting a website. RPi also has enough GPIO analog pins to interface with our embosser. It is extremely fast with the 4B model working at 1.5 GHz so it will be able to host our entire database and send signals quickly. We decided to use Flask for our web app because it has an easy-to-use library while still being able to do all the requests we need. To meet our requirement of accessibility, we wanted to design an interface that would run most smoothly with accessibility features that are built into phones. As a result, we ensured that there was limited text on our user interface to avoid superfluous auditory inputs and had our entire website fit into a phone screen to prevent complications from scrolling.

C. *Mechanical System*

Accuracy, product longevity, and portability requirements all informed decisions made around the x/y gantry system. We decided to decouple x and y motion, so that there are 2 different mechanisms controlling motion in each direction. A benefit of this design choice is that it simplifies the design process because we can think of these systems in isolation. The y motion system is a set of idler and driven rollers that move the paper for each new line of braille.

We decided to drive movement on the x axis with a leadscrew connected to a stepper motor. The stepper motors are ideal for very precise applications because they can rotate a very precise increment. Each electrical pulse sent to the stepper motor translates to a consistent rotation about the shaft, as opposed to a DC motor which drives continuously, and has inconsistent rotation amounts even for the same input electrical pulse. Leadscrew driven gantries are also ideal for high precision applications because they don't suffer from mechanical wear as much as other drive systems such as pulley-based systems that stretch, and they also aren't susceptible to slippage. The leadscrew is also ideal because it is a very durable part, and will be less susceptible to needing replacement, unlike pulley-based systems which stretch and become less accurate over time. Smaller pitch threads on leadscrews enable even smaller controllable movements. With smaller pitch threads comes higher manufacturing costs, so there is a tradeoff here with high precision small pitch thread lead screws costing more than leadscrews with bigger pitch threads.

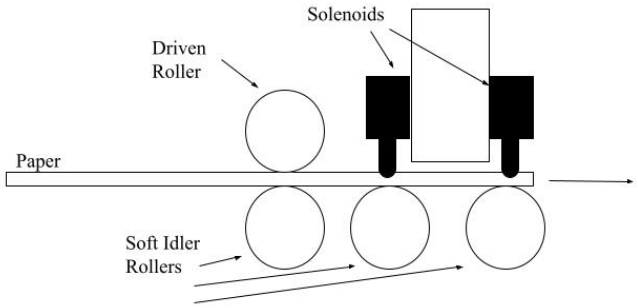


Figure 4. Mechanical embossing system side view

D. *Back-end Algorithm*

Using Python's internal timers, we found that it would take about 3.1 pages/sec to web scrape and compare the titles of each of the products from *directionsforme.org*, making the algorithm take an entire day in the worst-case scenario. While it may be possible to use machine learning to figure out what category each of these products fall under, the categories don't have strict guidelines, so it was hard for us to find recipes that way. Also, searching through that algorithm would take just as long as searching through a database that already compiles all the products in these websites which would only require one day for web scraping once. Another option would be to just use our Google Search function but that may be less accurate for certain suggestions and difficult to web scrape, so we still have it but as a last resort option. Thinking about speed, a cache is only worth it if the percentage of hits is large enough that the average time it takes for a product to be searched is less than the average time if there was no cache at all. The cache has an O(n) lookup for N_1 products while the database

has O(1) lookup with value list O(n) for N_2 products. Equation (1) refers to the comparison of speed between the two data structures where h is the percent of hits in the cache, f is the translation speed, and b is the number of buckets in the database.

$$n_1 + (1-h)\left(\frac{500000}{b}\right)f < \left(\frac{500000}{b}\right)f \quad (1)$$

V. System Implementation

Our system is made up of hardware and software components. Within hardware, our system is split into the solenoid embossing system and the mechanical plotting system. Within, software, our implementation is split into the front-end UI and back-end algorithm for web-scraping and text-to-braille conversion.

A. *Solenoid System*

Figure 5 is representative of the control circuit for a single solenoid. The flyback diode placed in parallel with the solenoid prevents the sudden voltage spike from massive changes in inductive load from harming the circuit. The N-MOS transistor will be used to control the voltage going into the solenoid. When a binary 0 is sent from the RPI the transistor will prevent the flow of current through the solenoid and will allow the flow of current when a binary 1 is sent. We will have four of these solenoid circuits connected in parallel to a 12 V power source to realize our electrical embossing system. The four solenoids will be attached to the mechanical embossing system by the carriage shown in Figure 6. The arrangement of the solenoids is illustrated in Figure 7, which is a zoomed in section of a braille page. We will use a 3 mm screw for the embosser head screwed into the tail end of the solenoid arm. The four solenoids will emboss in parallel and move in the x-direction. Once they reach the end of a line of braille dots, they will return to the beginning of the x-axis, and the y-axis rollers will move the paper up. Signals will be sent to the solenoids through the RPi.

B. *Web-application*

The web-application will be implemented in Flask, HTML, and CSS to allow the user to input product information. The back end will be written in Python. Figure 8 shows the user flow of the web-app and Figure 14 shows the mock-up of the web app design. It is crucial that the web app be designed with accessibility in mind. Thus, we are designing using the Apple Accessibility Documentation specifications, which is reflected in both our user flow and web-app design choices.

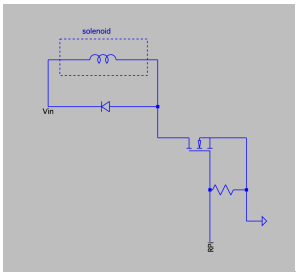


Figure 5. Solenoid control Circuit

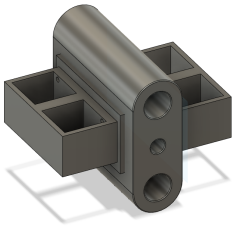


Figure 6. Solenoid carriage

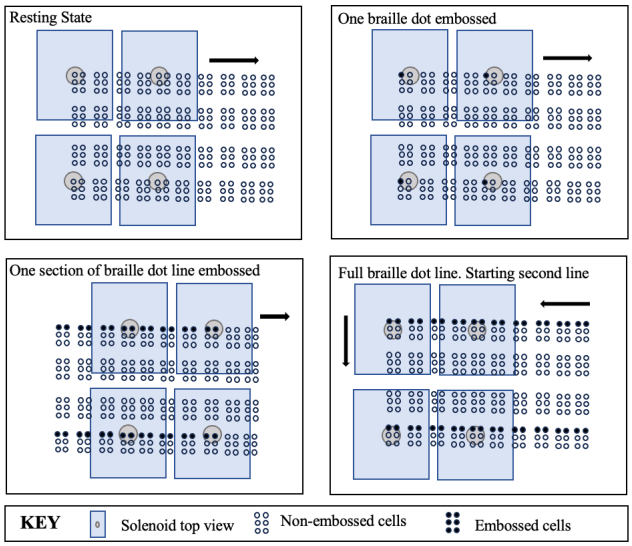


Figure 7. Embossing mechanism

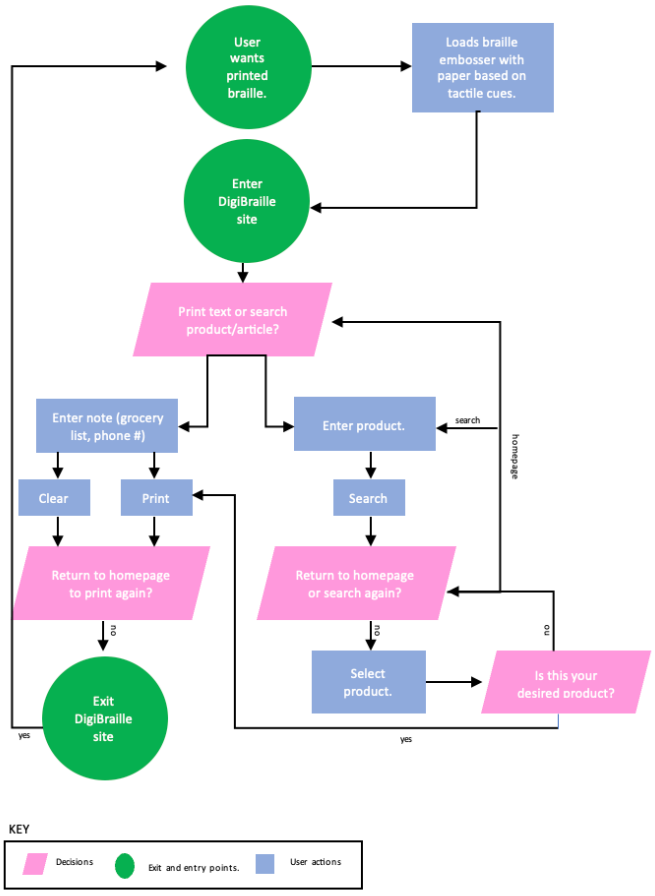


Figure 8. Web-app user flow diagram

C. *Back-end Software*

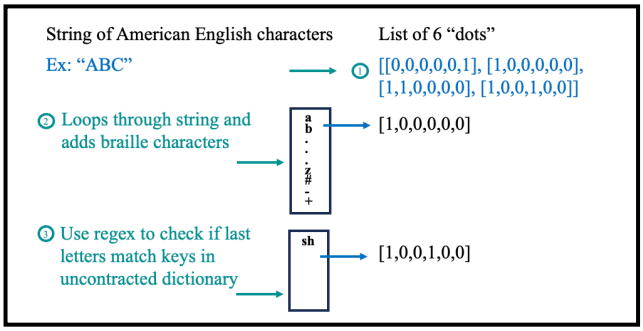


Figure 9. Contracted braille translation

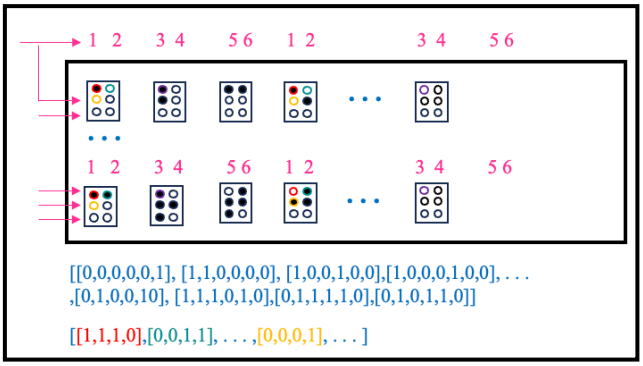


Figure 10. Braille to solenoid translation diagram

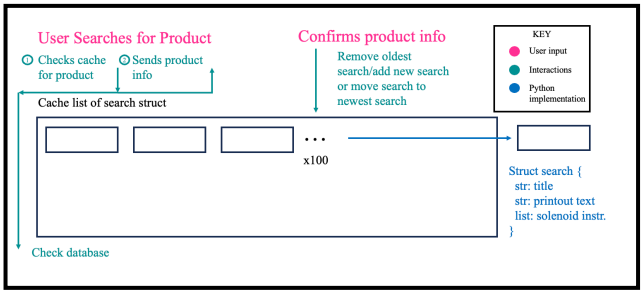


Figure 11. Cache diagram

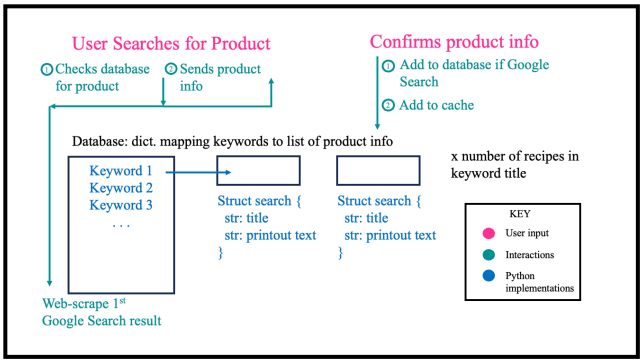


Figure 12. Database diagram

Every time a user presses print, the Python program uses the Python library Flask to interpret the user's request. If the user fills out the print field, the backend runs the translation function in translation.py which uses dictionaries and conditions to map American English characters to braille characters (Figure 9) then braille characters to solenoid instructions (Figure 10). The contracted translation has a lot of special conditions including but not limited to handling parentheses, special characters that are represented by more than 1 characters, capital letters separately vs. in a row, numbers separately vs. in a row, and contractions that are in words vs. by themselves. If the user fills out the product search field with a link, the backend runs web scraping of the site in websearch.py using the python library Beautiful Soup and sends that American English text to translation.py for translation. If the user fills out the product search field with a product, the backend looks for it in the cache, then the database, then web scrapes the first page in google results as a last case scenario. The cache implementation (Figure 11) is a python program that contains a list of the 100 most recent searches with their American English directions and solenoid instructions.

The database implementation (Figure 12) is also a python program but is a dictionary that maps keywords to products but not their solenoid instructions because of the memory-speed efficiency tradeoff, so if the product is found in the second step in the database, it also goes through translation.py. The database is created from web scraping 3 with product directions: *directionsforme*.org, *backofthebox.com,* and *harrietsblindkitchen*. The frontend file webapp.py then uses requests to post back to the user for confirmation and send a buzzer sound output after confirmation when the backend begins to send signals to the embosser. This website is hosted on a RPI with the Python file raspberry_handler.py. The binary signals for the embosser's solenoids and stepper motors are connected to the raspberry pi's GPIO pins and signals are sent to these pins with this handler file for each 4-solenoid system at a time after the translation has been completed. These files can be viewed and downloaded from our GitHub: https://github.com/joshna-ii/digibraille.

D. *Mechanical System*

The mechanical system is depicted in figure 13. We have an embosser that is connected to a gantry on the X axis. This gantry is driven by a lead screw. The Nema 17 stepper motors we are using have a step angle of 1.8 degrees, which when coupled with a 8mm pitch lead screw, can be controllable to .01mm per step of the stepper motor, which will meet the specifications described by Figure 2. The Y axis is driven by a roller, which has tension against it from idler rollers on the opposing side. There is another set of rollers acting against the push down from the solenoids further down the Y axis of the system. This is placed such that tension can be maintained between the paper and the embosser so that the braille is embossed correctly.

The Y axis roller system that moves the paper down for each line of braille conserves space in the width direction. The system can print on sheets of paper that are longer than 11 inches and takes up less space than a system where there are constraints on printable areas in the Y direction.

The enclosure of the system will have different textures to ensure that the user knows how to put the paper in. The enclosure will have a slotted opening for the paper which will make it simple for the user to load the paper. We have decided

that after the braille has been embossed that the paper will come out the same side it was loaded in, to ensure that the user knows where the printed braille is.

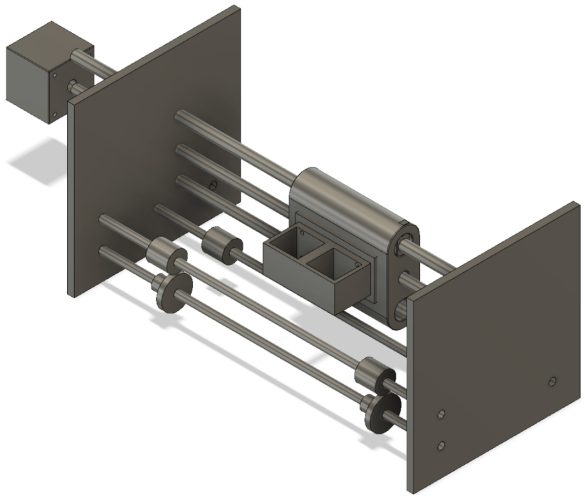


Figure 13. Mechanical design for embosser

E. Integration

We will integrate the software and hardware components by using a RPi, which will be able to host the website the user interacts with to send in text, which will be parsed and interpreted as a series of stepper motor and solenoid commands. These hardware components will be easily controlled via the GPIO of the RPi. We will be designing a PCB to plug into the GPIO, and connect the buzzer, solenoids, and Stepper Motors. The mechanical parts will be fitted into an acrylic enclosure and fastened with M3 screws.

VI. Test, Verification and Validation

A. Tests for Back-end

The success of our back-end component is defined by its ability to produce accurate and complete results for our user. We will test output of text to braille conversion on 100+ scripts of text with output from Duxbury Braille Translation software to ensure 95% accuracy as defined in our requirements. We will also test our web-scraping results on 100+ different products to ensure completeness of our web-scraped database.

B. Tests for Web-application

We will test the web-application through user-studies and Apple's Accessibility Inspector Tool. We will conduct user studies to test the web application at each point in the process which includes testing the user's ability to input into the site, the site's ability to interact with the back end with user input, and the end-to-end performance of the site. We will augment our user tests with Apple' Accessibility Inspector Tool for debugging, which will allow us to test the web-app's performance against built in accessibility features. We will test the latency of the web-application by timing how long it takes the site to display a result in response to user input. As described in sections II and III, speed and accessibility are crucial requirements for this device, so we will be specifically testing for speed and ease of navigation through the site. We will use Python internal timers to test how long it takes for each function to run for 100 get requests including but not limited to the cache extraction algorithm, database extraction algorithm, contracted braille translation, uncontracted braille translation, and post requests. We will also run unit testing for accuracy on each of these individual functions, manually testing with the 100 most popular product searches on each of the 3 websites we will be web scraping from.

C. Tests for Hardware User-Interface

We will also be conducting user tests to measure the accessibility of the hardware component of our project. We will primarily be testing the ability of our tactile and auditory cues to guide the user. We will measure the amount of time it takes a user to input a sheet of paper into our embossing device to quantify the success of our tactile cues against the amount of time it would take a sighted user to input paper in the same manner. We will also test the accuracy of our auditory cues by measuring the time stamps for each status update and ensuring that they are accurate to the state of the system. These user tests will also include testing for readability of the printed braille output.

D. Tests for Speed

We will be testing the speed of the system by timing the worst-case scenario printing time of a full sheet and ensuring that this time is less than 10 minutes, as set by our user-requirements.

E. Tests for Electromechanical System

Our electromechanical system tests are driven by our speed and accuracy requirements. We will test our physical circuit implementation by first running simulations in LT Spice to ensure that the solenoid control circuit is properly handling the input voltage from our microcontroller. We will also test the control of a single solenoid using this circuit on a breadboard before we design and fabricate our PCB. We will also test the embossing capabilities of a single solenoid using our control circuit without connecting it to the x/y gantry and work up to testing the embossing capabilities of four solenoids in parallel. We will also test the print area of the mechanical embossing system, how fast we can move the mechanical printer while still producing readable braille to specifications. We will also test the tension on the y axis, test the placement of the paper, test the movement of the rollers, and test the movement of the x-gantry by giving different commands through the RPi.

F. Integration Tests

To test the integration of our hardware and software components, we will first test our RPi output to make sure it is properly displaying the correct binary voltage values to send to solenoids, and the position sent to the stepper motors. For integration between front-end and back-end, we will consistently ensure the data being sent is accurate as we add more components to our software. For backend integration testing, we plan to use the black box method to test inputs and outputs for each individual function then each combination of functions starting from the middle outwards. For backend-hardware integration testing, we plan to test signals received

from the Raspberry Pi's GPIO pins given different 100 manual string inputs to the backend from length 0 to 100 characters. After ensuring the RPi output is correct, we will be able to connect our hardware and software components.

VII. Project Management

A. Schedule

Our schedule (Figure 15) is broken down into sections of team deadlines, hardware, front-end software, and back-end software. Briefly, our schedule is broken up into building the web-app front-end, developing braille translation and web-scraping algorithms, designing and implementing the solenoid embossing system, and designing and implementing the x/y embosser gantry. We accounted for fall break in our slack time and have left appropriate time for integration of the three main components of our project.

B. Team Member Responsibilities

Our project has been divided into two primary components: hardware and software. Each team member has a set of responsibilities that fits into these two sections.

Joshna is working on the frontend-backend communication, software algorithms for translating English text to uncontracted braille characters to solenoid instructions, database creation by web scraping multiple websites with lots of product directions, database and cache extraction algorithms, google search algorithm, communication with the raspberry pi and sending signals, hosting the website on the raspberry pi, unit testing for each of the above steps, and integration testing between frontend/backend and backend/embosser.

Zeynep is responsible for the creation of the web-application and the design of the solenoid embossing system. Zeynep has taken courses on electromagnets and has an interest in working with solenoids. She is also interested in gaining experience with front-end development.

Becky is responsible for designing and manufacturing the X/Y gantry system and designing and assembling the custom PCB that connects the buzzer, solenoids and the stepper motors to the GPIO on the RPi and power.

Each member is responsible for testing their individual components. Integration testing will be performed as a team.

C. Bill of Materials and Budget

See Table 2 for our bill of materials and budget. We have managed to keep the cost low by laser printing some of our parts in TechSpark, using the RPi available to us through the ECE Department, and testing our circuits with available components in the ECE department. We expect our custom PCBs and mechanical embossing system to dominate most of our budget.

D. Risk Mitigation Plans

We will continue to add to the database to improve speed if enough products can't be found. We will reconcile similar products from different websites together if the database returns too many search results to fit on one page. If the database extraction algorithm isn't accurate, we will compare products to google search engine's results to find more accurate results. In the absolute worst-case scenario, we will use the google search function and web scrape and print those pages. If the cache doesn't get 25% hits, then we will increase the cache size until it does but if it continues to get limited hits then we will decrease it for speed. If the contracted braille translation isn't accurate, we will either only include most common and simple contractions, revert to uncontracted braille which should not be a problem, or use existing websites for certain translations. If the raspberry pi is unable to host the website, then we will use a Carnegie Mellon University computer to do so or just use local hosting for the purpose of the project. If the time delay between sending out instructions to the solenoid and stepper motor system isn't long enough or is extremely irregular, we will have the motors send a signal back to the RPi software every time it moves for the software to keep track of when and where to send the next signal.

VIII. Related Work

The most similar work are commercial braille embossers such as Index Braille's embossers which operate at a speed of 140 cps but range from around $1800 to $5000 and this is the industry standard. While our embosser isn't as fast, it is significantly cheaper and can be controlled via a mobile device, unlike these other printers. Additionally, these embossers need to be connected to a laptop while our embossers can be accessed from both laptops and phones, which we found visually impaired people tend to use everyday compared to a laptop which they may use once in a few months based on user interviews. While many visually impaired people use dictation software to read out text to them and therefore don't want to spend the money on an expensive braille embosser, we also found in these user interviews that there are many people who want to be able to read things in braille on demand and this is even a service that some accessibility libraries such as LAMP provides. Our embosser would allow people to receive this service whenever they need it and for a much cheaper cost. Another competitor are Braille eReaders, which use a similar solenoid system to act as the actual braille dots and can be recalibrated by downloading a new book to be read. However, this calibration typically requires a sighted person to help with while our product can be used by anyone. Also, it has a similar price range to industry standard braille embossers which is much more expensive than our embosser. Additionally, there are places where a person may not want to bring an electronic device such as a kitchen or bathroom in case of spillages and since our embosser prints on actual paper, this would not be a problem.

IX. Summary

Our low-cost phone-controlled braille embosser will enable more information to be easily accessible through braille. Our device streamlines the process for getting directions for products, while also leaving the possibility for inputting other information. The device has been designed with high speed, accuracy, and user requirements in mind such that the device is usable by our intended audience. Our device is able to overcome challenges posed by traditional embossing systems

that need a computer to run through the use of a RPi which enables web connected GPIO. While there are many components necessary to creating this system, we have created an implementation plan to give us plenty of time to integrate the components of our design and test them. Additionally, we have many alternative solutions in case certain parts don't integrate the way we expect.

Glossary of Acronyms

CSV – Comma separated values
GPIO – General Purpose Input Output
MMS – Millimeters per second
PCB – Printed Circuit Board
RPi – Raspberry Pi
ECE – Electrical and Computer Engineering

References

[1] Lise. "The Smartphone: A Revolution for Blind and Visually Impaired People!" *Inclusive City Maker*, 5 July 2022, www.inclusivecitymaker.com/the-smartphone-a-revolution-for-the-blind-and-visually-impaired/.

[2] "Braille Printers." *The American Foundation for the Blind*, www.afb.org/blindness-and-low-vision/using-technology/assistive-technology-products/braille-printers. Accessed 12 Oct. 2023.

[3] "Refreshable Braille Displays." *The American Foundation for the Blind*, www.afb.org/node/16207/refreshable-braille-displays. Accessed 12 Oct. 2023.

[4] *Research at Google*, static.googleusercontent.com/media/research.google.com/en/pubs/. Accessed 12 Oct. 2023.

[5] "Low Cost Braille Embosser: Manualzz." *Manualzz.Com*, manualzz.com/doc/24178544/low-cost-braille-embosser. Accessed 12 Oct. 2023.

[6] *Low Cost, Compact Braille Printing Head for Use in a Handheld Braille ...*, repository.library.northeastern.edu/files/neu:1672/fulltext.pdf. Accessed 13 Oct. 2023.

[7] "Size and Spacing of Braille Characters." *Size and Spacing of Braille Characters | Braille Authority of North America*, brailleauthority.org/size-and-spacing-braille-characters. Accessed 12 Oct. 2023.

[8] "Human Interface Guidelines." *Apple Developer Documentation*, developer.apple.com/design/human-interface-guidelines/. Accessed 12 Oct. 2023.

[9] "Prevalence Estimates Vision Loss and Blindness." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 31 Oct. 2022, www.cdc.gov/visionhealth/vehss/estimates/vision-loss-prevalence.html#:~:text=Approximately%206%20million%20Americans%20have,as%20nursing%20homes%20or%20prisons.

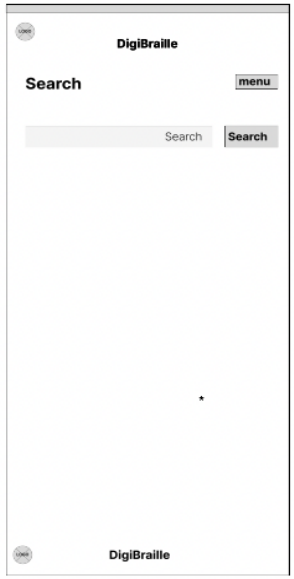
TABLE II. BILL OF MATERIALS

Description	Model #	Manufacturer	Quantity	Cost @	Total
DC 12V 2A Solenoid	1199703	Harfington	4	$7.18	$28.72
M3-8 Screw	36-9906-ND	Keystone Electronics	25	$0.148	$3.70
M3x12mm Grub Screw	JDJM007	Abbot Store	50	$0.11	$5.50
Nema 17 Stepper Motor	17HE15-1504S	Stepper Online	1	$8.99	$8.99
Linear Slide and Mount	L526C34617Z	CNC Yeah	1	$17.99	$17.99
Linear Bearing	LM8UU	Edoneery	1	$11.99	$11.99
X-axis nema 17 w leadscrew and coupler	MY-17LS16-1504E-310J	MybotOnline	1	$28.99	$28.99
8mm Linear Rails	FG008	Feyrix	4	$5.70	$22.80
Idler Rollers	U-0846.5-10	Preamer	6	$2.08	$12.48
Drive Rollers	2471K16	McMaster Carr	2	$32.19	$64.38
Stepper Motor Drivers	A4988	HiLetGo	2	$2.04	$4.08
Raspberry Pi 4 GB	4B	Element14	1	$0.0	$0.0
Custom PCB	-	FlashPCB	1	$0.0	$0.0
M6 x 20mm	-	Ideate	VAR	$0.0	$0.0
M3x 20mm	-	Ideate	VAR	$0.0	$0.0
12 V 12 A Power Supply	BOBHWMP8H5	ALITOV	1	$26.99	$26.99
Custom 3D Printed Parts	-	TechSpark	1	$0.0	$0.0
Lasercut Acrylic 3mm	-	TechsSpark	VAR	VAR	VAR
					$236.61

Wireframe - Homepage

Wireframe - Notes feature

Wireframe - Search feature



Wireframe - Displaying Options

Wireframe - Displaying options (next page)

Wireframe - Select Recipe/Product

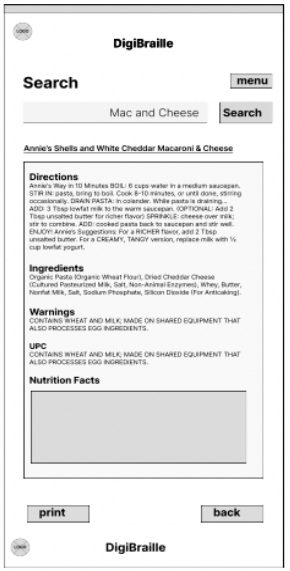


Figure 14. Web-app wireframe

DigiBraille

18-500 ECE Design

Becky Button, Joshna Iyengar, Zeynep Ozkaya

Project Start: 9/11/2023

Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	DAYS	END
Course Deadlines					
Proposal Presentation	All	WORKING	9/10/23	7	9/17/23
Design Presentation	All	NOT STARTED	9/17/23	14	10/1/23
Design Report	All	NOT STARTED	9/24/23	19	10/13/23
Final Presentation	All	NOT STARTED	11/19/23	14	12/3/23
Slack	All	WORKING	9/10/23	84	12/3/23
Front-end Software					
Initial design	Zeynep	DONE	9/17/23	14	10/1/23
Website mock-up and usability research	Zeynep	DONE	9/17/23	1	9/18/23
Finalize web app workflow	Zeynep	DONE	9/19/23	3	9/22/23
Build basic framework	Joshna	DONE	9/22/23	9	10/1/23
Set up server	Joshna	DONE	9/22/23	3	9/25/23
Incorporate notes feature	Joshna	WORKING	9/25/23	2	9/27/23
Incorporate recipe finding feature	Joshna	WORKING	9/29/23	3	10/2/23
Slack	Zeynep	NOT STARTED	10/2/23	4	10/6/23
Testing and improivng	Zeynep	NOT STARTED	10/6/23	5	10/11/23
Test with apple's accessibilty feature	Zeynep	NOT STARTED	10/6/23	3	10/9/23
User testing	Zeynep	NOT STARTED	10/9/23	1	10/10/23
Incorporate changes	Zeynep	NOT STARTED	10/7/23	5	10/12/23
Slack	Zeynep	NOT STARTED	10/13/23	8	10/21/23
Integrate with back-end	Zeynep	NOT STARTED	10/22/23	2	10/24/23
Refine	Zeynep	NOT STARTED	10/25/23	5	10/30/23
Testing and improivng	Zeynep	NOT STARTED	10/30/23	7	11/6/23
Test with apple's accessibilty feature	Zeynep	NOT STARTED	10/30/23	3	11/2/23
User testing	Zeynep	NOT STARTED	11/2/23	1	11/3/23
Incorporate changes and refine	Zeynep	NOT STARTED	11/3/23	7	11/10/23
Slack/Thanksgiving	Zeynep	NOT STARTED	11/10/23	17	11/27/23
Final integration and refinement	Zeynep	NOT STARTED	11/27/23	6	12/3/23
Back-end Software					
web-scraping for kraft mac n cheese from directions for me	Joshna	WORKING	9/10/23	18	9/28/23
basic python/html framework	Joshna	WORKING	9/25/23	3	9/28/23
testing and improivng webscraping	Joshna	NOT STARTED		0	
convert string to braille characters	Joshna	NOT STARTED		0	
test string to braille		NOT STARTED		0	
create gcode file with movements	Joshna	NOT STARTED		0	
test braille to gcode		NOT STARTED		0	
Hardware Systems					
Design Solenoid System	Becky/Zeynep	DONE	9/17/23	14	10/1/23
Initial Ciruit Design for 3 solenoid embossing system	Becky/Zeynep	DONE	9/17/23	3	9/20/23
Simulations in LT Spice to test power limitations	Becky/Zeynep	DONE	9/20/23	3	9/23/23
Refine and adjust initial design	Becky/Zeynep	WORKING	9/22/23	3	9/25/23
Design print heads	Becky	DONE	9/17/23	14	10/1/23
CAD embosser "print head" prototypes (1 vs 2 vs 3 vs 6)	Becky	DONE	9/17/23	3	9/20/23
Design x/y plotter	Becky	DONE	9/17/23	14	10/1/23
CAD x/ty plotting system	Becky	DONE	9/17/23	3	9/20/23
Combine solenoid and plotter designs	Becky/Zeynep	DONE	9/25/23	4	9/29/23
Design control circuit	Becky	DONE	9/26/23	5	10/1/23
Slack	Becky/Zeynep	NOT STARTED	10/1/23	3	10/4/23
Refine initial designs	Becky/Zeynep	WORKING	10/4/23	8	10/12/23
Slack	Becky/Zeynep	NOT STARTED	10/13/23	8	10/21/23
Begin prototyping of system	Becky/Zeynep	WORKING	10/21/23	3	10/24/23
Fabricate and test print head prototypes	Becky	WORKING	10/24/23	3	10/27/23
Fabricate enclosure	Becky/Zeynep	NOT STARTED	10/27/23	3	10/30/23
Fabricate and assemble parts of x/y plotting system	Becky	NOT STARTED	10/30/23	3	11/2/23
Test x/y plotting system	Becky/Zeynep	NOT STARTED	11/2/23	4	11/6/23
Stress testing	Becky/Zeynep	NOT STARTED	11/2/23	4	11/6/23
Revisions and demo preparation	Becky/Zeynep	NOT STARTED	11/4/23	12	11/16/23
Begin integration	Becky/Zeynep	NOT STARTED	11/16/23	6	11/22/23
Design system to communicate w hardware via wifi	Becky/Zeynep	NOT STARTED	11/20/23	7	11/27/23
Slack/Thanksgiving	Becky/Zeynep	NOT STARTED	11/22/23	5	11/27/23
Final integration and refinement	Becky/Zeynep	NOT STARTED	11/27/23	6	12/3/23

Figure 15. Schedule