B-1: Aware-ables

Authors: Chester Glenn, Jong Woo Ha, Kevin Xie
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— Braille is an exclusively tactile system of embossed or raised dots that allow individuals with impaired vision to have similar access to inscribed forms of information. However, given how over 90 percent of legally blind Americans are braille illiterate [5], a means to mitigate this situation is imperative for guaranteeing adequate education to support a strong career, as well as navigational assistance. Aware-ables is a solution that aims to provide those with impaired vision with the ability to read braille regardless of their education. The proposed solution will use a head-mounted camera to capture an A4-sized braille document at arms-length, then translate and read the contents back to the user. It is our hope that this solution will alleviate some of the current restrictions experienced by legally blind individuals.**

***Index Terms*— Braille, Accessibility, Wearables, Optical Character Recognition (OCR), Text-To-Speech**

1 INTRODUCTION

Historically, braille literacy in the United States has been on a sharp decline, and fewer than 10 percent of the legally blind Americans are braille literate [5]. Therefore, the vast majority of the visually impaired individuals can not fluently read braille text, embossed on paper or other surfaces, meant to provide pivotal guidance and assistance. Given that braille is such an essential form of written language for the visually-impaired individuals in both education and navigation, a device that provides auditory accessibility and assistance by means of text-to-speech translation could bring about a meaningful turnaround. In order to assist visually impaired and legally blind readers in reading braille as well as to improve braille literacy overall for educational purposes, Aware-ables will take the form of a wearable device equipped with a mounted camera used to capture braille text a fixed distance away, then translate said braille to the user via a pair of speakers located near their ears with a single button click on the side of the device.

Through the subsequent usages of computer vision algorithm for braille capturing and pre-processing, ML pipeline for character recognition, spell-check algorithm for post-processing, and an external text-to-speech API, Aware-ables will ensure a smooth translation of a full A4 page braille text within 2 seconds of button activation. Currently, there are different devices that can translate English text to braille or translate digital/limited real-world braille text to English text, but no mode of direct translation of braille to speech is provided within the open US market. Aware-ables hopes to not only provide convenient translation within 2 seconds, but also ease out the learning curve of the braille language in the long run.

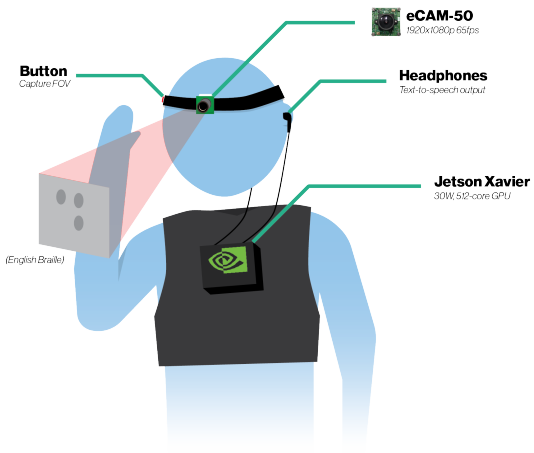

Figure 1: Initial vision for final demonstrable product (Aware-ables)

2 USE-CASE REQUIREMENTS

In order for Aware-ables to effective in our suggested context, two core requirements that must be guaranteed to the users: a maximum of 2 seconds of translation latency and over 90 percent translation accuracy.

For a relatively convenient and uninterrupted experience, the entire process from braille capturing to direct speech translation will be completed within two seconds, following the common usability standard for loading wait times [8]. Furthermore, braille readers can read at speeds ranging from 200 to 400 words per minute[2], Aware-ables will aim to match this pace by recognizing up to 10 words each two seconds at arms-length, reaching a maximum of 300wpm. However, it is important to note that any rate of speech over 200wpm can significantly impair comprehension [4]. Given that our chosen medium of delivery is speech, we will need to tune this rate for improved comprehension.

As far as accuracy is concerned, we are targeting a 10 percent character error rate to match the conventional error rates of traditional optical character recognition (OCR)[6], which will be further alleviated through post-processing and spell-check algorithms. Our final target word error rate is less than ten percent, to ensure comfortable and accurate playback for educational purposes.

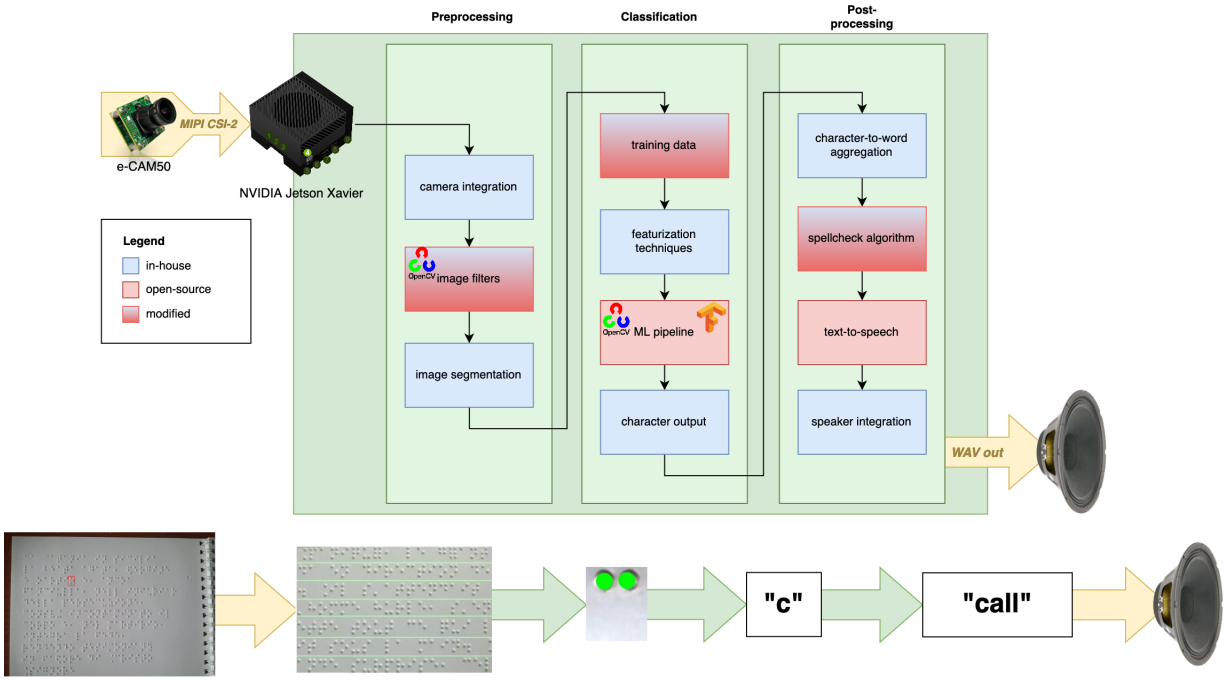

Figure 2: High level block diagram and datapath for our proposed solution.

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

Fig. 1 represents the team's vision for our final prototype. The hardware architecture will include the eCAM50_CUNX, a Jetson-compatible camera which is capable of 5 megapixel (2592 x 1944) video capture at up to 28fps; an NVIDIA Jetson, a semi-portable form-factor platform powerful enough to support real-time capture and inference; a button; and a mode of audio delivery. The eCAM50 is also capable of lowering resolution to increase framerate, which we will consider tuning as we test. On triggering the button, a still of the video feed is sent to a Jetson running our software stack to process the input. Once the braille has been translated, the result is read out of an audio device connected to the auxiliary port (audio output jack).

For wearability, we have chosen to secure the relatively unwieldy Jetson (274g) device on a vest connected to a head-mounted camera. To best capture the point-of-view of the visually-impaired user, we have chosen to mount the camera on a headband centered on the user's forehead. Anecdotally, we expect users to position the angle of their head to point toward the braille they are attempting to read.

Above, Fig. 2 presents a high-level block diagram for our intended implementation. As previously mentioned, our software stack is split into three successive subsystems: pre-processing, classification, and post-processing. Later sections will dive into more detail about implementation specifics, however it is important to note the color coding of the blocks indicating which software components will be sourced off-the-shelf and which will be developed in-house.

Below the block diagram, we have provided a high-level visualization of modifications being made to the input image at significant points in our datapath, however, here again later figures will provide more detail. From the high-level diagram, it is clear that our software stack will expect an (1) uncropped, well lit image of a braille document, which will then be (2) cropped, filtered, and segmented into single characters, then (3) classified and (4) concatenated into an English word, which can then be (5) read out via the speaker.

4 DESIGN REQUIREMENTS

4.1 Pre-processing

The first subsystem in our software data path is the pre-processing of captured braille images through computer-vision and segmentation algorithms. Image of printed braille text will be captured using CMOS camera, ecam-50 with a resolution of 2592 pixels by 1944 pixels, and trigger button with the distance between camera and braille text being approximately 30cm apart in order to adjust the dimension of the initial physical crop to match that of the A4-sized paper. The original image will then be pre-processed through various computer vision algorithms of OpenCV libraries in order to increase the overall quality of the collected image, facilitating the next process, recognition, that utilizes machine learning classification models. The pre-processed image will then be horizontally and vertically segmented, with the results being a folder of individually cropped rectangular braille characters that have been

processed and segmented from an initial braille capture to be fed into ML pipeline in the next procedural step.

The primary design requirement of this subsystem is adjoint with the classification stage's necessities in that Aware-ables is aiming for less than 10 percent error rate for each optical character recognition. In order to attain this, various approaches to computer vision pre-processing are studied in the design studies section to not only maximize the quality of final cropped braille character images , but also minimize character recognition error rate.

4.2 Classification

The second subsystem in our software data path is the classification model. This subsystem should accept an array of images of segmented and pre-processed braille symbols as described above and output an array of translated english characters in the same order. The primary design requirement for this subsystem is a character error-rate of under 10%. There are a few reasons for this design requirement. An article from early 2021 cites an Australian study which places average OCR character error-rate in a range of 2-10% [6]. Because braille recognition is a relatively smaller field of research with a number of unsolved technical challenges, we are setting expectations at the higher end of this range. Furthermore, we expect the post-processing step to correct some of the errors that may arise from classification.

In line with latency requirements, we expect classification to be fairly low-latency on our chosen hardware. Therefore, we will not be specifically optimizing classification for latency, instead putting emphasis on whether it is as accurate as possible.

4.3 Post-processing

Our final software subsystem is post-processing, which includes concatenation, spellcheck, and text-to-speech generation. A 2013 study performed by Lund et al. showed that OCR character error-rate had a detrimental effect on word error rate, with a 1.4% CER resulting in a word error-rate of up to 7% [7]. However, the paper does not describe whether spellcheck or other post-recognition corrections could be utilized to reverse this impact. We hope to use an in-house spell checking algorithm to lower word error rate when compared to character error rate.

In order to minimize the overall likelihood of falsely recognized characters from the prior classification subsystem, we are aiming for a final accuracy of <5% for all words.

5 DESIGN TRADE STUDIES

5.1 Pre-Processing

Upon capturing the original braille image, following steps [9] are required to pre-process, or to properly refine the original image so that the ML recognition model can correspondingly translate individual images of Grade 1 braille alphabets to English characters with the desired character accuracy rate of 90 percent:

1. Grayscaling
2. Thresholding
3. Application of various blur filters
4. Erosion and Dilation
5. Further application of edge filters such as canny edge filters using non-max suppression

In the figures below, further details of each procedures and reasoning behind adoption of specific functions from OpenCV libraries will be explored.

Initially, in step 1, cv2.imread() method is used to load an image from the specified file location in order to load the original image of braille text, which is the most efficient way to load an image file to python. In step 2, grayscaling, cv2.cvtColor(src,code[,dst[,dstCN]]) is used to convert the original image comprised of RGB color scale into gray scale images, which is a necessary prerequisite for thresholding step.

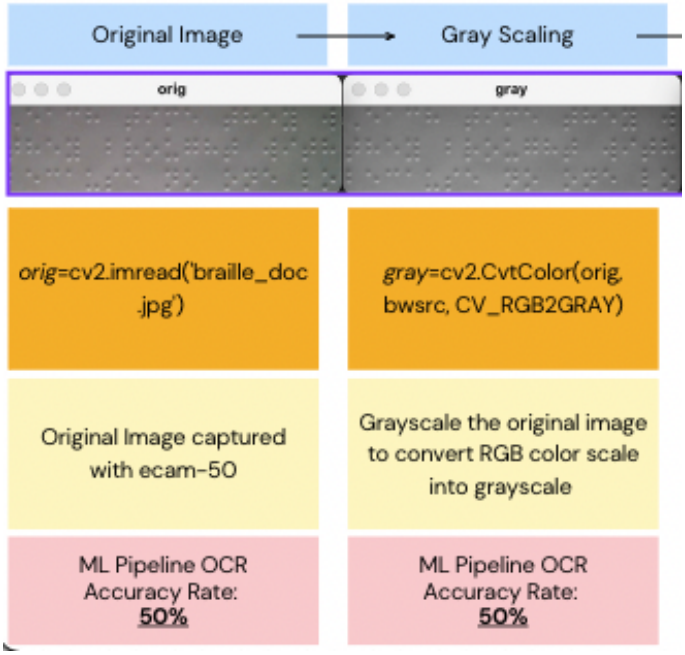

Figure 3: Results of Step1: Capturing of the original image, and Step2: Gray-scaling of the original image

Third step, thresholding, is a process of converting the grayscaled image into a binary image that is only comprised of color scale values 0 and 255. Because the input image is captured from e-cam 50 which accommodates various intensities of each pixels unlike scanned documents, OpenCV's adaptiveThreshold method is adopted over regular threshold to support a range of maximum value of 255 and minimum value of 0. The purpose of the fourth step, blurring, is to reduce unwanted noise by applying various

blur filters such as median blur, Gaussian blur, or bilateral filtering. Applying Gaussian Blur only have reduced the number of unwanted noise count from 50+ to 17, and further application of median blur have reduced the noise count to 8 in the tested image described in Figure 4.

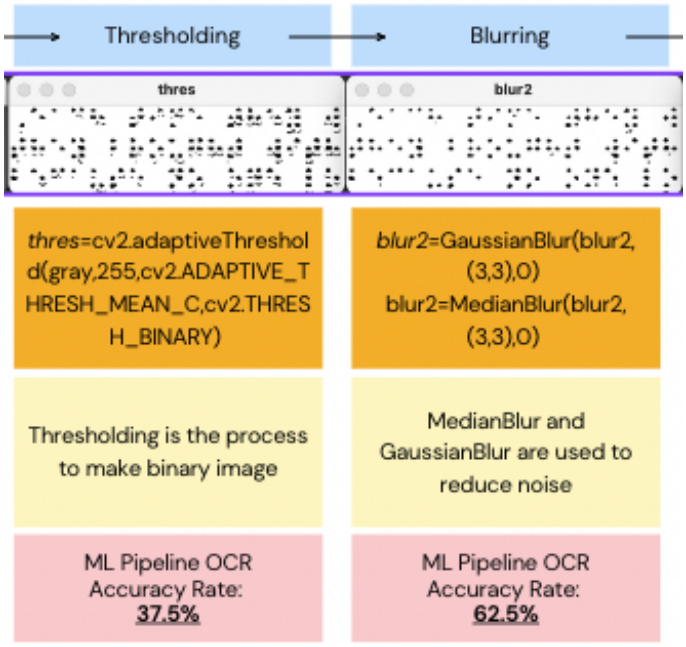

Figure 4: Results of Step3: Thresholding the gray scaled image to create a binary image, and Step4: Application of both median blur and Gaussian blur on binarized image to reduce unnecessary noise

During step 5, erosion, a process of equating the pixel values inside to the outside value, and dilation, a process reverse of erosion, take place to reduce further noise. Erosion procedure is to minimize the area of the black dot and remove remaining noise from thresholding that are still left after blurring. Then, the shrinked dots are enlarged through dilation which allows re-visibility of braille dots along with the elimination of noises left behind. After completion of the erosion and dilation, the 8 remaining noises have reduced to 6. In the last step, canny edge filters are applied through non-maximum suppression, or by collecting the center point coordinates of individual braille dots and drawing a colored circle with a radius found from Hough Transform. This method is adopted to maximize the edge contrast of the final pre-processed image, which will facilitate the next ML recognition process.

As OpenCV library's functions ensures optimization and performance, the entire pre-processing procedure can be completed between 200 to 400ms, and subsequent execution of necessary steps will gradually increase the OCR accuracy rate tested during the ML recognition phase, approaching near the desired rate of 90 percent as shown in figures 3,4,5.

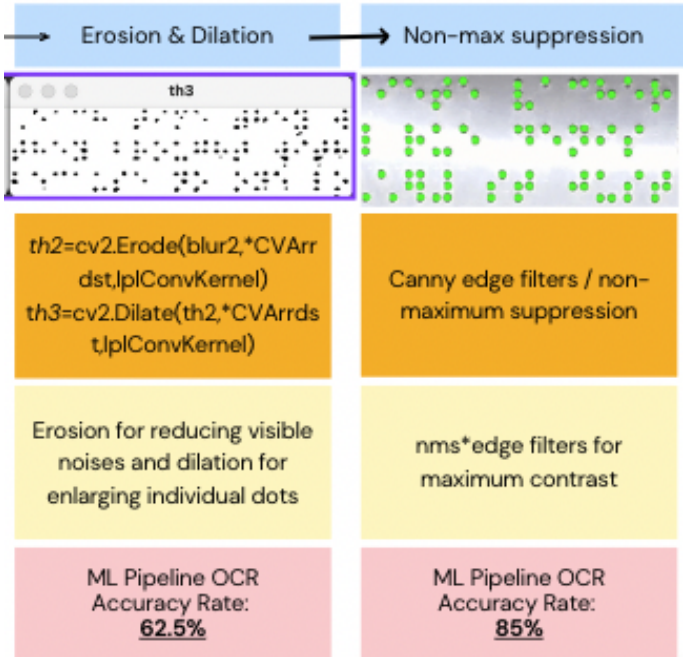

Figure 5: Results of Step5: Erosion and dilation for further reduction of discrepancies by reducing(eroding) visible noises and enlarging(dilating) individual dots, and Step6: Application of further non-max suppression edge filter algorithm for enhanced OCR accuracy rate

5.2 Classification

One important consideration made when designing the classification subsystem was whether to use a pre-trained model or train a new model in-house. As part of our classification subsystem design, we performed a keyword search for existing open-source Braille classification models and libraries. The most effective model available [3] provided a pre-trained model with two sets of layers (excluding input and output), each with a 2D convolutional layer, a pooling layer, and a ReLU (rectified linear unit). Testing this model against its own 20,000 image dataset, we were able to achieve, on average, an 89.6% character accuracy, with the lowest individual character accuracy being 73.5%. However, since this model was likely trained against the same dataset, this experiment tested the best case scenario for the model.

While this model represents a convenient off-the-shelf option for our project, we do not expect to use it in our final prototype. As cited above, testing against the training dataset is not representative for validating the model in the real world. However, even so, the model was not able to reach our design requirement of a 10% character error rate. Combined with the opportunity to tailor our training dataset to better represent the output of the preprocessing subsystem, this provides ample motivation to train our own in-house model. The tradeoffs for this decision will be time and resources spent training a new model. However, we hope that we are able to gain better accuracy in the context of our solution, as well as clearer knowledge of dataset

division for cross-validation testing. While we are moving forward with an in-house solution, we may experiment with transfer learning using this model as a foundation in the future.

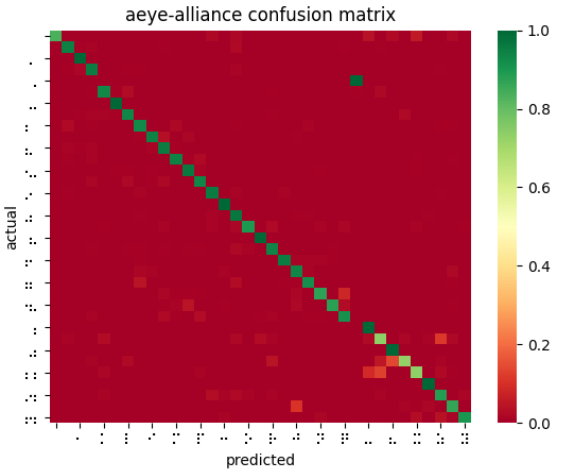

Figure 6: Abbreviated confusion matrix for a pre-trained classification model. Green diagonal line indicates high confidence tested against training dataset.

We also examined the tradeoff between using a neural network and a classical tree-model machine learning algorithm. From our review of pre-trained models available on the web, we found that neural networks, deep or convolutional, were far more prevalent than tree-models for image classification. One possible reason for this is that tree-models are generally geared towards tabulated data. Furthermore, neural networks gain far greater accuracy from training on larger, more complex datasets without overfitting or needing to be pruned. The downside of using a neural network is that the resulting models and weights are often opaque, making it difficult to test and verify possible edge cases. Furthermore, it requires more pathways to be activated before an inference can be made, leading to higher latency and power consumption when compared to tree-models.

Because we are working on an AGX Xavier to begin with, we believe we have the computing overhead to use a neural network. In our initial experiments, we have seen inference times between 10-30ms for a given segmented image. Choosing neural networks allows us to use existing literature on OCR and braille recognition as a foundation for our classifier. Using our customized training dataset, we believe that it will be easier to train and tune a neural network to reach an error rate requirement of 10%. However, here again we will reevaluate our options to optimize once we have a working demonstration. For example, we may choose to manually extract features based on certain heuristics (bump count, average bump position, etc.) and feed the resulting feature vector into a tree-model.

5.3 Post-Processing

From a workload perspective, the post-processing subsystem is broken into the inflow of characters, error-checking of concatenated words, and lastly the text-to-speech. The main consideration before beginning the infrastructure of both software and hardware components was understanding the individual complexities of each subsection in the overall subsystem. The first subsection is trivial and can be designed by hand. The next two sections required a deeper understanding of algorithms however. At a deeper level the spell-checking section only requires an understanding of software and general algorithmic thinking. This can be written by hand with medium complexity, but it will not sacrifice in any usability when done correctly. On the other hand, the difficulty of creating fluid text-to-speech incorporates software, hardware, embedded systems, and a deep understanding of signal processing.

In consideration of the skill sets of our team, the decision came to writing the error-checking by hand and leaving the text-to-speech application to an API. In terms of usability, this decision gives us access to a pre-trained and more realistic model based on natural language processing. The three main benefits of using an API is understandibility, efficiency, and complexity. An API has existing voice infrastructure that resembles natural speech, and also in a highly efficient manner since it is scaled by larger budget corporations like Google. If we were to write the text-to-speech by hand, it would most likely result in a crude interpretation that is hardly understandable and barely scaleable at low speeds.

6 SYSTEM IMPLEMENTATION

This section provides the technical details for the implementation of each component of our solution.

6.1 Hardware

We expect hardware bring-up to be secondary to the development of our software stack. The eCAM50 communicates with the Jetson via a MIPI CSI-2 connector. Because the AGX Xavier does not natively support MIPI CSI-2, we have ordered a CSI-2 to USB adapter from DigiKey to ensure that we are able to use the camera. Once the camera has been integrated and the feed can be readily accessed, we will use a button to trigger still capture of the feed to send to the pre-processing subsystem. Finally, we will use a USB to auxilary interface for audio output.

Once we have achieved our minimum viable product, we hope to begin optimizing hardware size and power efficiency. One of the ways to do this would be to transition our platform to a Jetson Nano. The Jetson Nano solves many of the problems described above, such as native MIPI CSI-2 support, aux jack, and WiFi connectivity (also absent on the AGX Xavier). However, we have chosen to begin by working on the AGX Xavier to maximize the performance overhead for our software stack.

6.2 Pre-Processing

The primary deliverable of our pre-processing and segmentation algorithms is the folder of individually cropped rectangles each containing one braille alphabets to be passed on to the ML recognition model for direct braille character to English character translation. In order to properly pre-process the initially captured braille image into an individually cropped braille alphabets, two steps need to be followed: 1) pre-processing of the original braille image using computer vision algorithms to increase the overall quality of the captured image. And,2) vertical and horizontal segmentation to crop the pre-processed image into individual rectangular boxes of braille alphabets. Detailed implementation details of step 1 is already covered in section 5.1) Design Trade Studies of Pre-processing Subsystem.

The pre-processed braille image then needs to be vertically and horizontally segmented. For either vertical or horizontal segmentation, two end points of each line needs to be known, and Hough transform allows a binary edge map to be taken as input, locating the coordinates of edges placed as straight lines. Before applying Hough Transform for line detection, pre-processed images have their edges defined with OpenCV Canny edge detectors and the Hough Lines are drawn in between each edges that comprise each lines, practically segmenting the processed braille image into (row x col) rectangles. Now that the coordinates of each rectangular boxes can be identified, individual boxes can be cropped using OpenCV's numpy slicing, resulting in a following folder of processed and cropped braille alphabets ready to be sent to ML pipeline.

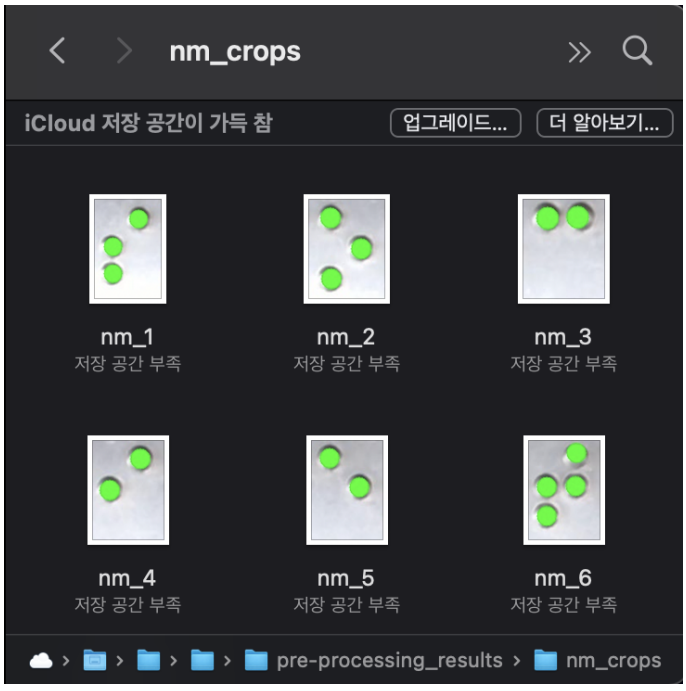

Figure 7: Folder of processed and cropped rectangles of braille alphabets, to be sent to ML pipeline for classification

6.3 Classification

As mentioned in section 5.2, our classification model will be a neural network trained on a variation of the 20,000 image dataset provided by [3] that has been passed through filters to resemble the output of our pre-processing pipeline. The dataset will be divided into sets with which to train and sets with which to test and validate. By training and testing with different sets, we guarantee that our model will not be overfitted or biased to the data it is trained with, and is instead a valid general solution for classifying braille. Our choice of neural network hidden layer configuration is guided by the existing solution as well: we will be normalizing the dimensions of the image to 28 x 28, then inputting the image into an input layer of 3 x 28 x 28 (3 color channels), which will run through two sets of hidden layers (2D convolutional, pooling, rectified linear unit) and through an output layer of 37 classes of braille symbols. We will make adjustments and optimizations based on two heuristics: (1) hidden layer nodes should be close to sqrt(input layer nodes * output layer nodes) and (2) to keep on adding layers until test error does not improve any more.

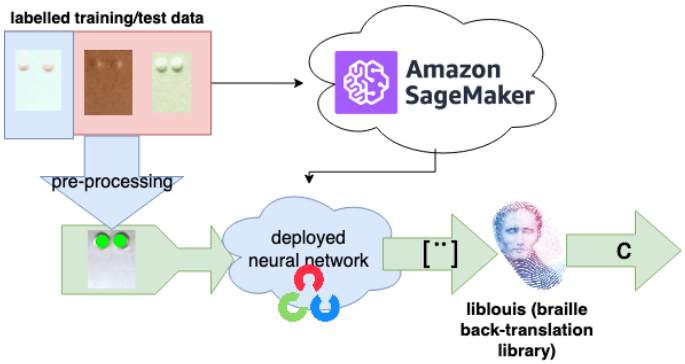

Figure 8: Block diagram for classification subsystem

We have initially been training simple models on local hardware (RTX 3080). However, long training times have become a blocker and we are investigating cloud training solutions like AWS SageMaker. To deploy the trained model, we will be using OpenCV's built-in functionality to maintain consistency between pre-processing and classification subsystems, reducing the overhead of adapting data classes between frameworks. According to initial testing, we are able to achieve between 10-30ms of latency for each inference.

Unlike existing OBR solutions, we will not be classifying braille characters directly to their english translations. Instead, we are relabelling our dataset to correspond to the braille symbols, then sending the inference through an open-source braille translation library called LibLouis (http://liblouis.org/) to translate from the braille symbol to English. By first classifying the braille character as its unicode symbol equivalent, we leave room to extend our solution to different languages of braille, which all use the same symbols but translate to different written characters.

The classified symbols will be outputted to the next subsystem in the data path as an array of English characters.

6.4 Post-Processing

The final large subsystem includes the post-processing to speech application. The main deliverable from the previous sub-process is ascii characters which have been individually characterized by the ML pipeline. The post-processing deliverable provides the final output to the user in clear and concise audio. As a whole, the post-processing section includes 3 main components. The first and relatively simple action is to take in characters and concatenate into words. The second step involves the continuous checking of concatenated words against the english dictionary for basic error correction. Lastly, the text will be converted to speech via a REST API.

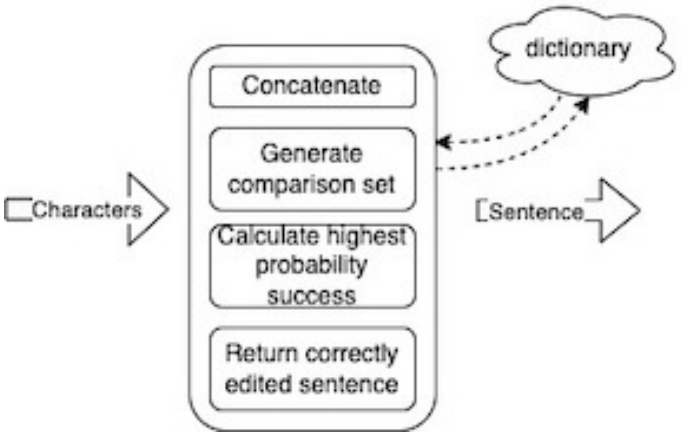

Figure 9: Subsystem C error checking flow chart.

In order to create a working spellcheck model, several key factors have to be considered. The first and most obvious implication is the dictionary of choice. A single reference source file for each word to be checked against efficiently and within a reasonable time frame. For this consideration, a static text file will be allocated with the full dictionary alphabetized and ordered. This dictionary can be used to compare against each word before proceeding forward with error correction. In terms of error correction, we opted for a simplistic model that would be beneficial for both the necessity for efficiency and relative error rates in classification. This model assumes that the possibility of errors resulting in one character less or one character more for a word are 0%. Using this theory we can greatly minimize the set of possible words resulting in a single character error. Due to the sheer quantity of possible combinations of words with 2-character errors, and the relatively low error rate of single characters that we are aiming for, it is safe to say statistically that words will only see errors once every 10 characters.

The main algorithmic approach to the spellchecking subsection involves generating the set of all possible single character different "words", that are also contained in the static dictionary. Initially, the set of words with one single character different is quite large, however, this is greatly reduced when considering words that are in the English dictionary. Overall, each word will then be assigned a probability of correction and the max word will be returned and re-added to the sentence.

After each word has been processed and the text is completely synthesized, the data will be sent as a request to the Google REST API, and a respective .WAV file will be returned. The subsequent .WAV file will be processed and sent through the speaker or headphones connected to the device as stereo output. With the addition of the Google API, there are multiple features that allow for custom manipulation of speech. This includes custom voice recognition, as well as pitch and volume manipulation for optimal user interpretation.

7 TEST & VALIDATION

Use-case Requirements: In order to validate the usability of our product as a real-world application, there are several key metrics that need to be measured. For testing, the two categories of focus are Latency (wpm), and Error rate (% incorrect). As defined in both the design and use-case requirements, these are the metrics that will determine the overall applicability to our end user. In addition to error rate and latency, several other considerations must be taken to ensure full usability.

Taking into account the user and the real-world, we must also consider wearability and average battery life of the product (hours). We believe these will be difficult to qualitatively test, and will be assumed upon initial calculation of battery life to average power draw. This being said, our testing process will require coordination between the pre-processing, classification, and post-processing subsystems as a whole. In order to validate error rate and latency of the product, the testing process will include multiple stages in a controlled environment which will be described further below for each individual test.

For the purpose of testing our solution in a controlled environment, our product will be mounted like depicted in Fig. 10. The camera will be aligned perpendicular to the ground such that it is pointing in a horizontally towards the selected document. In order to guarantee optimal lighting conditions, the environment will consist of an overhead light aligned with the camera and diagonally pointing at the document in front of the camera.

Design Requirements: In addition to the use-case testing, our product will undergo consistent validation to meet rigorous timing and error related constraints in subsystem modules. In order to maintain parallel components in software while obtaining similar latency described in the use-case, each subsystem must be tested individually.

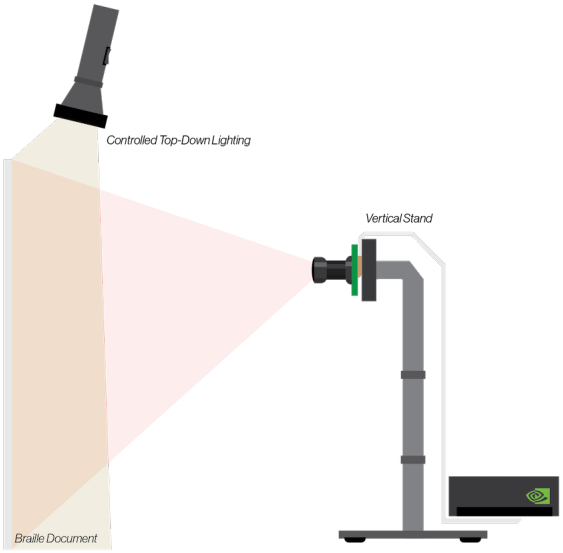

Figure 10: Controlled testing environment.

7.1 Tests for Latency

In order to validate the latency of our product based on the quantitative requirements set forth in section 2, 20 randomly chosen pages of 50 braille words will be tested in sequence. These braille documents will be sourced from the Library of Accessible Media (LAMP) and other public resources. Each page will be placed at the same distance from the camera 1.5 ft away, with the center point of the document aligned with the center of the camera. The latency of a single page will be determined based on the time from initial image capture to the completion of the last spoken word. Assuming a wpm spoken rate of 150 wpm will allow us to quantify the processing time required by subtracting the time spoken for 50 words.

7.2 Tests for Error Rate

Similar to latency validation, the test specifications for error rate will rely on a set of 20 randomly chosen pages of 50 braille words. Due to the nature of this experiment, this can be run in parallel to use-case A by quantifying the number of words incorrectly processed in comparison to the number of possible words. This will give us an overall sample of 50 words per page, and 1000 words processed total.

7.3 Tests for Design Specification A

In conjunction with the pre-processing subsystem described in section 6, one of the predicted errors involved in the image segmentation is cropping offset. In order to limit possible unnecessary false identifications in the ML pipeline, systematic dimension testing will be done on a set of unprocessed braille documents and compared to a standard. To guarantee accuracy and consistency, the success of this design testing will be measured in vertical and horizontal offset (mm) rather than by pixel. This model assumes the rotation is fixed as well.

7.4 Tests for Design Specification B

In parallel with the pre-processing done in subsystem A, the classification of characters will be verified through testing of a large subset of manually processed and cropped braille images. The bulk of our character error rate will most likely be incurred in the ML pipeline, which is why testing is essential for understanding the overall picture. During development and optimization of our neural network model, we will maintain an exclusive testing and validation set (that does not contain the training set data) for cross-validation of our machine learning model. Doing so will ensure that we are working toward a general solution that is not over-fitted or biased toward training data. Being able to control what data our model is trained on is one of the tradeoffs set forth in 5.

7.5 Tests for Design Specification C

Lastly, the characters identified through the classification subsystem will be checked as words in subsystem C. In order to measure validity of the spellchecking algorithm, sets of 50 words will be processed at a time with either no errors or single character offsets. The test is limited to single character offsets to mimic regularity in classification as well as limit the processing speed needed by the spellchecking algorithm. Success will be measured in both the false positive rate as well as ability to correctly return the edited incorrect words.

8 PROJECT MANAGEMENT

8.1 Schedule

Our development cycle is split into three phases. Phase one focuses on the research and development of concept/requirements. This is the initial proposal period when pitching the overall use-case and idea for our product. This phase spans two weeks, and transitions into phase two, design. The design phase is focused on coloring in the specifications of our solution with more detail for both our software and hardware components. This phase lasts a total of 3 weeks, culminating in the design review and report. The most significant portion of our project is spent on the development of the solution. Through parallel workflows, each individual develops their section of software with the understanding of the deliverables from the preceding subsystem as well as what is to be delivered to the next subsystem. The breakdown of work will be further described in the following subsection. The schedule is shown in Fig. **??**.

8.2 Team Member Responsibilities

There are three subsystems involved in the full product's main processing, and the overarching hardware de-

velopment which is split between all three members. For the first subsystem, Jong Woo Ha is responsible for coordinating the pre-processing of images through openCV. The deliverable is an array of cropped single braille characters under non-max suppression filtering. Kevin Xie is then in charge of classifying individual images into characters using a ML pipeline. Lastly, Chester Glenn is responsible for the text-to-speech integration, which involves error-checking words and a smooth text-to-speech application.

8.3 Bill of Materials and Budget

The estimated Bill of Materials and overall cost of the project are included in Table 1.

8.4 Risk Mitigation Plans

One risk we considered when initially choosing to pursue this project is the limited experience that our team members had with computer vision and machine learning. As machine learning is at the core of our project for classification and computer vision for pre-processing, one way we have chosen to mitigate this risk is to assign team members to specifically focus on these concepts. By assigning "topic specialists" instead of spreading expertise among team members, we hope to be able to improve development efficiency and decrease areas of risk between parallel workloads.

9 RELATED WORK

From the initial brainstorming to the creation of our relatively finalized design, there were several industry paths that branched off from the key components of our product. In the current field, OCR, Machine Learning, Augmented Reality, and accessibility technology are all well-researched disciplines. Although our product may not necessarily represent all of these fields to a large degree, there are segments of each that can be compared to our initial creation and have also been used in part as inspiration to the design.

There are various assistive technology products listed on the websites of the American Foundation for the Blind (AFB) and National Federation of the Blind (NFB), including various types of braille translators such as BrailleMaster, Duxbury Braille Translator, GOODFEEL Braille Music Translator, or Toccata [1]. All of these translators execute quick and accurate of braille to text or braille to text translations or even translate music to braille music, but braille to speech translations is not supported.

10 SUMMARY

The goal of the Aware-able product is to aid in the reading and understanding of braille literature for both visually impaired individuals as well as the generally braille-illiterate sighted population. This product will be beneficial in improving overall braille literacy rates, and therefore information symmetry, among all individuals. This product can also be used as a teaching tool for non-visually impaired persons to learn the language.

Moving forward, there are still several challenges that we will need to overcome in order to meet our requirements. In terms of latency and error rate calculations, these are both general challenges that we are cognizant about. One of the hardware constraints we need to keep in consideration are the necessity for WiFi in our Jetson, as well as making sure the power draw and battery life are within reasonable use-case requirements. The initial product design uses the Jetson AGX Xavier, which requires Ethernet to be connected because it does not provide WiFi. The alternative suggested in 6 to swap platforms to the Jetson Nano would provide WiFi and better efficiency, but at the cost of limited performance in comparison.

Once we've completed our minimum viable product as described in this report, we hope to tackle these further technical challenges in a measured and intentional manner, ultimately in service of creating the ideal solution for improving braille literacy and navigational awareness for the visually impaired.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- BOM - Bill of materials
- CER – Character error-rate
- OBR - Optical braille-recognition
- OCR – Optical character recognition
- WER – Word error-rate

References

[1] American Foundation of the Blind(AFB). In: *Home/ Blindness and Low Vision / Using Technology / Assistive Technology products* (). URL: https:https://www.afb.org/blindness-and-low-vision/using-technology/assistive-technology-products/braille-translators.

[2] Susan Ford. "Braille Reading Speed". In: *National Federation of the Blind* (). URL: https://nfb.org/images/nfb/publications/bm/bm99/bm990604.htm.

[3] Helen Gezahegn. *Aeye (Ai4socialgood final project)*. URL: https://github.com/HelenGezahegn/aeye-alliance.

[4] Roger Griffiths. "Speech Rate and Listening Comprehension: Further Evidence of the Relationship". In: *TESOL Quarterly* 26.2 (1992), pp. 385–390. DOI: https://doi.org/10.2307/3587015.

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Jetson AGX Xavier	F21072	NVIDIA	1	$0.00	$0.00
e-CAM50_CUNX	F21004	e-con Systems	1	$0.00	$0.00
ADAPTER CSI-2 TORADEX IXORA CARR	14908	Allied Vision, Inc.	1	$33.75	$33.75
Tactile Switch SPST-NO Top Actuated	1241.1055.8000	SCHURTER Inc.	1	$4.50	$4.50
USB External Stereo Sound Adapter	AU-MMSA	SABRENT	1	$8.99	$8.99
Apple Wired EarPods	057-00-2070	Apple	1	$0.00	$0.00
					$47.24

[5] Jernigan Institute. "The Braille Literacy Crisis in America". In: *A Report to the Nation by the National Federation of the Blind* (Mar. 2009). URL: https://nfb.org/images/nfb/documents/pdf/braille_literacy_report_web.pdf.

[6] Kenneth Leung. *Evaluate OCR output quality with character error rate (CER) and word error rate (WER)*. 2021. URL: https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510.

[7] William B. Lund. "Combining Multiple Thresholding Binarization Values to Improve OCR Output". In: *Proceedings of SPIE - The International Society for Optical Engineering* (Feb. 2013). DOI: https://doi.org/10.1117/12.2006228.

[8] Jacob Nielsen. "Response Times: The 3 Important Limits". In: *Nielsen Norman Group* (Jan. 1993). URL: https://www.nngroup.com/articles/response-times-3-important-limits/.

[9] Joko Subur. "Braille Character Recognition Using Find Contour Method". In: *Conference: 2015 International Conference on Electrical Engineering and Informatics (ICEEI)* (Aug. 2015). URL: https://www.researchgate.net/publication/308829784_Braille_character_recognition_using_find_contour_method.

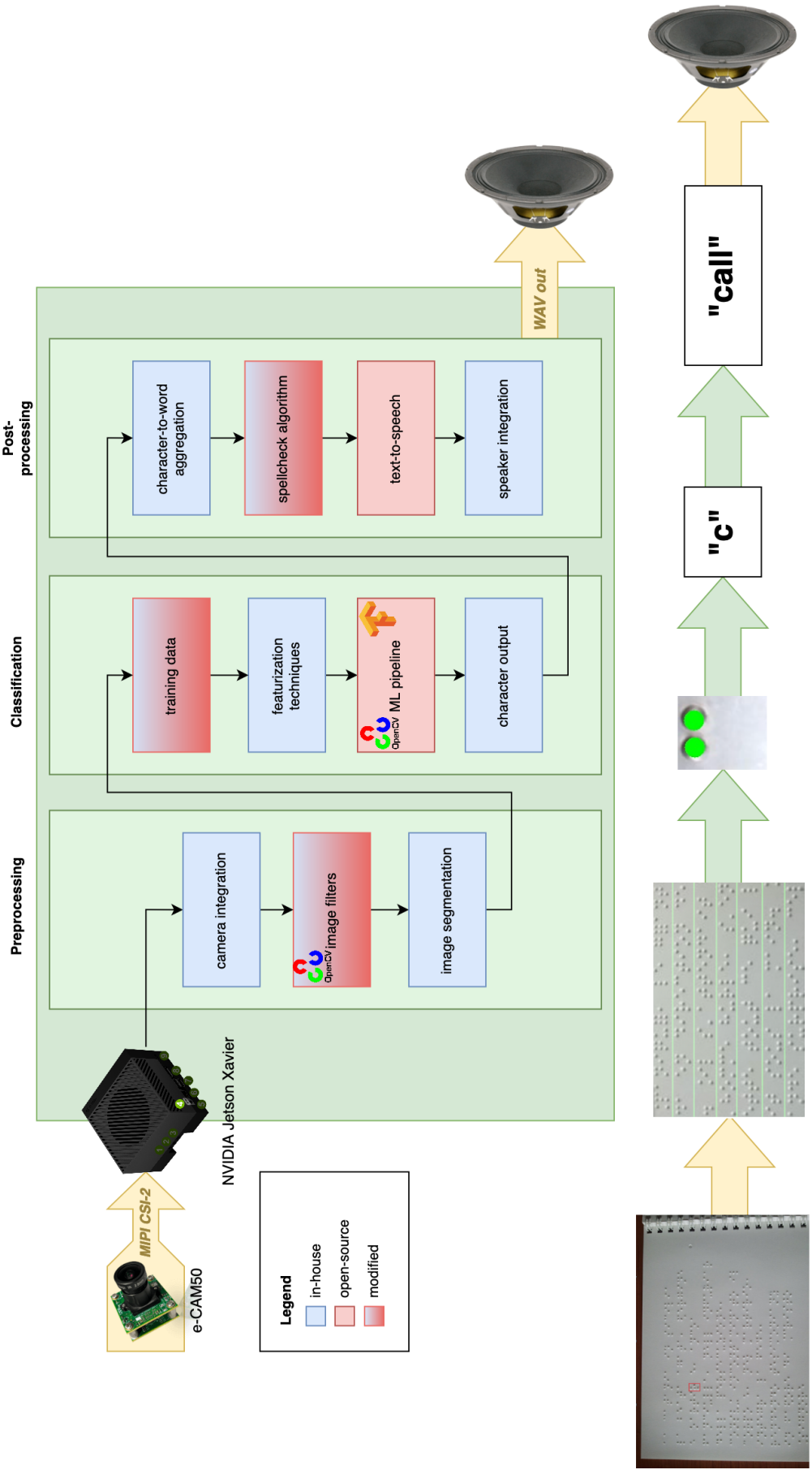

Figure 11: A full-page version of the same system block diagram as depicted earlier.

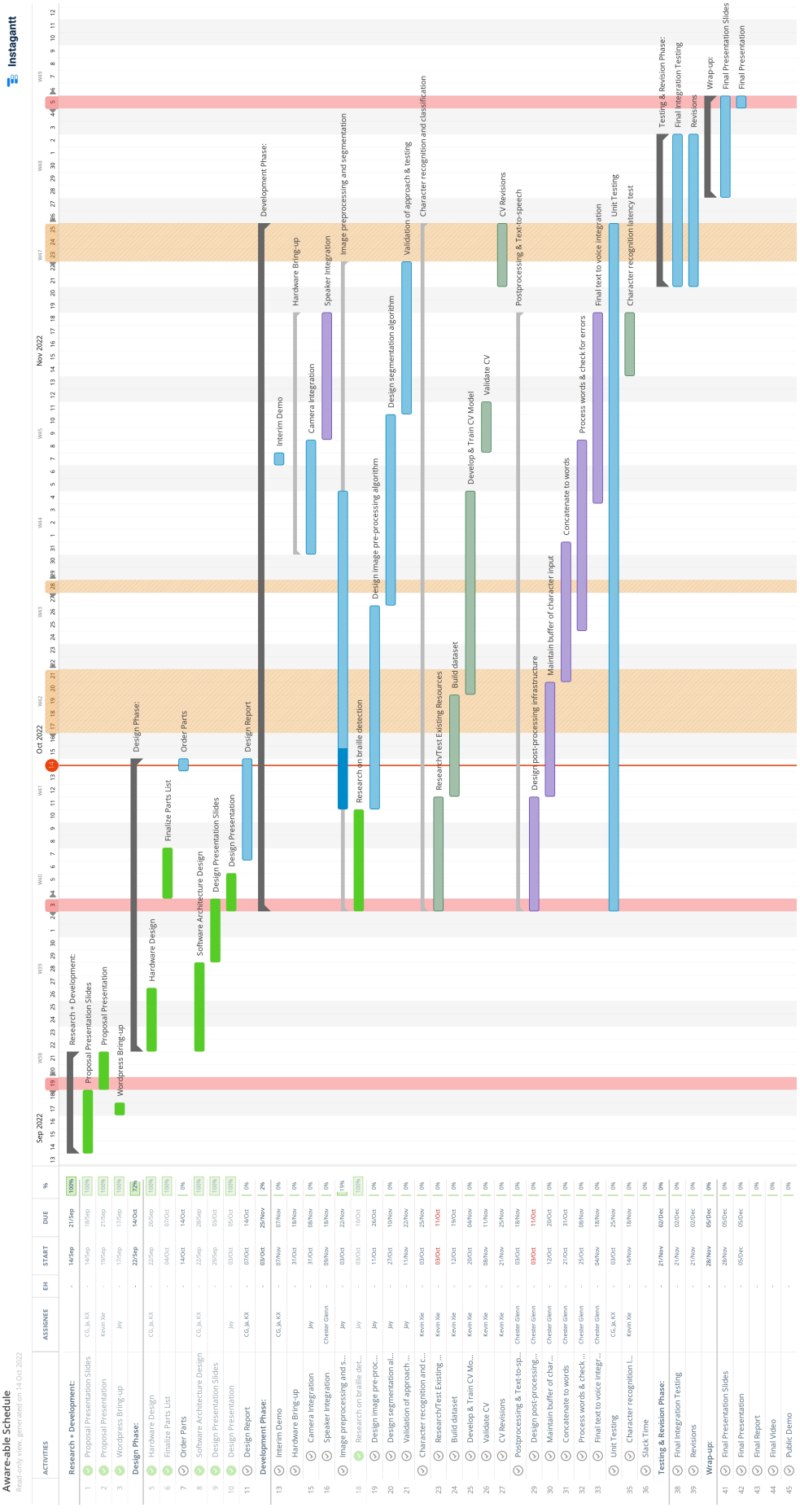

Figure 12: Our Gantt chart is split into three distinct phases and parallelized between subsystems.