B-1: Awareables

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 allows individuals with impaired vision access to inscribed forms of information. However, given that less than 10 percent of legally blind Americans are braille literate, a means to make braille learning more accessible is imperative for guaranteeing adequate education to support a strong career. Awareables is a solution that aims to provide those with and without impaired vision with the ability to read braille regardless of their education. While originally planned to be a wearable, the completed solution is a stationary appliance that can capture an A4-sized braille document, then translate and read the contents back to the user. It is our hope that this solution will alleviate some of the current educational disadvantages experienced by legally blind individuals.**

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

1 INTRODUCTION

Historically, braille literacy in the United States has been on a sharp decline, and recent published statistics have shown that fewer than 10 percent of the legally blind Americans are braille literate [1]. Therefore, the vast majority of the visually impaired individuals can not fluently read braille text, embossed on paper or otherwise, 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 was originally envisioned to take the form of a wearable device (Figure 1) 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.

The final solution presented in this report is rescoped as a stationary appliance (Figure 4) intended to provide convenience and education in classrooms and libraries. Using computer vision algorithms for braille pre-processing, machine learning for character recognition, bayesian spell-check 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. While there are currently a variety of devices and software packages that can translate English text to braille or vice versa in a limited capacity, no mode of direct translation of braille to speech is provided within the open US market. Aware-ables is designed to not only provide convenient translation within 2 seconds, but also ease out the learning curve of the braille language in the long run.

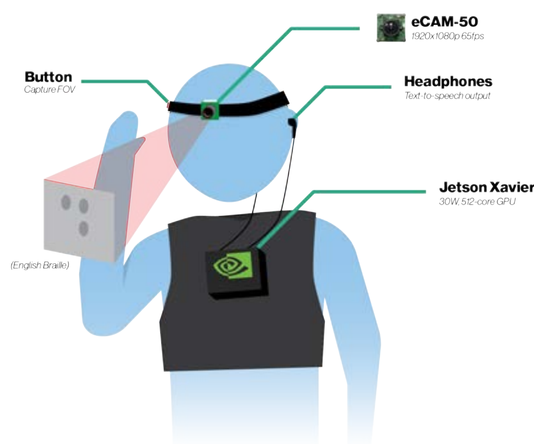

Figure 1: Initial vision for final demonstrable product (Aware-ables)

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 web wait times [2]. Furthermore, braille readers can read at speeds ranging from 200 to 400 words per minute[3], Aware-ables will 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)[5], which will be further alleviated through post-processing

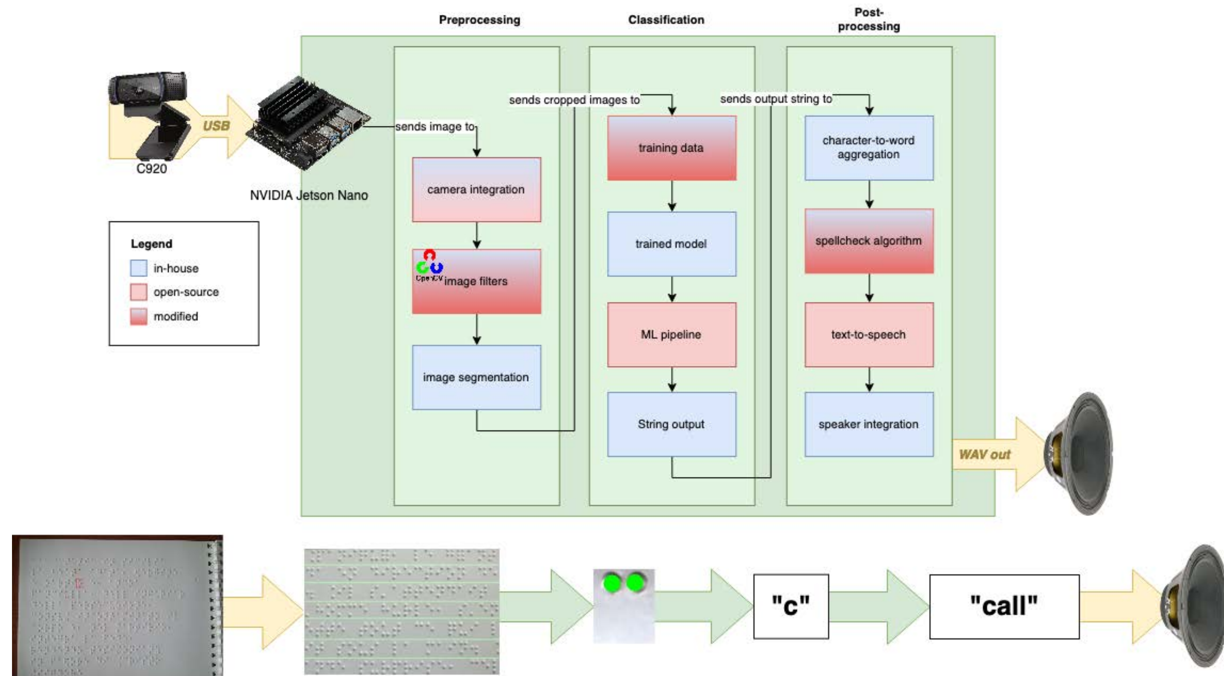

Figure 2: High level block diagram and datapath for our proposed solution.

and spell-check algorithms. Our final target word error rate is less than ten percent, to ensure comfortable and accurate playback for educational purposes.

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

Fig. 4 represents the team's vision for our final prototype. The hardware architecture includes a Logitech C920 USB web camera; an NVIDIA Jetson AGX Xavier, a semi-portable form-factor platform powerful enough to support near real-time capture and inference; a button; and a USB-to-Aux digital audio converter. On triggering the button, a still image captured from the C920 is sent to the 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).

The final solution mounts the C920 camera at a controlled distance away from its document tray, allowing for optimal capture of Braille documents. Furthermore, we control lighting using an LED lighting rig to replicate the lighting conditions of the images used to train the machine learning algorithms in our software stack.

Above, Fig. 2 presents a high-level block diagram for our intended implementation. As previously mentioned, our software stack is split into three sequential 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 were sourced off-the-shelf and which were be developed in-house.

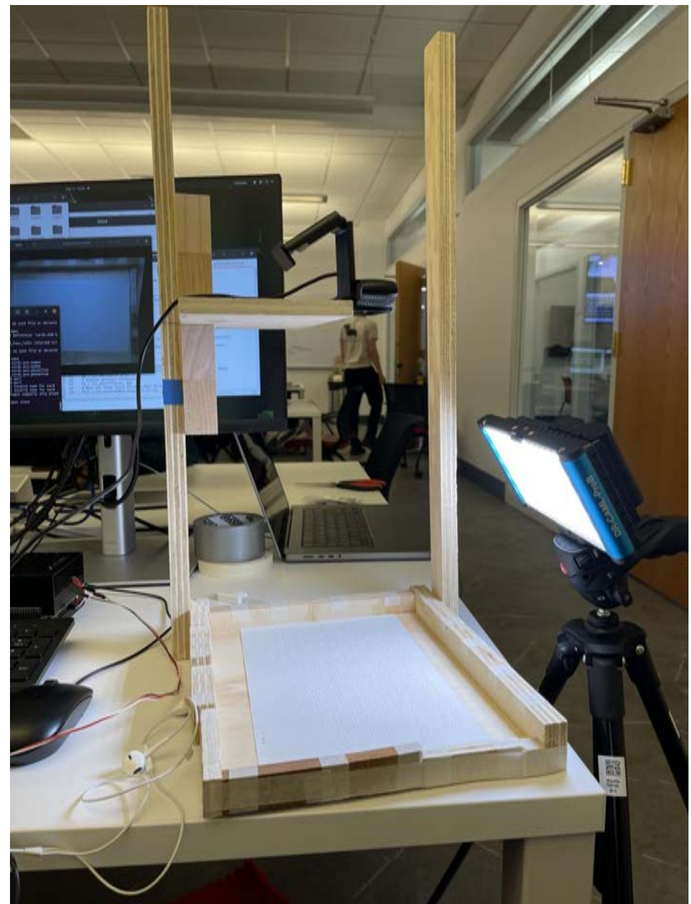

Figure 3: Image of functioning final solution

Below the block diagram, we have provided a high-level visualization of modifications being made to the input im-

age 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) concatentated into an English word, which can then be (5) read out via the speaker.

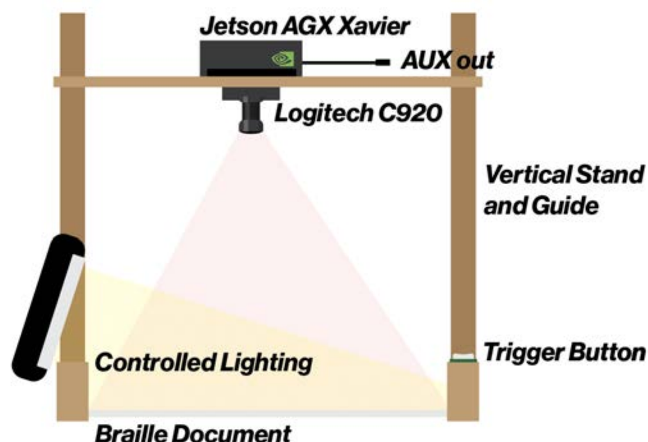

Figure 4: Annotated diagram of final solution

4 DESIGN REQUIREMENTS

4.1 Pre-processing

The first subsystem in our software data path is the preprocessing of captured braille images through computer-vision and segmentation algorithms. Image of printed braille text will be captured using Logitech C920, 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 preprocessed 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 a string 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% [5]. 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 expected classification to be fairly low-latency on our chosen hardware. However, it became clear during testing that classification latency scales linearly with character count and that more powerful hardware improves latency substantially (See section 7.2.2)

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% [6]. 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 [7] 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.

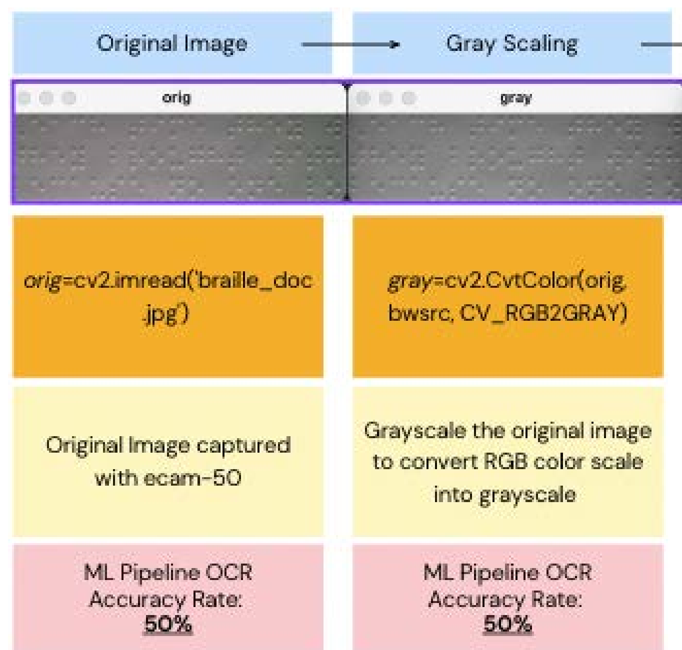

Figure 5: Results of Step1: Capturing of the original image, and Step2: Gray-scaling of the original image

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.

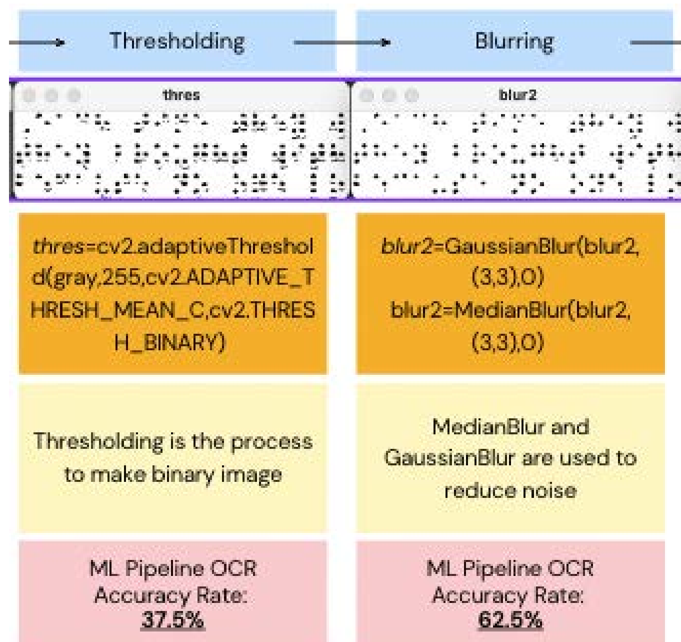

Figure 6: Results of Step3: Thresholding the gray scaled image to create a binary image, and Step4: Application of both median blur and Gaussian blur on binarized image to reduce unnecessary noise

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.

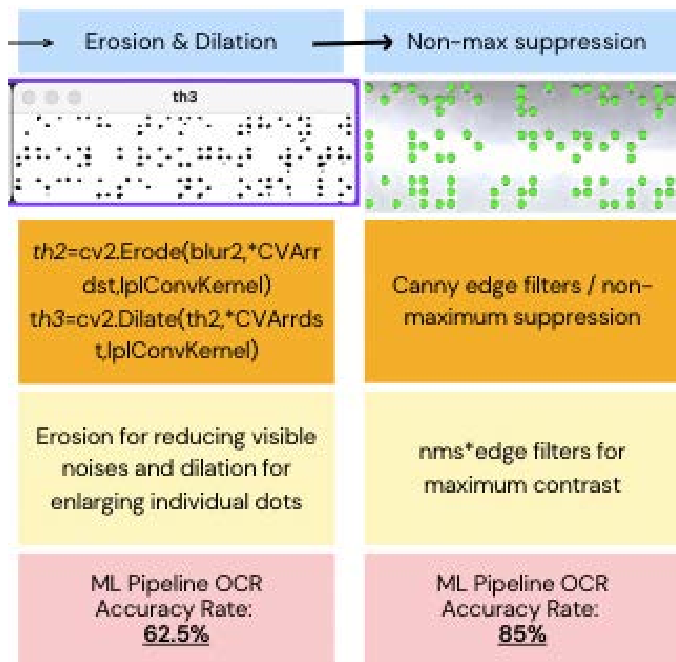

Figure 7: Results of Step5: Erosion and dilation for further reduction of discrepancies by reducing(eroding) visible noises and enlarging(dilating) individual dots, and Step6: Application of further non-max suppression edge filter algorithm for enhanced OCR accuracy rate

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 [8] 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). Using the test method found in Section 7.1.2, 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.

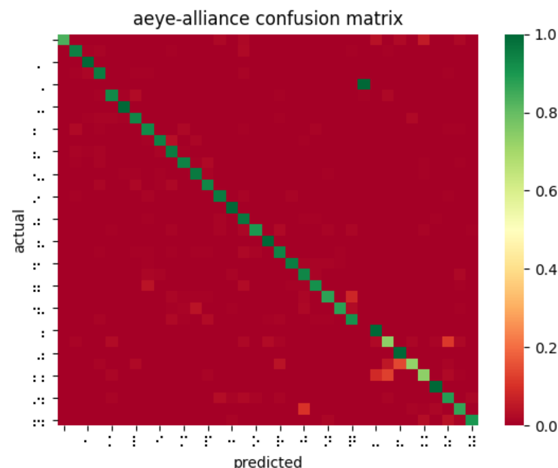

Figure 8: Abbreviated confusion matrix for a pre-trained classification model. Green diagonal line indicates high confidence tested against training dataset.

Since we chose a machine learning path for Classification, it was also important to examine tradeoffs when tuning different parameters of the machine learning model. For example, how does layer depth affect latency? How does learning rate or dataset partitioning affect model performance? Data used to examine this trade-off can be found in Section 6.3. As a result of our experiments, we chose to proceed with an 18-layer ResNet convolutional neural network.

Finally, we also examined the trade-off between the Jetson Nano and Jetson AGX Xavier primarily using Classification (Section 7.3). From our results, it is clear that the AGX Xavier has the edge in both latency and energy efficiency. In our initial experiments, we expected the Jetson Nano to provide an energy advantage due to its smaller thermal envelope. However, we have found that the AGX Xavier's superior inference latency is able to outweigh its more agressive power rating.

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

When choosing a hardware platform for Awareables, meeting the latency requirement was a key determining factor. As a result, while test development began initially on the Jetson Nano, we ultimately chose to deploy our final product on the Jetson AGX Xavier, as originally proposed in our Design Review. To address the compromises of the AGX Xavier platform over the Nano, we installed drivers [9] which allowed us to use a USB WiFi dongle for wireless connectivity, switched from our original MIPI CSI-2 camera to a USB Logitech C920 (which also provided superior clarity) by adapting code from JetsonHacks [10], and used a USB-to-AUX audio converter to output text-to-speech audio. Because our final solution is stationary, weight and power draw became less of a concern, but we were able to compute that energy per classification inference is actually lower on the AGX Xavier when compared to the Nano (see section 7.3).

6.2 Pre-processing & Cropping

The primary deliverable of our pre-processing and cropping subsystem 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 classification and translation. In order to properly 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 steps of preprocessing from original capture to erosion and dilation is already covered in section 5.1) Design Trade Studies of Pre-processing Subsystem.

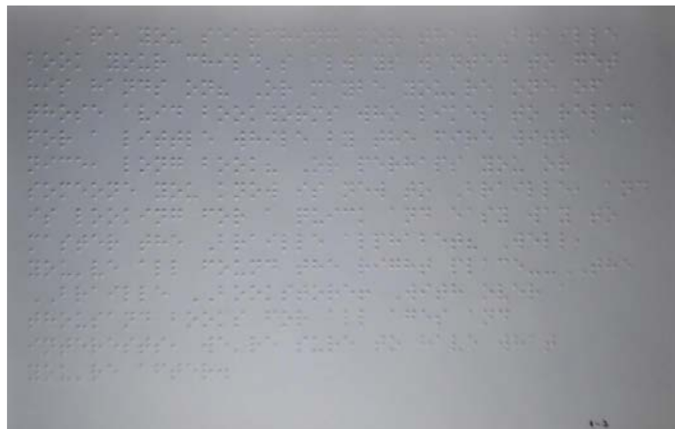

Figure 9: original braille image capture upon botton activation

Figure 10: Image after gray scaling and adaptive threshold

Figure 11: Image after the application of MedianBlur, GaussianBlur filters as well as merging the canny edge detection filter with blurred threshold

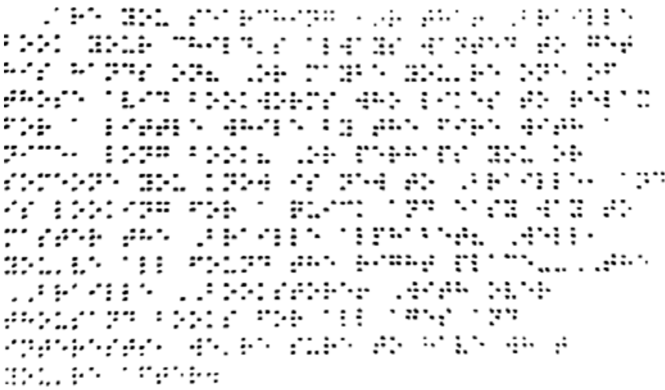

Figure 12: Image after erosion and dilation to get rid of some of the residual noises from canny edge merge

In order to further boost the classification accuracy during our second software subsystem, OpenCV's ConnectedComponentsWithStats function was utilized to extract the top,left coordinates, width, and height of the commonly appearing components, braille dots. With these stats, an image of braille dots can be expressed as a matrix of individual dots comprised of its center coordinates as well as its radius. Using these information, a green circle is drawn on top of the preexisting braille dots as shown in figure 14. Furthermore, along with the usage of non maximum suppression to get rid of some of the redundant dots, the overall pre-processing procedure creates a crisp overlay that resulted in higher classification accuracy.

Figure 13: Filtered stats(left,top,width,height) matrix from the stats value from the OpenCv's ConnectedComponentsWithStats function

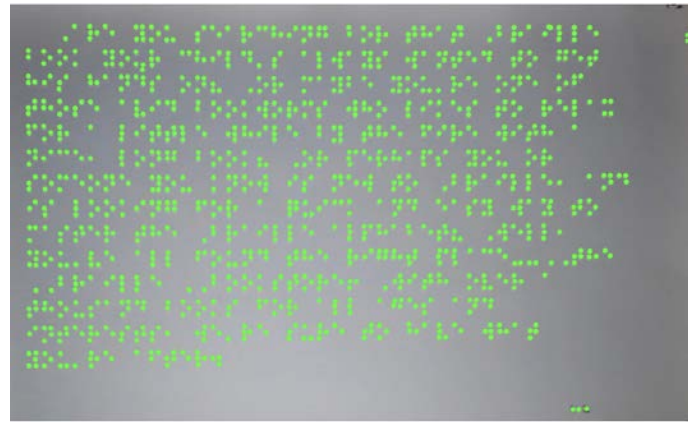

Figure 14: overlay of green circles on top of the original braille dots offers maximum classification accuracy

As far as the cropping is concerned, in order to meet the initial use case requirements of 2 seconds latency from original image capture to translated audio output, the final processed image was manually cropped by image[startHeight:endHeight, startWidth:endWidth]. Compared to the Dynamic ML Crop introduced in section 6.5.1 of this report, the suggested way of cropping requires about 10 20 percent of latency.

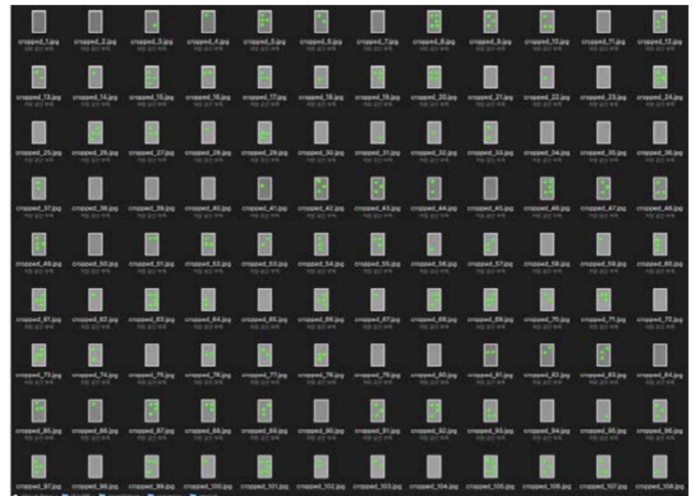

Figure 15: resulting folder containing 560 individually cropped braille dots

6.3 Classification

The character classification subsystem is implemented as an 18-layer ResNet [11] convolutional neural network. ResNet is a popular image classification neural network architecture that uses skip connections to avoid a vanishing/-exploding gradient and enable more reliable training. The architecture was chosen because it was readily available for training at different layer depths on AWS SageMaker. When compared to the alternative option of using a pretrained 3-convolutional block PyTorch model from aeye-alliance (hereon referred to as the Aeye model) [8], this

method yielded better accuracy and comparable inference time.

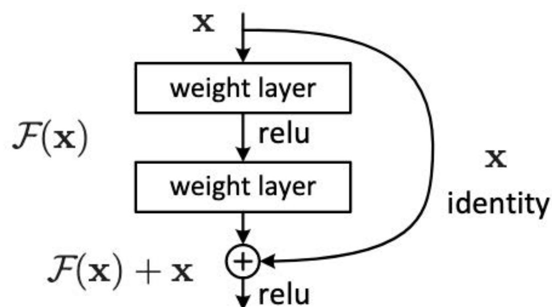

Figure 16: A Resnet building block, visualized (Credit: [11])

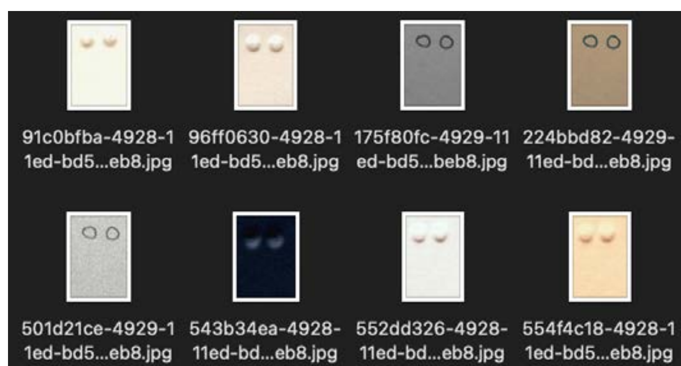

Figure 17: Sample of Aeye-0 dataset

To train the ResNet, we began by writing a script to scrape jpgs off the csv datasets from the aeye-alliance repository to a local directory (hereon referred to as the Aeye-0 dataset). This produced a total of 20,636 images labelled with 37 classes. One caveat to the Aeye-0 dataset is that it is labelled in English translations of each braille character, so we used another script to relabel each class as their unicode braille characters for future extensibility in other languages. We then run this dataset through a version of the pre-processing filters to create the Aeye-filtered dataset. Additionally, the Aeye model is trained on filtered variations of a curated subset of the Aeye-0 dataset that only includes embossed braille, not written braille (referred to as the Aeye-1 dataset). Aeye-1 has a total 26,724 images. It is important to note that SageMaker does not dynamically resize training dataset images to the expected input layer size, but instead center crops images that are larger than the expected input. This nuance corrupted a number of earlier experiments' accuracy data, but still provided good information on the relationship between layer depth and latency. To train correctly on SageMaker, we later resized all image datasets to a uniform 28x28.

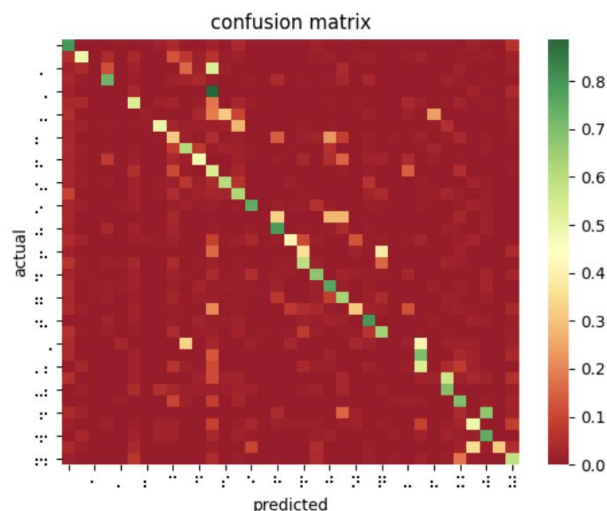

Figure 18: Confusion matrix as a result of center-cropping on AWS (53.50% accuracy)

After a design space search, we decided to implement our model as an 18-layer ResNet trained on 85% and validated on 5% of the Aeye-filtered dataset. 18 layers is the shallowest ResNet architecture available on AWS and provided the best inference latency with little compromise to accuracy. This is likely because braille is a fairly predictable dataset which should not need as many layers for extracting minute features. Using Aeye-filtered yielded the best accuracy on the Aeye-filtered dataset, but performed poorly on Aeye-0. As a result, we then performed incremental training on 70% of the Aeye-0 dataset, keeping 5% of the Aeye-filtered dataset as a validation set to maintain effectiveness on filtered images. This produced a model with a good balance of accuracy and speed on both raw and filtered images, with a combined accuracy of 93.38%.

Image classification models on SageMaker are trained using the Apache MXNet framework, which received limited support on the Jetson Nano platform we were primarily testing on during development. Since no prebuilt package image/wheel was available on the Python Pip package manager, we attempted to build MXNet directly on the Nano, which failed to complete after 24 hours. As a result, we used a separate host system to convert the MXNet param/symbol files to ONNX, an Open Neural Network Exchange framework whose runtime/deployment process on the Jetson platform was more thoroughly documented.

Upon receiving a directory of cropped characters from the pre-processing subsystem, the classification subsystem runs each image through the ResNet-18 model using onnxruntime deployed on the Jetson AGX Xavier's accelerated TensorRT platform. The TensorRT platform uses parallel Tensor Cores specialized in matrix multiplication to perform high performance computing in machine learning contexts. The inferences produced are concatenated and translated using LibLouis [12], an open-source library for translating braille to a number of supported languages; for testing purposes, we have hardcoded this parameter to

Grade I English.

Finally, the concatenated and translated output is returned to the next subsystem, along with a confidence matrix containing any characters with Top-1 prediction confidences under 0.9 (less than 90% confidence that a given image belongs to the predicted class) and an associated list of the next five most probable predictions for each.

6.4 Spell Check & Text-to-Speech

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 2 main components. The first step involves the sequential checking of words after initial concatenation for basic error correction. Lastly, the text will be converted to speech via the Google text-to-speech API.

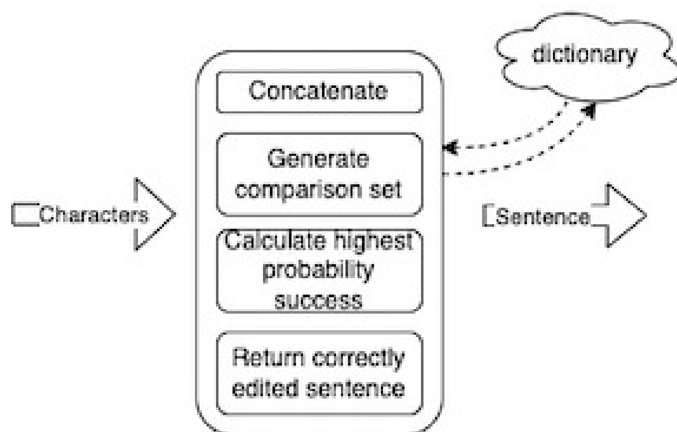

Figure 19: Subsystem C error checking flow chart.

In order to create a working spellcheck model, several key factors have to be considered. The first indication is whether or not a word is in need of error checking after pre-processing and classification. As stated in the classification deliverable, a confidence matrix is provided to the post-processing section in the form of a dictionary. This dictionary provides the indices of characters below the 90% confidence threshold we determined, and their 5 character array of possible options.

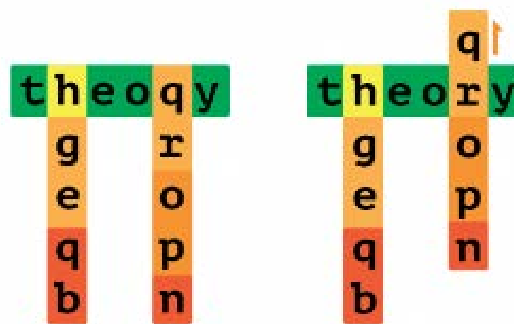

Figure 20: Confidence Matrix being applied to correct characters 'h' and 'q'.

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

The second key factor in development is the "dictionary" checker 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 along with pre-generated volumes of text from classic works to simulate usage probability. The best way to represent the simulated usage probability is through the Bayesian statistical model that defines more data as a means to achieve a stronger statistical average and distribution. By supplementing an English dictionary with large volumes of English text, we can assume word frequency as well as account for edge case possibilities.

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%. This is based on the use case of translation rather than transcription. 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.

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.

6.5 Experimental Features

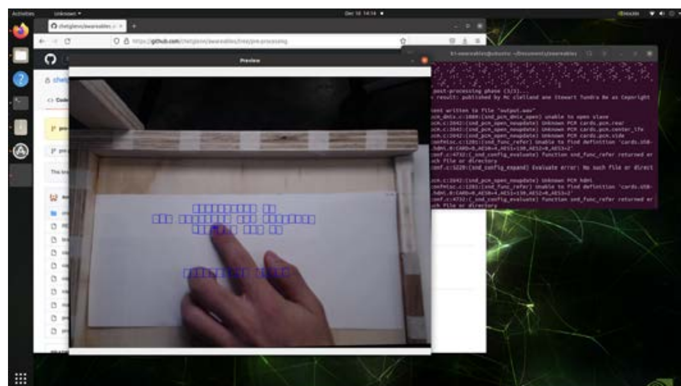

Figure 21: Experimental features (bounding boxes from Dynamic ML Crop and finger detection from Finger Cursor)

For the purposes of public demonstration, a number of features were quickly prototyped using modified off-the-shelf libraries and very little testing. The following is a brief description of each implementation.

6.5.1 Dynamic ML Crop

To implement a more dynamic cropping mechanism that leverages machine learning for detecting braille characters in a scene, we adapted and modified AngelinaReader's inference pipeline [13] to extract the bounding boxes of each braille character. AngelinaReader uses a PyTorch RetinaNet to run object detection on embossed braille characters on a white page. RetinaNet is an object detection model particularly suited to detecting scenes densely populated with objects that the model needs to detect when compared to more lightweight models such as YOLOv5.

We initially attempted to use AngelinaReader's dataset and the DSBI (double-sided braille image dataset) [14] to train our own RetinaNet on a AWS's EC2 P3 machine, but the results were less than satisfactory given the time remaining in our timeline. As a result, we chose to adapt the existing pretrained model, though future implementations could experiment further with fine tuning a custom trained model. One drawback to note about AngelinaReader is that the model is trained on images of braille embossed on white paper lit from above. As such, in order for the model to recognize braille characters, lighting conditions and material must be matched, which is not the case for our classification model. Given more time, a custom model may be trained to be more flexible.

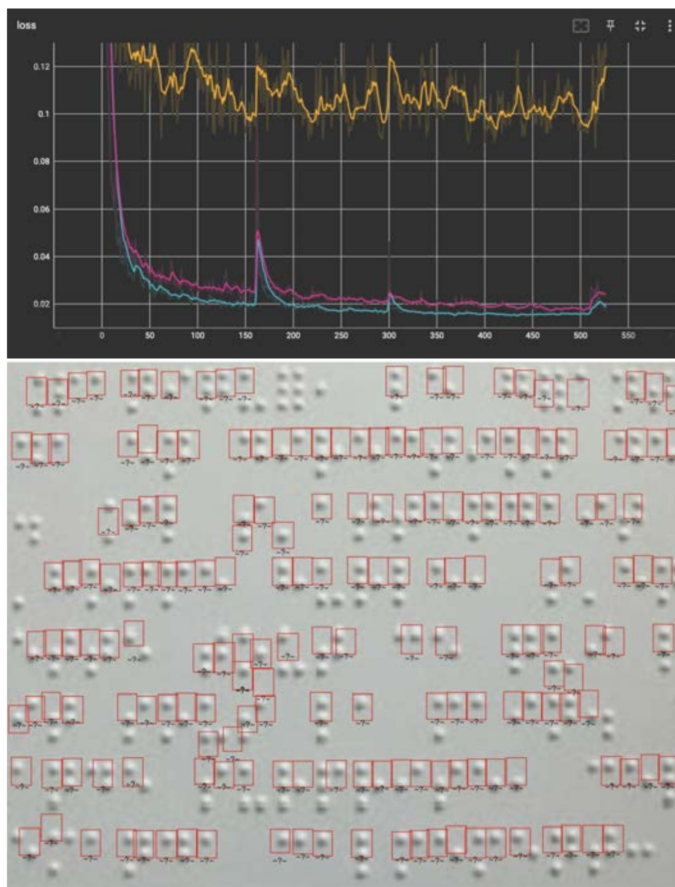

Figure 22: An example of promising training loss curve for one attempt at training a custom RetinaNet on P3, followed by the incorrect bounding boxes of an inference (each bounding box should be associated with a single character)

6.5.2 Cursor Reading

Cursor Reading allows the user to learn braille by running their fingers over the braille and hearing each braille character translated back to them in real time. This feature was devised in response to ethical concerns that were raised that users may become overreliant on the appliance and choose not to learn braille at all. By associating the tactile action of touching the braille character with its translation, it is our hope that users will be able to more effectively learn braille.

Cursor Reading is implemented using Google MediaPipe's hand pose estimation model [15] and the cropped bounding box output of the chosen cropping pipeline. When the top of the user's index finger is within the bounding box of a cropped character for the first time, its translation is read aloud to the user. This feature was implemented quickly for demonstration purposes and therefore was not tested for reliability or accuracy.

Table 1: System testing data when using Dynamic Crop

Document #	Word Count	Average Latency (s)	WER (no Spellcheck)	WER (Spellcheck)
Ad	91	4.4594	1.099%	0.366%
Publishing	9	0.9357	7.407%	7.407%
Ending	54	2.744	2.469%	0.617%
Synopsis	31	1.7920	1.08%	0%
Title Page	14	1.1806	9.523%	7.143%
Overall		**2.222**	**4.315%**	**3.107%**

7 TEST & VALIDATION

Through extensive testing and validation, the overall system, as well as each constituent subsystem was measured for accuracy and latency.

Requirement	Target	Reality
Overall Latency (s)	2	1.736
Words Per Frame	10	98
Character Error Rate	10%	6.63%
Word Error Rate	<10%	3.11%

Figure 23: Table showing use-case requirements met when using static crop (all but latency are met when using dynamic ML crop unless word count is capped at 35 words)

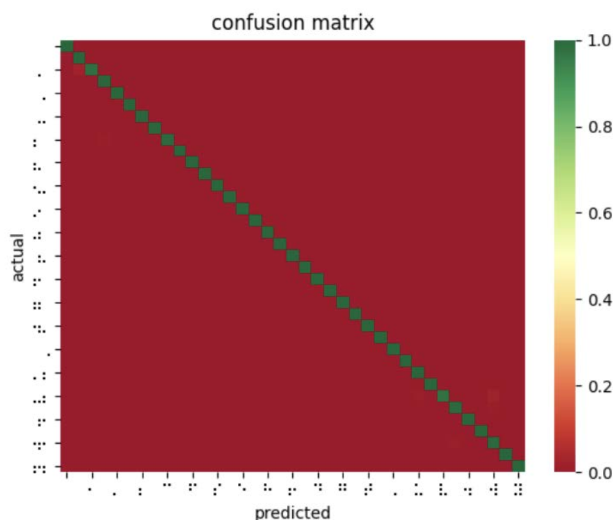

Figure 24: 99.86% accurate confusion matrix of our final model on Aeye-0 dataset.

7.1 Results for Accuracy

7.1.1 Pre-Processing

To examine the accuracy of the pre-processing phase, two factors are being considered; 1) ratio of individual braille dots that have been properly overlaid with green circle through OpenCV's connectedComponentsWithStats and non maximum suppression functions. 2) ratio of proper crops for individual braille characters. For all five braille documents tested(Ad - 91 word count, Publishing - 9 word count, Ending - 54 word count, Synopsis - 31 word count, Title page - 14 word count), the average green circle overlay accuracy exceeds over 95 percent, and as far as cropping is concerned, the accuracy is near 100 percent.

7.1.2 Classification

To examine classification accuracy, we ran each trained model over the Aeye-0 and Aeye-filtered datasets and visualized their performance using a confusion matrix. The final model, trained on 85% of Aeye-filtered and 70% of Aeye-0 was able to achieve an accuracy of 99.86% on Aeye-0 and 86.89% on Aeye-filtered, averaging to 93.375% (6.625% character error-rate).

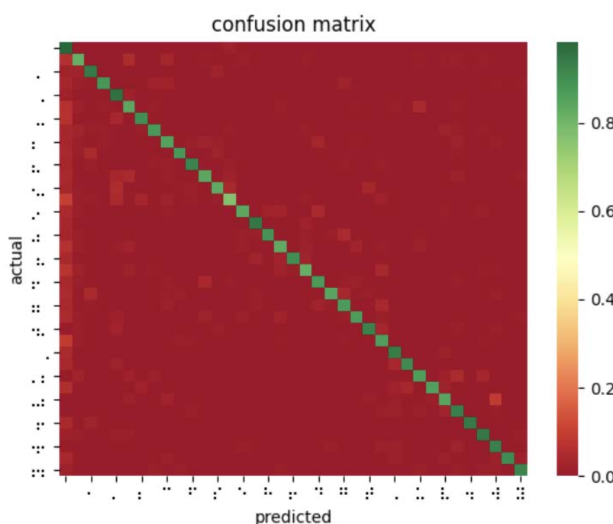

Figure 25: 86.89% accurate confusion matrix of our final model on Aeye-filtered dataset.

Furthermore, because we chose to train our model on 85% of Aeye-filtered and 70% of Aeye-0, it was important to verify that our final model architecture was not overfitting to the dataset it was trained on. To do this, we

performed 4-fold cross validation by training four models, each on 75% of the Aeye-0 dataset, leaving four disjoint training datasets each representing 25% of Aeye-0. Evaluating across these four models, we found that accuracy remained consistently above 99% (99.86, 99.88, 99.84, and 99.78), indicating that the model is learning to recognize important features rather than overfitting to the dataset.

7.1.3 Post-Processing

The post-processing error checking accuracy is based on the algorithms ability to correctly replace errors in English words with single to multiple errors provided per word. Individual metrics are based on sets of 100-150 words with single to multiple character errors. The average results gave a statistical correctness of 97% for the final system implementation.

7.1.4 Overall

Overall accuracy was assessed as word error-rate (WER) using printed single-sided braille pages obtained from the online braille book repository, Braille Bookstore. After translating each page manually, we assessed average word error rate by scanning and cross-referencing the produced string with the actual solution. Overall we had five pages of varying lengths that were assessed for accuracy and latency. Table 1 collates all of the data collected over 3 trials per page (15 trials in total).

From the 15 experiments, we find a WER average of 4.315% without spellcheck, and 3.107% after spellcheck. Where WER maxes out at 9.523% without spellcheck, spellcheck lowers this maximum to 7.143% and reaches an ideal minimum of 0%. We find that while the classification subsystem's raw output fulfills our WER requirement of less than 10%, spellcheck does a good job of lowering the WER further by **1.39x** on average in real-world usage.

For these experiments, we decided to use the dynamic ML crop feature due to its reliability over varying page positions, allowing us to test more quickly in a variety of situations. Static crop requires highly controlled page alignment in the document tray, which may not reflect real-world usage, despite it plausibly being able to yield better latency or accuracy.

7.2 Results for Latency

7.2.1 Pre-Processing

The latency of the pre-processing phase was measured through python's time.time() function. As specified in Table 1 in section 7.1.2 above, the average latency for the model with ML boundary cropping would vary between 2 to 5 seconds whereas the average latency for models with manual cropping parameters was less than 0.5s. Although the ML cropping with green circles overlaid on top of the original images may take slightly longer than the initial use case requirements, given how primary users of our device are legally blind, enhancing the usability of our apparatus through ML boundary cropping outweighs couple seconds of extra latency.

7.2.2 Classification

Classification pipeline latency was assessed by timing the inferences performed during accuracy tests, then dividing by the number of images fed to the model to produce an average latency. Latency was assessed across different layer depths to examine the trade-off between model accuracy and model latency. It is important to note that we were unable to test ResNet-152 on the TensorRT provider due to memory constraints on the Jetson Nano. Classification latency was also used to assess hardware performance differences and the trade-off between power and latency between the Nano and the AGX Xavier.

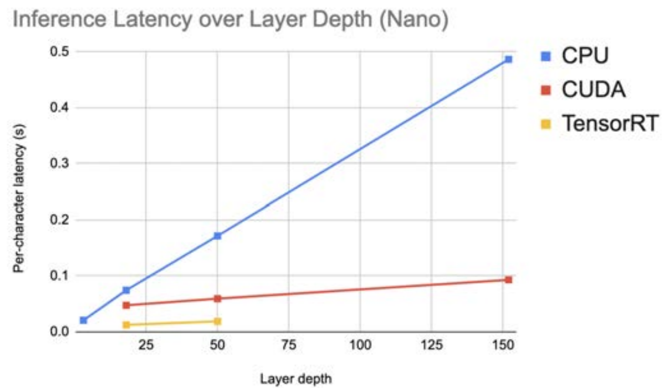

Figure 26: Inference latency on the Nano as layer depth increases.

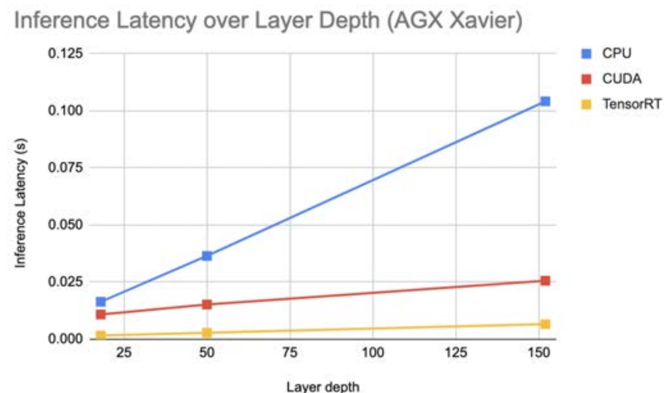

Figure 27: Inference latency on the AGX Xavier as layer depth increases.

From these findings, the relationship between layer depth and inference time is similarly linear on the AGX Xavier, however, the 30 watt operating power provided a substantial boost in baseline latency. As a result, ResNet-18 inferences using TensorRT outperformed the 3-layer Aeye model on the Jetson Nano.

7.2.3 Post-Processing

The main latency trade offs for the post processing sub section were centered around the usage of a confidence matrix versus normal error checking as discussed in system implementation. In order to measure the time it took to correct errors when given a confidence matrix and when not, we tested the algorithm on 3 sets of 100 words with single character errors. As the baseline, this relies on the reliability of the classification, and focuses more on the speed of correction. Overall, the tests showed that with the confidence matrix, the overall error correction sub section took only 0.5 seconds for 100 words as compared to 1.0 seconds without it.

One important note in testing the latency of the confidence matrix individually is that our given threshold is at 90%. This indicates that we assume any character classified with above 90% confidence will be correct. Although this is a slight over assumption, it helps to simplify testing for the individual sub system.

7.2.4 Overall

Overall latency when using dynamic ML crop was assessed in parallel with WER (Table 1). Latency here is defined as the time spent between capturing the initial image and sending the final string to Google's text-to-speech API. Using this configuration and metric, latency varied greatly between a low of 0.8897s and a high of 5.4812s, achieving an average of 2.222s. When compared to latency measured over 6 trials using static crop, static crop took the lead in both variance and overall performance. When using static crop, the average latency of 1.736s falls under the established design requirement of 2s of latency, with the added detail that latency never goes above the 2s limit.

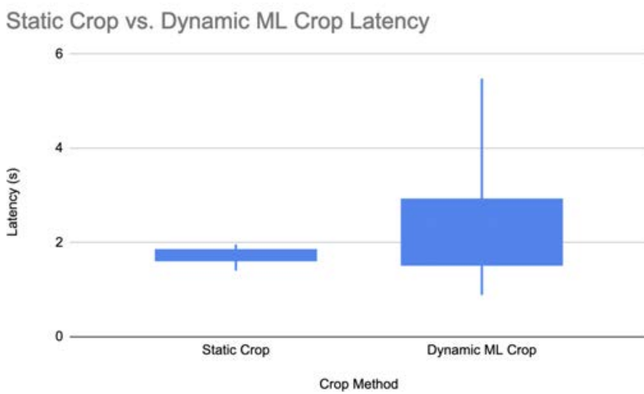

Figure 28: Comparing overall latency between dynamic ML crop and static crop.

Comparing the two crop latencies relative to the number of words processed, static crop performs 560 crops, which, when divided by the average character count for a given English word (4.7 characters + 1 space) yields 98 words. Meanwhile, a 91 word document required an average of 4.45s when using the dynamic ML crop. Here, we see a clear trade-off where dynamic crop provides more flexibility at the cost of meeting our latency requirement.

7.3 Results for Energy Efficiency

While not initially a design requirement, we combined datasheet information and average classification latency across all models tested to roughly assess the energy efficiency of the Jetson Nano and AGX Xavier.

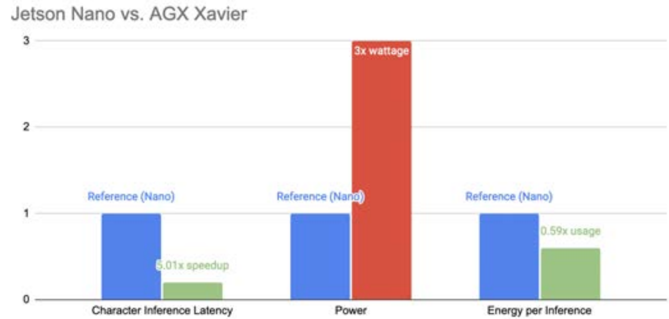

Figure 29: Comparing latency, power, and energy-per-inference between Jetson Nano and AGX Xavier

Based on the average energy per inference of the AGX Xavier compared to the Jetson Nano, we were able to justify switching platforms for our final implementation.

8 PROJECT MANAGEMENT

8.1 Schedule

Our development cycle was split into three phases. Phase one focused on the research and development of concept/requirements. This was the initial proposal period when pitching the overall use-case and idea for our product. This phase spanned two weeks, and transitioned into phase two, design. The design phase was focused on coloring in the specifications of our solution with more detail for both our software and hardware components. This phase lasted a total of 3 weeks, culminating in the design review and report. The most significant portion of our project was spent on the development of the solution. Through parallel workflows, each individual developed 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. 31.

8.2 Team Member Responsibilities

There are three subsystems involved in the full product's main processing, and the overarching hardware development was initially assigned to all three members. For

the first subsystem, Jong Woo Ha was 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 was then in charge of classifying individual images into characters using a ML pipeline. Lastly, Chester Glenn was responsible for the text-to-speech integration, which involves error-checking words and a smooth text-to-speech application.

During development, each member was involved in testing their own subsystems. Kevin took charge of the majority of hardware bring-up and integration, with Chester helping with audio integration. Jong Woo was responsible for designing and prototyping the wooden rig and measuring the best height and lighting conditions for capture. Overall testing was completed as a team, implemented and collated by Kevin. In order to add more interactivity to the public demo, Kevin also prototyped the Experimental Features (Section 6.5).

8.3 Bill of Materials and Budget

The estimated Bill of Materials and overall cost of the project are included in Table 2.

8.4 AWS Usage

As part of our budget, Awareables was assigned $100 AWS credits for use. Using these credits, AWS SageMaker, S3, and EC2 were employed to train and host CNN models and datasets. Specifically, we used AWS SageMaker to train a custom MXNet Image Classification model (ResNet) for classifying braille characters. Using the utilities available through SageMaker and CloudWatch, we were able to fine-tune and track parameters and test different layer depths to help us choose the optimal model configuration for our needs.

In addition to the MXNet Model, we also used P3 accelerated computing machines on EC2 to experiment with training a custom PyTorch RetinaNet for detecting and labeling braille characters from a page. Using EC2's built-in PyTorch 13.0 AMI configuration meant that setting up our environment was easy and we could focus on training our model.

For both of these applications, S3 buckets were our central repository for training data and model checkpoints. Configuring S3 for access from different systems was well documented, making it easy to batch upload 20,000+ image datasets without hassle.

8.5 Risk Management

Project risks encountered were handled on a case-by-case basis. Because each team member was responsible for one subsystem, we were able to handle risks independently if they only affected our subsystem. However, communication was key in maintaining information parity between subsystem leaders and ensuring no subsystem fell behind in development.

One key risk identified early on in the project's design was that despite our enthusiasm, the team overall lacked experience in computer vision, machine learning, and natural language processing – topics key to implementing our final solution. To diffuse this risk, we split the task into subsystems and assigned a "designated expert" for each one. In doing so, each member was only responsible for learning a sub-section of the more complex task, increasing efficiency and reducing risk of overburdening or confusing task-switching between novel topics.

One risk identified during development was the difficulty of hardware bring-up. Originally, we had planned to develop on the AGX Xavier, giving us ample performance overhead. However, we had trouble using SDK Manager to flash the AGX Xavier for use. As a result, we quickly swapped to the Jetson Nano. Kevin was responsible for bringing up the Jetson Nano, followed by the Jetson AGX Xavier. Upon encountering driver installation issues with the Jetson Nano, this was quickly communicated to the team via group chat. In response, the team formulated alternate strategies and decided to swap from the CUNX eCAM-50 to the Logitech C920.

Overall, the team did a good job of managing risks encountered throughout the development process. Any blockers were identified quickly and communicated with the team, who in response was eager to help formulate alternate strategies.

9 ETHICAL ISSUES

From the initial debate on the ethics of our product, we had several main concerns. As the original scope of our product was a wearable device for users, our concerns focused on privacy, security, information integrity, safety, and overreliance. In comparison to the wearable device, several of our concerns were mitigated due to the altered use case, but we still want to guarantee each ethical component meets a reasonable standard or has been considered in the final design.

Any system that relies on a camera will find that it is essential to guarantee the privacy and security of a user's personal life and interactions. Whether or not the device is plugged in, it should not be storing information on the user, or recording without consent. Initially, the wearable camera device provided a much stronger case against user privacy, as it relied on the user to wear the camera and interact within their daily lives. This provides a threat to personal information being stored or used by a malicious adversary. In order to prevent this, our stationary build guarantees a fixed camera that only takes in the frame of braille/documents provided. In addition, no external image storage or additional processing beyond the camera capture for pre-processing is necessary due to the already pre-trained classification model.

In addition to privacy and security, several concerns we

Table 2: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Jetson AGX Xavier	F21072	NVIDIA	1	$0.00	$0.00
USB Webcam	C920	Logitech	1	$0.00	$0.00
Tactile Switch SPST-NO Top Actuated	1241.1055.8000	SCHURTER Inc.	1	$4.50	$4.50
Prototyping Board	N/A	Treedix	1	$0.00	$0.00
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
LED Film Light	CamLux Pro	Dracast	1	$0.00	$0.00
Tripod	49-inch Tripod	Manfrotto	1	$0.00	$0.00
Braille Book	Snow Goose	Blind in Mind	1	$12.95	$12.95
Wood apparatus	N/A	Self-made	1	$0.00	$0.00
AWS Credits	N/A	Amazon Web Services	1	$100.00	$100.00
					$160.19

had stemmed from the importance of integrity around data and information. As our use case states, we aim to provide braille literacy and to help aid the community to improve recognition of the language. In order to help educate the user and public, it is important that data integrity is a main concern of ours going forward. With the addition of both character classification confidence and word error checking in the post-processing sub section, we aim to minimize the possibility of misleading information or misidentified information that could lead to improper use or falsification of data.

Continuing with the importance of our use case in helping to educate the public in braille literacy, another concern of ours was an increase in over-reliance. Given the accessibility and usability of our product as an overall text-to-speech translation, it could lead users to rely heavily on the processing of material rather than actually learning braille. In order to help minimize this likelihood, we introduced a feature that allowed individuals to identify character-by-character on the page and have it read out loud to them. This greatly increases the likelihood of character recognition by helping the user to directly correlate each braille character with a distinct letter/symbol.

Lastly, with any hardware, there are safety concerns. Especially with our initial use case centered around the wearability of our product, there were concerns that misuse or lack of precise user accessibility could result in significant costs to the product and user. After rescoping to the stationary build, this problem was greatly mitigated to a more manageable framework. Not only does this allow for us to focus on the design and control more specifically how an individual uses our product, it minimizes human contact with the more valuable pieces of the project like the Jetson Xavier or the camera mount.

10 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 Braille-Master, Duxbury Braille Translator, GOODFEEL Braille Music Translator, or Toccata [16]. 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.

11 SUMMARY

Overall, we consider our product to have satisfied the design requirements we set forth at the beginning of the semester as well as through to the design review/report. One of the main concerns that we had was with the overall latency of the full pipeline processing times, especially when considering the wearable device. With a 2s latency from image capture to full post-processing spell check, we knew that this would be the primary bottleneck that could also affect the reliability of our error checking as well. In order to minimize the timing constraints of the final product, the Jetson AGX Xavier provided substantially improved performance in comparison to the Jetson Nano, helping us to reach our goal within a small margin of error. Due to the rescope to the stationary frame, the Jetson Xavier is no longer a concern for wearability.

11.1 Future work

Although we feel that our product could be a very useful tool for educational purposes and for the visually impaired community, we have not yet decided whether or not we would like to continue working on it going forward. Consid-

ering that we will all be graduating in the coming months, our main priority is on the new jobs we are coming in to and our future careers in the industry.

One thing that was brought to our attention during the final demo is that this product may be patent worthy as software or even sponsored by the university/related disability services. Going forward, we still feel like we are focusing on our own careers beyond the product, and the work it would need to be truly marketable is not within a reasonable time frame for us.

11.2 Lessons Learned

To conclude, the semester has led us on a tumultuous journey of learning about openCV, Machine Learning, Embedded Systems hardware, and spell-checking challenges. We had a lot of difficulties in the beginning of the semester integrating with the hardware we wanted, and in general it was never easy to test/use the Jetson products perfectly. We highly recommend any future groups to work diligently together to set up their hardware before hand so that all team members can test on the hardware independently, and try not to work in a way that relies to heavily on prior segments. By working in parallel most of the semester, we were able to each polish our individual sub sections and connect them together for the final pipeline.

Glossary of Acronyms

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

- AWS - Amazon Web Services
- BOM - Bill of Materials
- CER - Character Error Rate
- CNN – Convolutional Neural Network
- OCR – Optical Character Recognition
- OBR – Optical Braille Recognition
- WER - Word Error Rate

References

[1] 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.

[2] 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/.

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

[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.

[5] 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.

[6] 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.

[7] 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.

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

[9] brektrou. *Realtek RTL8811CU/RTL8821CU USB wifi adapter driver*. URL: https://github.com/brektrou/rtl8821CU.

[10] JetsonHacks. *USB-Camera*. URL: https://github.com/jetsonhacks/USB-Camera.

[11] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (2015). DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[12] Liblouis. *Liblouis - An open-source braille translator and back-translator*. URL: http://liblouis.org/.

[13] Ilya Ovodov. *Angelina Braille Reader*. URL: https://github.com/IlyaOvodov/AngelinaReader.

[14] Renqiang Li et al. "DSBI: Double-Sided Braille Image Dataset and Algorithm Evaluation for Braille Dots Detection". In: (2018). DOI: 10.48550/ARXIV.1811.10893. URL: https://arxiv.org/abs/1811.10893.

[15] Google MediaPipe. *Hands — mediapipe*. URL: https://google.github.io/mediapipe/solutions/hands.html.

[16] 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.

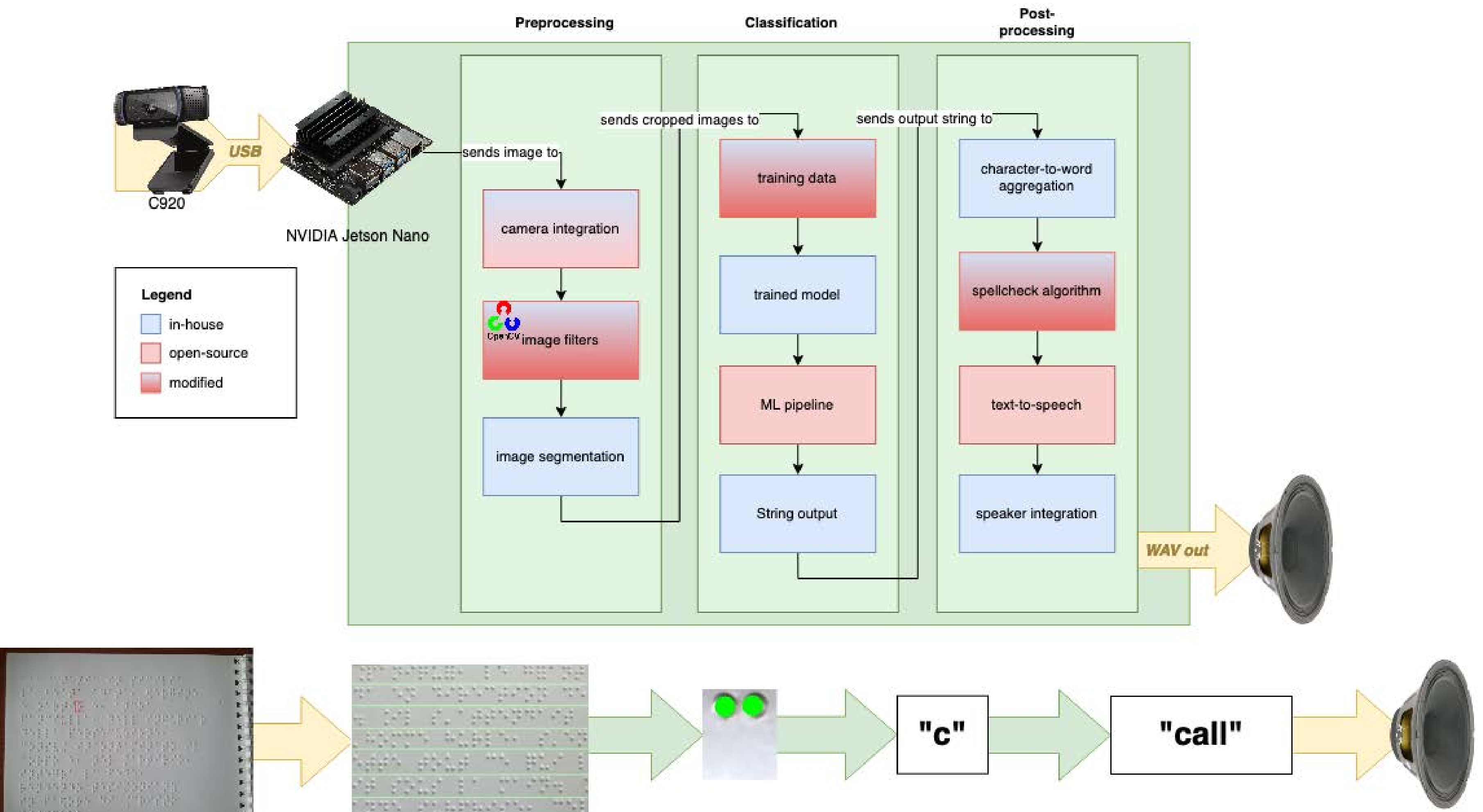

Figure 30: A full-page version of the same system block diagram as depicted earlier.

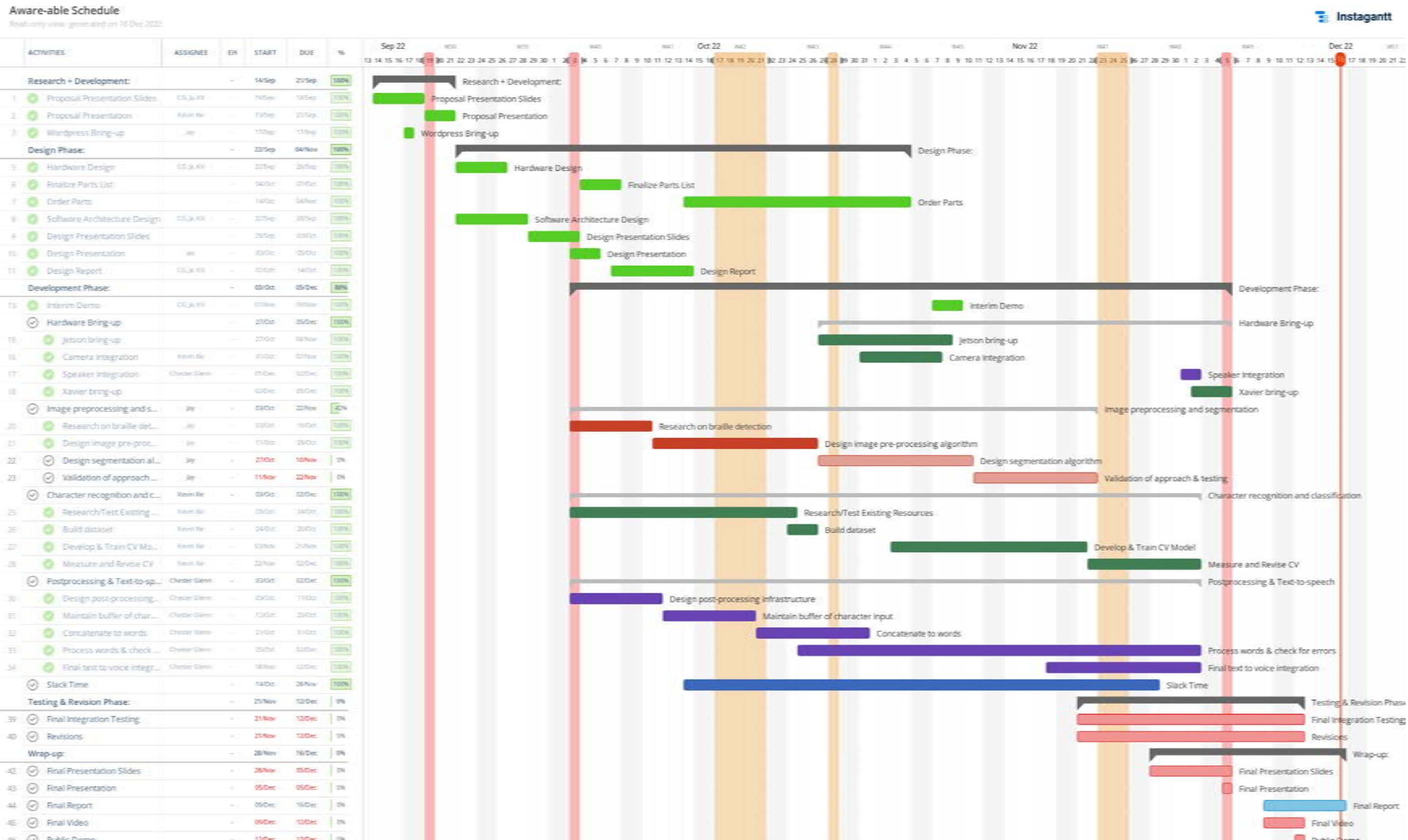

Figure 31: Gantt Chart