SceneScribe

Authors: Aditi Narasimhan, Nithya Sampath, Jaspreet Singh
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—We present a system capable of improving lecture experiences for visually-impaired students by narrating text and graph data displayed on instructors' presentation slides. Current solutions for this task include apps where students can upload a photo of the slide and receive spoken output of extracted text, which is inconvenient and time-consuming. Our solution is a camera attachment to glasses which will capture an image of a slide, match it to a pre-uploaded lecture slide using a combined detection-classification approach, extract text and graph descriptions with PDFReader and a CNN-LSTM, and read them aloud to the user through an iOS app 10% faster than current solutions.**

***Index Terms*—Accessibility, CNN-LSTM, Graph Description, iOS App, Machine Learning, Siamese Network, Object Detection, Classification, Text Extraction, Text-to-Speech, Voice Over, YOLO**

1 INTRODUCTION

1.1 Motivation and Application

During lectures, instructors do not always completely explain all of the text and graph data on the slides they are presenting. While this may not seem like an issue for sighted students, who can easily glance up at the screen and quickly read the text, it causes a substantial information disparity between sighted students and blind/visually impaired students. We classify the two possible kinds of information disparity in the following ways:

1. The slide contains *necessary* information: information on the slides is necessary for the visually impaired student to understand the lecture material.

2. The slide contains *supplemental* information: information on the slides is not necessary for the understanding of the lecture, but can be used as reference.

Information disparities provide an inherent imbalance between students in the class - in this case, between visually impaired and sighted students. This can lead to visually impaired students understanding the material less and performing worse on exams when compared to their sighted peers, which is unfair and may hinder visually impaired students from performing at their highest standards.

Therefore, our solution is to provide the user with a system that provides them with a narration of the displayed slide at their request. During lecture, they can indicate to our system that they want to know what is on the screen (through a mechanism such as a button press) and our device will narrate the projected information to them.

Our use case expects students to use the device in class because any necessary or supplemental information will be most valuable *during lecture*: the extra information can help students better understand what the instructor is talking about during the lecture, and can ask questions to help clear up misunderstandings that the instructor might build on later in the lecture. This way, the user will likely be able to understand the entire lecture during class, and will not be forced to go back and review it after class time with our device.

Because our system will be used during class, while the professor is speaking, we expect concerns about the usefulness of the product. However, we consider multiple factors to alleviate these concerns:

1. There will likely be pauses during the lecture during which the visually-impaired student can use our device. For example, instructors usually pause for sighted students to copy down slide information in their notes.

2. Our system allows the user to stop the audio if the instructor starts speaking.

3. Visual impairments have been linked to enhanced auditory perception, meaning blind users generally have a much easier time distinguishing and processing multiple audio streams when compared with sighted users [1].

4. When the slide contains *necessary information* (as mentioned earlier), knowing the content on the slide is more valuable than catching every sentence that the instructor speaks, so missing a few sentences in favor of a slide audio description is justified.

1.2 Goals

Our system's main goal is to correct information disparities, which helps provide equal opportunities between sighted and visually impaired students. We also sought to make the system as inexpensive as possible (while maintaining its accuracy) in order to level the playing field between more affluent students and students who cannot afford higher cost options. Since we are limiting our scope to three types of graphs, our product is not a replacement for instructor attention to accessibility needs; instructors

should still design their lectures so they are conducive to all students' learning.

2 USE-CASE REQUIREMENTS

2.1 Latency Requirements

1. The latency between pressing the start button and receiving an audio description of the slide should be at most 8 seconds. Our device should be faster than other options, such as taking an image with a smartphone and analyzing it with an app. After experimentation, we found that the average time to do this was 8 seconds, so our device should take less time than this.

2. The latency between clicking the button to upload a lecture from Canvas and the time that it takes to process it and extract the descriptions of slides must be less than 10 minutes because passing periods are about that long - our device should be able to process presentations between the end of a previous class and the start of the next class. We want the descriptions to be ready before the student goes to their next lecture.

3. The latency between pressing the stop button and hearing the audio stop should be at most 140 ms. The purpose of the stop button is to immediately halt any sound that is playing from our device, and we found that for humans, a latency of less than 140 ms for a change in sound would be perceived as instantaneous [2]. The text-to-speech should also be cut off before the start of the next word.

2.2 Weight Requirements

1. The weight of the attachment on glasses should be at most 60 grams. We want the attachment to be lightweight and convenient, since too much weight would cause an excess of pressure on the user's ears and nose.

2.3 Power Requirements

1. The battery life of the device should be at least 6.64 hours. Our device is intended to be used in classroom settings, and should be able to last throughout the day for convenience. Our system should be able to last as long as the average number of teaching hours in a day, which in the United States is 6.64 hours [3].

2. The app should consume an appropriate amount of power on the mobile device: at most 25% of the phone battery when used for 6 hours. Since this is an assistive device, the power usage of our smartphone app should not detract from our user's daily smartphone usage. We found that on average, most people have about 25% of their phone battery remaining at the end of the day [4], so we will limit our power usage accordingly.

2.4 Accuracy Requirements

1. 95% of well-formatted, standard font words that are spelled correctly must be accurately identified. We need to ensure that our text detection is accurate, as we do not want to misinform or confuse our user. With a lower accuracy, our device could actually detract from the user's learning experience.

2. The device must be able to identify the existence of graphs on the slides with about 95% accuracy, and must identify the type of graph (line or scatterplot). Similarly to before, we must avoid detracting from the user's learning experience. Therefore, it is important to correctly identify all graphs on the slide.

3. Graph trends (increasing, decreasing, or constant) and shape (linear or nonlinear) should be accurately identified 90% of the time. As of now, there is nothing on the market that parses graphs to natural language specifically for visually impaired students. Even if it does not work 100% of the time, it will still be helpful for blind students who do not have an alternative device.

2.5 Usability Requirements

1. The stop and start buttons for the device should be easily distinguishable. Because the device should be relatively straightforward to use out of the box, the time it takes for a blind user to distinguish between the buttons when they first get the device should be at max 0.5 seconds. This is because the average early or congenitally blind person can read up to 120 braille words per minute [5], which corresponds to 1 word per 0.5 seconds. In our system, each button will have "start" and "stop" printed on it in braille, so the length of time it takes a user to distinguish between the buttons should be about the length of time it takes them to read a short word.

2. All elements on the app should be compatible with the VoiceOver accessibility mode so that every button and header can be narrated to the visually impaired user.

3. The speed of the Text-to-Speech (TTS) should be understandable. We will provide multiple options for the speed of text to speech, and the user will be able to select it using our app. The default speed will be 150 words per minute, and we will allow users to select from a range between 75 words per minute (0.5x speed) and 300 words per minute (2x speed). This is because the average speaking speed of an English speaker is 150 words per minute [6], but we want to provide our user with options depending on what speed they desire.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

3.1 Block Diagram

See Fig. 15 in the Appendix for a complete block diagram of our system architecture. We discuss how the system will operate below, from both a user perspective and system perspective. Additionally, we provide an annotated photo of the (visible, hardware) components of our system here. Inside the case, we have connected our camera, battery, and buttons to our Raspberry Pi Zero. From the outside of the case, the user can charge the device, switch it on, interact with the buttons, and clip the case onto their glasses.

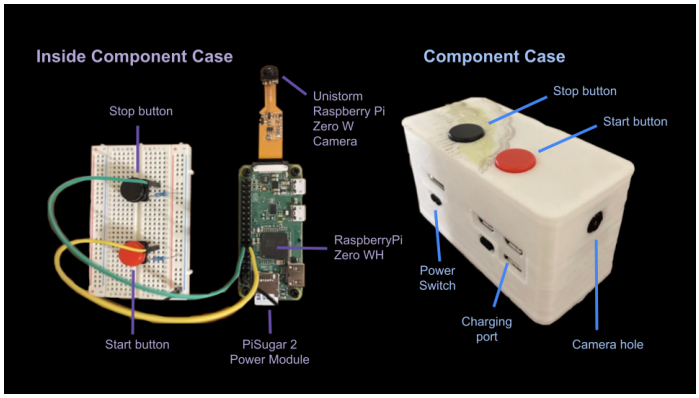

Figure 1: Annotated Photo of Device

3.2 Principle of Operation

3.2.1 User Perspective

When the user receives the system, these are the steps they will take to set it up:

1. Download the application.
2. Upon install, the application will prompt the user to input the names of their classes and the Canvas code associated with each class.
3. The professors of each class will also need to provide an API token which the user will input for each class as well, in order to meet Canvas's privacy guidelines.

Now, the app will be configured properly, with a display of buttons corresponding with each class that the student is taking. At least ten minutes before each lecture, the user will click on a button that corresponds to their next class.

At the start of lecture, the user should:

1. Attach the device to their glasses.
2. Face the projected slideshow.
3. Open the app, and pair their headphones with their phone.

During lecture, when the user wants to know what the slide says, they should press the "START" button and listen for the audio. If the user wants to cut off the audio, they should press the "STOP" button.

3.2.2 Developer Perspective

- **Before Lecture:** The iOS app will store a list of classes that the user is taking. Before going to their lecture, the user will indicate which class they are going to through the app interface. After this selection, the most recent lecture PDF from the corresponding Canvas course will be scraped and uploaded to our server hosted on a Jetson. The Jetson will be on campus at a central location (such as the disability resources office), and our app will communicate with it wirelessly.

  Then, each slide of the PDF will be analyzed for the existence of text and graphs; in particular, we will run a modified YOLO-v5 model to get bounding boxes around the particular types of graphs we are considering (line graphs and scatterplots).

  The text will be extracted with a PDF reader (PyMuPDF), and the graph descriptions will be extracted using our trained CNN-LSTM model. Then, all of the associated text for each slide will be stored in a JSON so that it can be easily accessed during lecture.

- **During Lecture:** Whenever the user wants to hear a slide description during lecture, the following steps will occur: The user will press the start button on the side of the glasses attachment, which will indicate to the Raspberry Pi that it should capture an image. The image will then be wirelessly sent over WiFi to a server hosted on the Jetson.

  Once received, the slide number (which is contained in a red box in the bottom right corner of the image) will be detected and cropped, the cropped image will be pre-processed (deskewed, warped, grayscaled) before being passed to the classification algorithm, which detects contours to find each digit and then runs a digit-classification model (standard CNN) to classify the digit; these digits are then strung together to get the final slide number. Once the correct slide is identified, we can fetch the stored text and graph description for that slide (which was already extracted and stored in a JSON) and send it to our iOS app, where it is read aloud to the user using text-to-speech. While this text description is being played, the user can press the stop button at any time when they don't want to hear the audio anymore.

3.3 Changes and Final Solution

Since our design report, our product went through multiple iterations. The major change involves the switch from a Siamese NN to object detection in order to perform slide matching. Now, the professor is required to add bolded slide numbers to the bottom right of each slide with a red bounding box around each one; for the professor's convenience, we provide a Python script which can take in a PDF/PowerPoint, convert each slide to an image, add the slide number in the proper format, and convert the resulting images back to a PDF/PowerPoint.

This switch will allow us to perform object detection with YOLO-v5, followed by digit classification, in order to extract the slide number, and match it with a pre-extracted PDF of slides. We also modified our initial requirements of identifying four graphs (line, scatterplot, bar, and pie) to just two graphs (line and scatterplot), so that we could focus our data collection and subsequent accuracy around our limited scope; additionally, we felt that this rescoping was justified by the fact that the vast majority of graphs we found in lecture slides during the data collection process were line graphs and scatterplots.

Additionally, a minor change is that the PDF Reader we selected works in a way such that we no longer have to perform the pre-processing step of whiting out the graph on the slide.

4 DESIGN REQUIREMENTS

4.1 Latency Requirements

4.1.1 Before-Class Latency

As we mentioned in the previous section, certain steps which we included in our design review report are no longer necessary/have been replaced by other methods. Below were our original design requirements for before-class latency.

The total before-class latency can be summarized by the following equation, which is explained in more detail below. We calculate the total latency l in terms of the following parameters:

- l_s, the latency of scraping the PDF from Canvas.
- l_{bb}, the latency of identifying the bounding boxes around the graphs.
- l_w, the latency of processing the graphs by whiting out the boxes.
- l_e, the latency of computing the slide embeddings.
- l_t, the latency of extracting the text from the slides.

$$l = l_s + l_{bb} + l_w + l_e + l_t \tag{1}$$

Note that the white-out and slide-embedding computation steps are no longer necessary, because we have changed our model to detect bounding boxes around graphs (rather than whiting out the text) and extracting a description directly. We also note that the total latency given above should have included the following latency requirement:

- l_g, the latency of generating the graph descriptions from the extracted graphs.

The modified equation, were we to recreate our design requirements while planning for the current system, would then be

$$l = l_s + l_{bb} + l_t + l_g \tag{2}$$

A summary of our before-class pipeline latency goals proceed as follows: The PDF will be scraped from Canvas (2s), the bounding boxes of the graphs will be collected (1s), the processing stage will white-out graphs (100ms), the slide embeddings will be collected (1s), then the text will be extracted (100ms).

This brings us to a total of 4.2 seconds, but we provide some extra buffer time, and set our goal to 10 seconds. Below, we discuss the latency requirements for the limiting subtasks (other subtasks' latency values are negligible):

- The PDF should be scraped from Canvas to our server within 2 seconds, which is possible when using chunking.
- The text from the PDF should be extracted within 1.5 seconds, which is possible using the PDFReader Python package for a 100 slide, 5000 word presentation (which is on the high end of the number of lecture slides and word counts).
- [Old System Requirement] The slide embeddings from the Siamese Network should be computed and stored within 1 second. Getting a single output from the network takes about 10 ms (based on our observed measurements), and even if we assume sequential rather than parallel processing, getting these embeddings will take about 1 second for a 100-slide presentation.
- The graph description from the CNN-LSTM should be generated within 5 seconds. Based on our observed measurements, inference can be done in the 400-500 ms range, but there may be additional time needed for loading and re-building the model from saved weights.

4.1.2 During-Class Latency

A summary of our during-class pipeline latency goals proceed as follows: user presses the button and an image is sent from the RPi to the server on the Jetson (600 ms),

the image is matched with a slide in the deck (2s), the corresponding text is fetched and sent to the iOS app (100 ms), and the TTS begins (100 ms). This totals 2.8 seconds, but we allow for plenty of leeway with our use-case goal of 8 seconds. Below, we discuss the latency requirements for the limiting subtasks (other tasks' latency values are negligible):

- The wireless data transfer speed for the image should take no longer than 600 ms to run. We can estimate that we'll send an image with a maximum resolution of 2 Megapixels where each pixel is represented by 3 bytes. Therefore, with a wireless data transfer speed of about 100 Mbps, the image would be sent within 600 ms.

- [Old System Requirement] The ML model for matching the captured image with a slide in the deck should take no longer than 2 seconds to run. This is where the majority of the latency in the "during class" pipeline will come from. Getting a prediction/embedding from a trained model will take around 10 ms (timed from experiments) since it just involves performing a series of matrix multiplications. This will have to be done for all slides in the deck, and then the captured image embedding will be compared to all other embeddings from the slide deck. We can expect this computation to take a similar amount of time ($<$ 10 ms), and given that most presentations are under 100-200 slides, this step should not take longer than 2 seconds.

- [New System Requirement] Were we to modify the previous design requirement to plan for our current system implementation, we would increase the latency design requirement for slide matching to 5.4 seconds, due to the need for using multiple models sequentially. Most of the latency typically comes from having to load and rebuild the models from weights, so we would allocate time (2.5s) to rebuild the detection and classification models. We also have some pre-processing steps and classification of digits, which should be no more than 10 ms each for inference; assuming the worst-case of a 3-digit slide number, this is 30 ms for those inferences, which brings us to a total of 5.4s when considering the detection inference time.

4.2 Size and Weight Requirements

We want our device to be a universal attachment onto the side of glasses. Therefore, it should be at most 100 mm long, since the sides of most glasses range from 120-150 mm long. It should have a width of at most 25 mm so it doesn't stick out as compared to the frame width of 125-150 mm, and it should not be taller than 35 mm, as compared to the average lens height of 32-38 mm [7].

Our weight requirements can be split up by component. We estimate that the battery will weigh the most since we want our device to have a long battery life, but it should not weigh more than 25 g. We found that most batteries satisfying our power requirements weigh between 20-25 g. Next is the 3D printed component case, which should not weigh more than 15 g. If we estimate that the case has a thickness of 2 mm, and calculate the volume from our size requirements, we get a total volume of (100 mm $\times$ 25 mm $\times$ 35 mm) - (96 mm $\times$ 21 mm $\times$ 31 mm) = 25000 mm^3, or 25 cm^3. If our plastic has a density of 1 g/cm^3, and we print using a 50% infill, the total weight would be 12.5 g, so 15 g is a reasonable bound. We can also adjust the infill of the part in order to decrease the weight if needed. Next, our on board computer should weigh at most 15 g, which is a reasonable expectation given that it needs to be small, and the attached camera should weigh at most 3 g. Finally, the buttons should weigh at most 2 g. In total, this would meet our desired use-case upper bound of 60 g for total weight of the attachment.

4.3 Power Requirements

The battery should be able to power our on board computer for about 6 hours according to our use case requirements. Therefore, if we estimate that the on board computer draws a current of 200 mA, we would want at least a 1200 mAh battery. Given that our buttons and camera will be attached to the on board computer, we do not have to separately provide power to them. The Jetson power consumption was considered, but we plan to have it plugged in at a central location in the school, so the power consumed will not matter.

4.4 Accuracy Requirements

1. The accuracy of the text extraction on the PDF should be 100% on standard font and symbols, and 95% on PDFs without standard formatting. We looked into the PyMuPDF reader and it worked extremely well, with close to 100% accuracy on special characters and even other languages.

2. The accuracy of the graph identification (recognizing the existence of graphs) should be approximately 95% - this is for identifying line graphs and scatter plots. The bounding box for these graphs needs to be sufficiently high such that it does not include any part of the slide with non-graph text (text other than graph title, axis labels, etc.).

3. The image-to-slide matching accuracy needs to be approximately 95%. Otherwise, we will be reading the wrong slide to the user, which would be confusing and counterproductive.

4. The graph description outputs should include the title and axis labels if they are present in the graph 95% of the time.

5 DESIGN TRADE STUDIES

5.1 Raspberry Pi

Our computing device needs to be able to send image data wirelessly to our server at the press of a button. It also needs to be small and lightweight so that it can comfortably fit on the side of glasses. The Raspberry Pi Zero WH fulfills all of these roles, and meets our requirements better than other options. It has dimensions of 65 mm × 30 mm × 10 mm and weighs 11 g, both of which meet our size and weight requirements. The W in the name indicates that it is compatible with WiFi, and the H indicates that it has GPIO headers which can be connected to our buttons. Another plus is that it has a built-in CSI camera connector, which we can take advantage of. Our next best option that we considered was the smaller ESP32-CAM board, which can serve as a backup option. However, we chose the RPi because it is a simpler solution, and the ESP32-CAM would require additional modules in order to transfer the images wirelessly. Although the RPi is larger, it still fits within our size requirements so we do not need to sacrifice functionality.

5.2 Camera

Since we are using a Raspberry Pi Zero, we also use a camera that is designed precisely for that board. Therefore, we chose the Unistorm Raspberry Pi Zero W Camera 5MP Mini Size Webcam. The camera portion itself is about 6 mm × 6 mm, and is attached to a 60 mm flex cable. In total, the camera weighs less than 2 g, which meets our weight requirements. We chose this camera module over other similar modules because the camera is not directly mounted onto a larger board, meaning it is small enough to meet the dimension requirements for our overall device. For example, the regular Unistorm module has a camera mounted to a 25 mm × 24 mm board, which is too large as our 3D printed component case should have a maximum width of 25 mm.

5.3 Battery

We need to be able to power our computing device, which our camera and buttons are connected to. Therefore, we chose the PiSugar 2 Power Module, which is a custom board made specifically for the RPi Zero and is a simple solution for powering the RPi. This weighs about 25 g, and has a 1200 mAh lithium battery. We chose the PiSugar 2 instead of connecting separate lithium batteries to a different power module because it is compact, convenient, and relatively small. Although the PiSugar 2 weighs about 5 g more than the alternative, we believe that the benefits regarding size and ease of use outweigh the fact that it weighs slightly more.

5.4 Jetson

We want to make sure that our ML models run quickly so that our system's latency is as low as possible. Therefore, we are hosting our server on an Nvidia Jetson, due to its relatively high computing power. This means that we can speed up our graph description and matching models, which will reduce the major components of our total system latency. However, in order to gain this increase in speed, the Jetson will definitely consume more power than if we had hosted our server on the RPi. We do not believe this will be an issue though, since as we mentioned in our power requirements, the Jetson will be plugged in at a central location such as the disability resources office.

5.5 Universal Attachment

We decided to create a universal camera attachment to glasses so that our users can easily use our product. We thought to attach the camera to the glasses rather than place it on a desk or other flat surface because the user can more easily control the direction that the camera is facing — all they would need to do is turn their head. Once we decided that the camera needed to be attached to the glasses, we decided to place the rest of the hardware components along with it. There are many reasons why a singular component box is advantageous to a separated system. First of all, it is much easier to keep track of a single piece of hardware and attach it onto glasses. Second, the start and stop buttons are easy to access, as they will always be on the side of the user's glasses. Third, we can avoid long wired connections between different components, which could cause safety issues. Overall, we want convenience: all the user has to do is attach our component box to their glasses, and press one of the two buttons.

5.6 iOS App

We decided to have all of the text and graph descriptions route through a phone app because we felt like it was the easiest for students to pair their headphones or listening devices with the phone itself because this allows the camera attachment and listening device to be separate. This way, the headphone wires will not get in the way of the camera. We assume that students already own headphones, because most audio captioning tools for televisions and phones already require a listening device. However, if they do not own one, a pair of basic wired headphones only costs about $10. We chose iOS rather than Android for two reasons: visually-impaired individuals overwhelmingly preferred Apple's VoiceOver feature to Android's TalkBack accessibility feature [8], and more United States residents own iPhones than Androids [9].

5.7 Graph Recognition Model

There are many different networks which are capable of performing object detection and extracting bounding

boxes; we chose the YOLO-v5 network for our design over other models. For object detection, there are two broad types of models: one-shot and two-shot models. These terms refer to the number of passes over the input image the model needs; for example, we were originally planning to use the Fast R-CNN model for object detection, and this (and similar networks like Faster R-CNN, Mask R-CNN, etc.) falls into the two-shot category. YOLO networks fall into the one-shot category. We decided to choose a one-shot model because these are better suited for real-time applications [10], as they only need to pass the input through one network to produce output bounding boxes for detected objects. We used YOLO-v5 which we observed achieves a 2.5× speedup on inference when compared to Faster R-CNN.

5.8 Slide Matching Model

There are a few different methods we tried to match the captured image of a slide during lecture with one of the pre-uploaded slides, and the method we chose ultimately depended on accuracy (these methods involved the classic speed-accuracy tradeoff).

We initially decided to perform the matching using a Siamese Network [11], from which we calculated similarity scores between all pairs (captured image, slide k) for all slides k in the pre-uploaded presentation. We discuss Siamese Networks in more detail in Section 6, but it consists of twin networks which output embeddings for a pair of inputs, and then calculates a similarity score or distance between these embeddings. With these distances, we could then identify the most similar slide and retrieve the corresponding text and description. This method resulted in an accuracy no better than random matching of a slide to the lecture, so we chose a different method.

Another approach we considered and tried was first using object detection to get a bounding box around the slide number, and then creating a multi-digit classification CNN to identify the number from the cropped slide number image; this was implemented as a CNN with 3 separate classification heads with output layer size of 10 nodes each. However, this method also led to a subpar accuracy no better than random matching, so we chose a different method.

The final method we considered and ended up using for the slide matching task was first using object detection, then contour detection, and finally single-digit classification of the contours. Similar to the method we described above, we first performed object detection to get the cropped slide number image, then contour detection to get a box around each digit, and finally classified each digit and combined the classification results to get a final slide number. The accuracy of this method is discussed in the Section 7, and the main tradeoff was increased accuracy at the expense of potentially increased latency from having to run multiple models sequentially as opposed to just one model.

5.9 Text Extraction

We chose to use PyMuPDF, a PDF to text reader, rather than an OCR model, like PyTesseract. PyMuPDF provided consistently higher accuracy. According to the benchmarks [12], PyMuPDF provides a 97% accuracy (based on 100% minus character error rate) on all PDFs, and we found a 100% accuracy on standard formatted PDFs. PyTesseract performed with a near 81% accuracy, and other PDF readers performed at between a 75% and 97% accuracy. After applying some natural language processing and spell-checking to the PyTesseract output, it was still only able to get to about 93% accuracy, and the latency was closer to 2 seconds when compared with PyMuPDF's average of 0.1 seconds. Out of all possible PDF to text readers (including PDFMiner, PyPDF2, PDFQuery and PyPDF), we chose PyMuPDF because of its speed: the only other PDF reader that matched PyMuPDF's accuracy was PyPDF, which has an average latency of 2.6 seconds. See Fig. 2 for speed comparisons.

#	Library	Average	1	2	3	4	5	6	7	8	9
1	PyMuPDF	0.1s	0.4s	0.2s	0.2s	0.2s	0.0s	0.1s	0.0s	0.0s	0.0s
2	pypdfium2	0.2s	1.9s	0.2s	0.2s	0.2s	0.0s	0.1s	0.1s	0.1s	0.0s
3	pdftotext	0.3s	0.8s	1.0s	0.3s	0.8s	0.1s	0.2s	0.2s	0.1s	0.0s
4	Tika	1.1s	12.9s	0.9s	0.6s	0.4s	0.1s	0.3s	0.2s	0.1s	0.1s
5	pypdf	2.6s	18.7s	4.8s	5.3s	2.3s	0.7s	0.9s	0.4s	0.5s	0.3s
6	pdfminer.six	4.5s	26.0s	12.9s	8.0s	4.6s	1.3s	2.1s	1.0s	1.2s	0.8s
7	pdfplumber	6.7s	41.7s	10.9s	11.5s	8.4s	2.4s	4.3s	2.0s	1.9s	1.9s
8	Borb	34.7s	111.2s	105.0s	1.4s	87.2s	21.1s	7.4s	83.5s	16.4s	20.3s

Figure 2: Comparison of Speeds for Different PDF Readers [12]

5.10 Canvas Scraping

We decided to scrape the PDFs directly from Canvas instead of having the user download the file from Canvas and upload it onto the app themselves. The latter process took, at best, 22 seconds for a sighted user, and we expect it to be higher for a visually-impaired user. The Canvas scraping takes about 2 seconds (to perform a handshake with Canvas, download the file, and send it to the app). While this is an immense improvement, it also requires the instructor of each of the user's courses to provide them with an API key which they will input at the beginning of each semester or quarter — the API key is required by Canvas' development guidelines. This entire setup process is expected to take less than 5 minutes once the student has received the API keys. Assuming the student takes 4 lecture classes with (a conservative estimate of) 14 lectures per semester, taking into account the 20 second difference between scraping from Canvas versus uploading a PDF manually, we get: $4 \times 14 \times (22 - 2) = 1120$ seconds, or about 19 minutes. The user will clearly save time with the Canvas scraping process. While the instructor might

suspect security concerns, the app will not be authorized by Canvas to download anything other than what is specifically under a "Lectures" module, which the students should be allowed to access regardless. However, this also means that the professor must publish a module titled "Lectures" and upload a PDF of their slideshow under this module at least 10 minutes before class, which we feel is a reasonable accommodation to make in order to enhance the learning experience of a visually-impaired student.

6 SYSTEM IMPLEMENTATION

6.1 Camera Attachment

Our camera attachment should have the ability to send images wirelessly from our camera to our server at the press of a button. To accomplish this, we will create a 3D printed component case designed to hook on to the side of the user's glasses. This case will hold the Raspberry Pi Zero WH, PiSugar 2 power module, Unistorm camera module, and the buttons. There will be holes in the case for the camera, buttons, and charging port, and there will be an attachment mechanism on the side opposite of the buttons. The start and stop buttons will be connected to the Raspberry Pi through the GPIO pins, so that the RPi can detect when they are pushed and then execute the associated task. When the start button is pushed, the Raspberry Pi will receive an image from the camera, and send it to the server on the Jetson through WiFi. When the stop button is pushed, the Raspberry Pi will indicate to the iOS app to stop playing the slide description audio. Fig. 3 is a model of our component box case, where you can see the Raspberry Pi and other electronics inside.

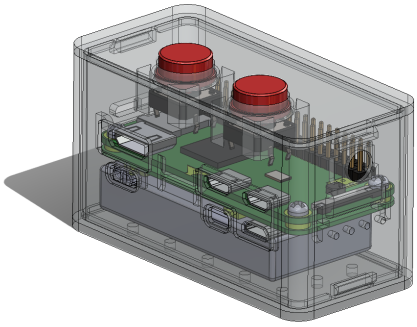

Figure 3: Transparent CAD Model of Component Case

6.2 Pre-Lecture Software Pipeline

See Fig. 16 in the Appendix for a diagram of the before-lecture software pipeline.

6.2.1 Canvas Scraping and PDF Download

To begin the pre-lecture pipeline, the user must first navigate to the app, which they can easily do using the native iOS Accessibility feature VoiceOver. Once they open the app, the name of our app, "SceneScribe" will be spoken out loud, because this is the heading of our application.

Then, they must choose the next lecture class they have from the button menu, so that the PDF of the lecture can be scraped from Canvas. See Fig. 4 below for a visualization.

Each button's text will be narrated using VoiceOver, so blind users will be able to tell which button they should click. When VoiceOver is turned on, users can swipe right to move to the next element, receiving a vibration to indicate that they have done so. Then, the element's text will be read out loud. For example, if they are on the top-most button, it will speak out loud "Class 1 Button". If they double-press anywhere on the screen while that button element is selected, that is registered as a button press, and the most recent lecture from the student's Class 1 course will be scraped.

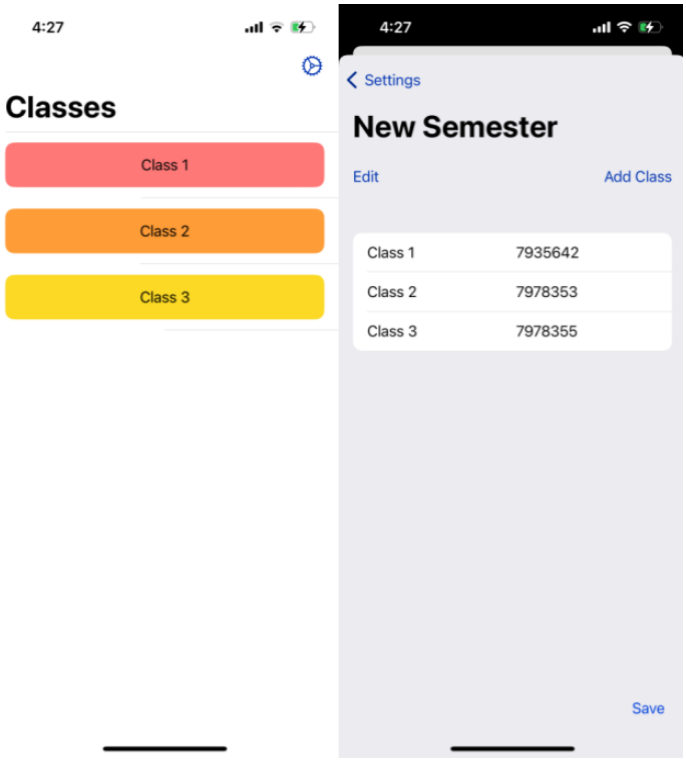

Figure 4: Class Selection Screen (left), Class Edit Screen (right)

In order to scrape anything from Canvas, we need to go through a security handshake. Specifically, we had to extract an API key from an authorized Canvas instructor account, and send it as a header in our HTTP GET request. Canvas responds with a JSON of the hierarchy of

all elements in the Canvas Course, with the class name at the top level, pages like "Home", "Modules", "Files", and "Assignments" below it, and all files within those pages under each page name. Because professors usually put their lecture PDFs under "Modules", we extract all filenames under "Class Name" → "Modules" → "Lectures", then request the bottom-most (which is the most recent uploaded) file, which returns a JSON with all metadata about the file, including the URL it is located. This is a sample JSON:


```
{
    {
        "id": 95444986,
        "title": "Week1Test.pdf",
        "position": 1,
        "indent": 0,
        "quiz_lti": false,
        "type": "File",
        "module_id": 13982117,
        "html_url": "https://canvas.
            instructure.com/courses/7935642/
            modules/items/95444986",
        "content_id": 230656097,
        "url": "https://canvas.instructure.
            com/api/v1/courses/7935642/files
            /230656097",
        "published": true
    },
    "tags": ["programming", "web development
        "]
}
```


Finally, we make a request to the URL under "html_url", and can accept the file returned in chunks of 1024 bytes.

6.2.2 Graph Identification Model

We need to be able to identify the existence of a scatterplot or line graph on a slide and get a bounding box around it. To do so, we will use the YOLO-v5 network, which will output the coordinates of the bounding box around the graph. See Fig. 5 below for the architecture of YOLO-v5 [13].

YOLO is a CNN, but with several key features. The first is a feature pyramid network, which is a feature extractor that takes an input image and produces feature maps at multiple scales; this feature pyramid network is the "neck" in Fig. 5, and it is comprised of layers taken, upsampled, and added together from the backbone. The second is the use of anchor boxes, which help pinpoint both the general shape and position of the detected object. The different heads (classification and bounding box regression) operate on the neck and help identify (1) the existence of an object of interest, and (2), the bounding box coordinates of the object of interest. By training YOLO on our custom dataset, we will be able to detect and identify bounding box coordinates around the graphs of interest.

6.2.3 Graph Description Model

For generating graph descriptions, we need a model capable of taking an image (the graph) as an input, and outputting a sequence (the caption/description). Image captioning is an extremely similar problem to our graph description task, and since these tasks are combined Computer Vision/Natural Language Processing tasks, they are solved using an architecture which combines CNNs with a sequence model: the Long Short-Term Memory network, or LSTM. These LSTM networks belong to a family of models called Recurrent Neural Networks, or RNNs, commonly used in generating output sequences such as sentences. The model essentially works in two parts: the CNN performs feature extraction from the image, and these features are passed to the LSTM, which generates the description. To give some more insight into how the description is generated, consider that on the first iteration, only the image embedding and a ⟨ SOS ⟩, or "start-of-sequence" token, are passed to the LSTM; the model then considers all tokens in the vocabulary and returns the token t with the highest probability of occurring next, based on the training data. On the next iteration, the image embedding, ⟨ SOS ⟩, and t are all passed to the LSTM, and it is asked to make a prediction for the most likely *next* token. This process continues until the LSTM returns ⟨ EOS ⟩, or "end-of-sequence", as the most likely next token, at which point we have our generated description. See Fig. 6 for the architecture of the CNN-LSTM [14].

6.2.4 Text Extraction

For text-extraction, we use a PDF to text extractor called PyMuPDF. The filename downloaded by the Canvas scraping software can be extracted, and thus the specific file can be opened and parsed with PyMuPDF. Then, it is parsed into a list that holds the text from each slide, indexed by the slide number. This extraction process will take about 0.1 seconds for even large PDFs. If we upload a presentation with the slide in Fig. 7:

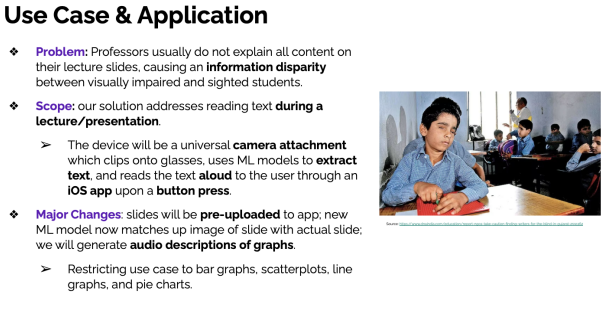

Figure 7: Example Slide for Text Extraction

We can extract the following text with PyMuPDF:

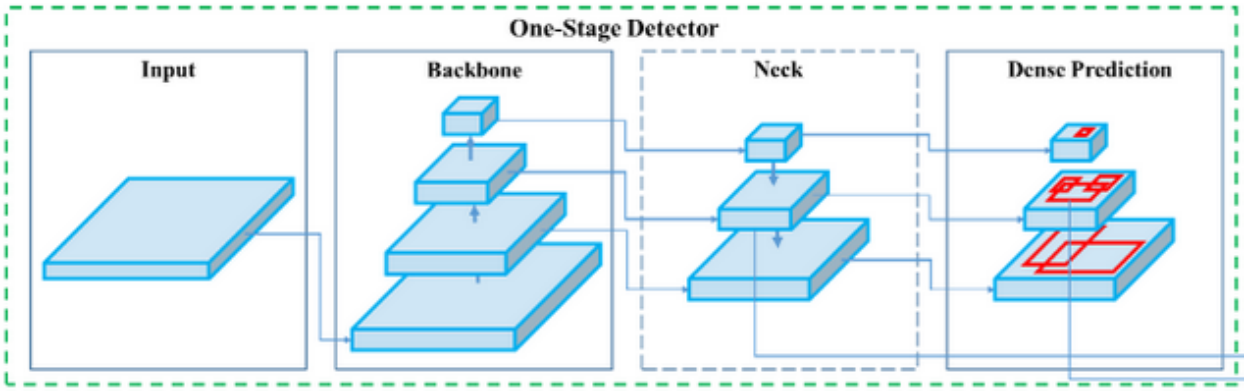

Figure 5: YOLO-v5 Architecture

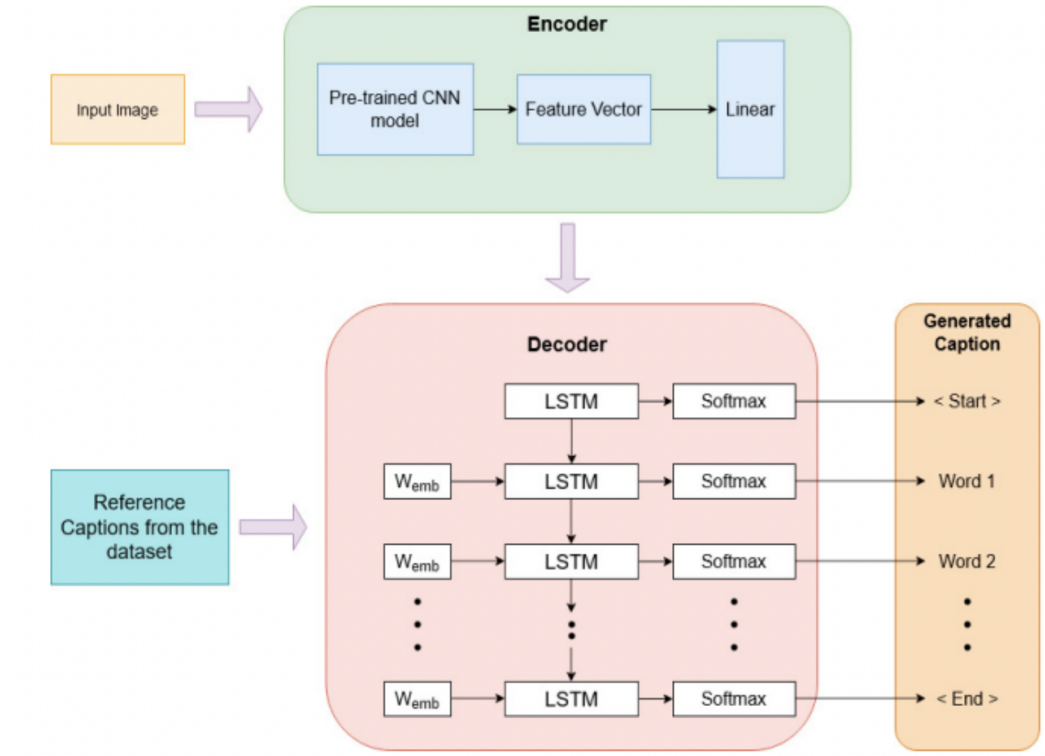

Figure 6: CNN-LSTM Architecture

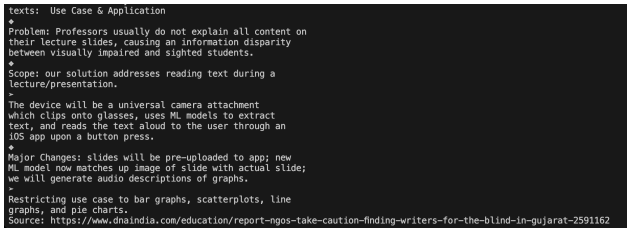

Figure 8: Extracted Text from Fig. 7

6.3 During-Lecture Software Pipeline

See Fig. 17 in the Appendix for a diagram of the during-lecture software pipeline.

6.3.1 Slide Matching Model

When the student uses our system to capture an image of a slide during class, this slide will need to be matched up with one of the slides in the pre-uploaded slide presentation, so that the corresponding text and graph description can be sent back to the app and read aloud to the user. To do this, we need a matching algorithm. Here are all of the approaches which we took to perform slide matching.

Siamese Neural Network This was the first approach we attempted to perform slide matching. This approach involved capturing an image of the slide during lecture, and computing an embedding for this image using the Siamese Network; embeddings were also computed for each slide in the lecture, and these were compared to the embedding of the image; the closest embedding in terms of Euclidean distance was then selected as the matching slide from the lecture. We used the SigNet architecture with a contrastive loss function, which attempts to space apart embeddings of mismatched slide/image pairs, and bring closer embeddings of matching slide/image pairs [11].

Here is a diagram depicting sample input slides and a captured image, and the output similarity scores computed between each pair:

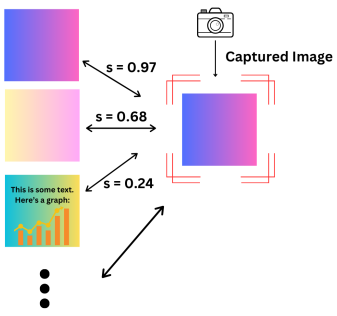

Figure 9: Similarity Scores for an Input Slide

We found that this approach yielded a very poor accuracy. We suspect that this may be due to a number of reasons, but specifically these:

1. **Differing Distributions of Captured and Slide Images:** When training a Siamese Network, it is important that the data distributions of images which we are comparing and trying to generate embeddings for are the same. Unfortunately, our data inherently comes from different distributions since a captured image of a slide using the Raspberry Pi camera will look much different from an image of a lecture slide taken straight from a PDF.

2. **Extreme Class Imbalance:** We trained the Siamese Network using pairs and labels of the form `(image, matchingSlide, 0)` and `(image, mismatchedSlide, 1)`. However, since for a particular image, there is only 1 matching slide, and $n-1$ mismatched slides (for an image taken from a lecture with n slides), the training data is very imbalanced, and this can lead to a higher bias in the resulting accuracy.

3. **Contrastive vs. Triplet Loss Function:** Studies have shown that triplet loss is much more effective at causing the network to learn good embeddings when compared to contrastive loss [15]. Since we implemented contrastive loss, we might have seen better results if we had implemented triplet loss.

Since the Siamese Network approach could not achieve an acceptable accuracy, we moved on to our next approach.

Detection & Multi-Digit Classification The next approach we tried involved further splitting slide-matching into two ML tasks. In order for this approach to work at all, we also had to rescope and require the professor to place slide numbers on their slides in a specific format (bold text, red box around the slide number in the bottom right corner of the image); we provided a script for the professor to modify their slides in this way. Here is an example of how the slides were modified.

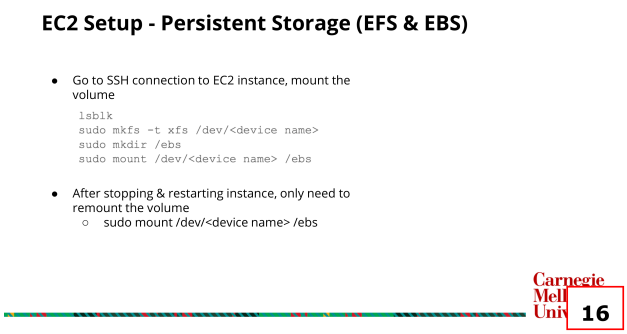

Figure 10: Modified Slide with Slide Number

The first ML task involved performing object detection with YOLO-v5 to identify a bounding box around the slide number box. This subtask required additional data collection and labelling ground-truth bounding boxes using the CVAT online annotation tool in order to train this detection model. An example of a detected bounding box from this subtask is shown below. The 0.93 value we can see on the bounding box is the confidence that there is a bounding box in that location (from the classification head of the YOLO-v5 model).

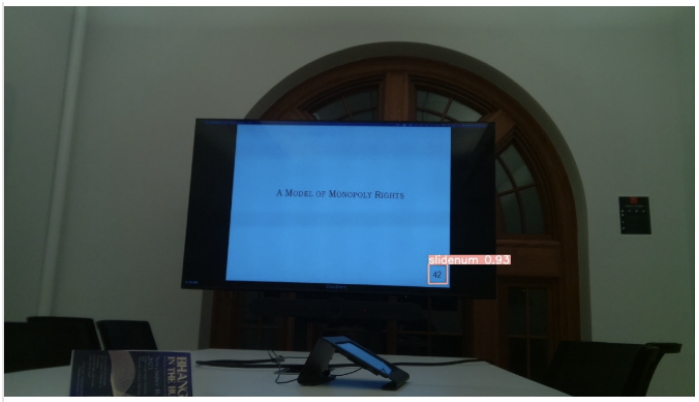

Figure 11: Detection of Slide Number

The second ML task involved performing multi-digit classification on the pre-processed (grayscaled, denoised, thresholded) cropped image of the slide number. Before diving into our multi-digit classification approach, we first briefly discuss how single-digit classification works. The input image of a single digit is passed through a CNN; the last few layers in a CNN are typically fully-connected (1-dimensional) layers. For the single-digit classification task, the final output layer has 10 nodes, and the node with the highest value represents the predicted class for the input image. This architecture is depicted below:

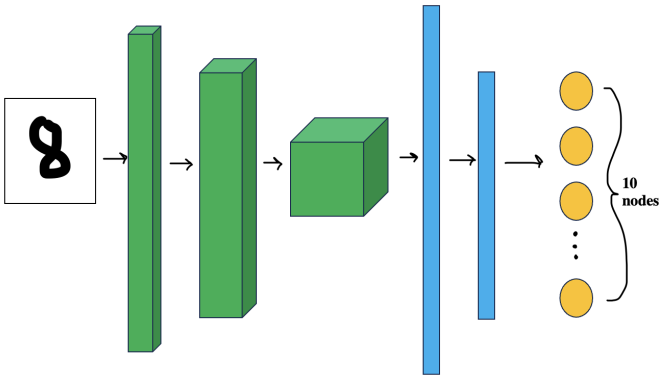

Figure 12: Single-Digit Classification Architecture

The premise of our idea for performing multi-digit classification was the following: lectures with over 1000 slides are practically nonexistent. Thus, we should be able to use a network with most of the same layers as single-digit classification, and have 3 classification heads: one for each digit. We implemented this with 11 output nodes in each head as opposed to 10 because not all slide numbers will be 3 digits. For a 2-digit slide number, then, we can consider the 3rd digit ground-truth output to be "10" - essentially, the 11th node should have a value of 1 and the other nodes should have a value of 0. Here is the modified architecture we created:

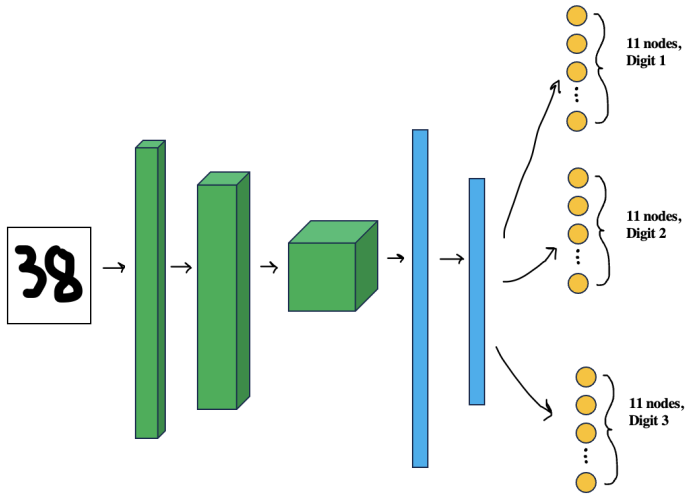

Figure 13: Multi-Digit Classification Architecture

Similar approaches had been successful on different datasets online, such as identification of numbers from street signs. However, when we tested this approach on our data to extract slide numbers, we saw a poor accuracy similar to the Siamese NN approach. Consequently, we decided to try yet another approach for slide matching.

Detection, Contours, & Single-Digit Classification The final approach we tried also involved splitting slide-matching into two ML tasks, and additionally an intermediate Computer Vision task. This approach again requires the professor to place slide numbers on their slides in a specific format (bold text, red box around the slide number in the bottom right corner of the image); see Fig. 10 for an example.

We then pre-processed (grayscaled, denoised, thresholded) the cropped image of the slide number, as in the previous approach, and performed an intermediate step of contour detection using `opencv`'s `findContours` method. This contour detection allowed us to detect a bounding box around each digit in the slide number.

The second ML task then involved performing single-digit classification on each contour. An example of this full pipeline is shown below: the leftmost image is the cropped slide number image, the middle image is after preprocessing, and the rightmost image shows the contour output (one digit at a time) which we are predicting on. In this

case, the rightmost image shows the first digit.

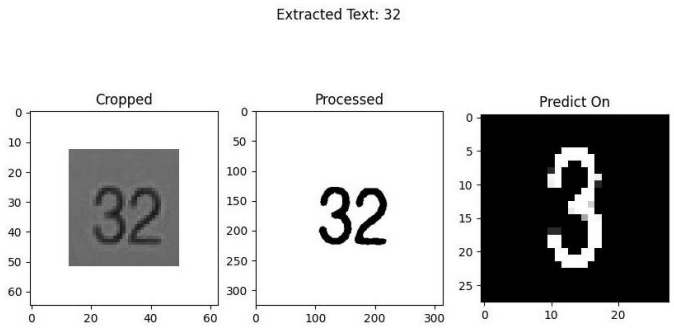

Figure 14: Cropped, Pre-processed, and Digit Contour Images

This method yielded the best accuracy, so it was the method we chose; the accuracy is discussed in Section 7 below.

6.3.2 Text Fetching and TTS

After the slide matching model identifies the slide number that the instructor is displaying on the screen, we can index into the list that holds all of the extracted text and graph data. The text from this list index is then posted to the iOS app. Then, the iOS app will use a speech synthesis object called AVSpeechUtterance to synthesize the text into speech, then a speech function can be called to actually speak it out loud. The app will also constantly be listening for a "STOP" button press, which can immediately cut off the text to speech.

7 TEST & VALIDATION

We performed many tests to validate the final performance of our device against our use-case and design requirements. Table 1 provides a summary of our general testing results, and Table 2 provides a summary of our user testing results.

7.1 Results for Accuracy of Text Extraction

We measured the test set accuracy from the PDF reader, using a metric called the character error rate (CER). The CER represents the percentage of incorrect output characters extracted, which clearly tests the accuracy of the text extraction, and this can be easily extracted. After performing text extraction on 50 well-formatted slides, we received a character error rate of 0%, which is what we hoped for during our design process.

7.2 Results for Accuracy of Slide Matching

For our slide matching ML model, unit testing involved taking a validation set of 110 images from the images of slides we captured with our device in TechSpark. We then tested both components of our slide matching system. First, we tested the detection of bounding boxes around the slide number on the slide. These numbers were detected with 100% accuracy. We then took the cropped images of the slide number boxes and then ran a second model on them in which we did pre-processing and then classification of each digit. These tests revealed an accuracy of 98.2%, so our total accuracy for the complete slide matching system from unit testing is 98.2%.

7.3 Results for Accuracy of Graph Detection

For our graph detection ML model, we gathered around 1000 real images of lecture slides which we split into a training and validation set. The validation set had a size of 100 images and our unit tests involved testing whether the graph detection model was able to detect good bounding boxes around graphs in the slides. To measure the accuracy of this, we used mean Intersection over Union (IoU) as a metric – calculating the overlap/total area of the predicted and true bounding boxes from what we labeled. We found that all graphs were accurately detected (100% detection rate) and the mean IoU was about 95%, so bounding boxes were reasonably good at capturing the whole graph.

7.4 Results for Accuracy of Graph Description

For our graph description ML model, our unit tests involved measuring the mean accuracy in terms of token-to-token matching with the reference descriptions. We conducted these tests on a set of 75 graphs out of the graphs we extracted from real slides, and this revealed an accuracy of 96%.

7.5 Results for User Experience

In order to test our system's user experience, we created a test presentation. We had volunteers go through the pre-lecture steps, then had them use the device for each slide of the test presentation. We asked the following questions:

1. How accurate was the description of the text on the slide? **(Completely inaccurate, somewhat inaccurate, neutral, somewhat accurate, completely accurate)**

2. How accurate was the description of the graph on the slide? **(Completely inaccurate, somewhat inaccurate, neutral, somewhat accurate, completely accurate)**

Table 1: General Testing Results

Requirement	Specification	Performance
Start Button Latency	Time between press and receiving audio: ≤8 s	Average Latency: **7.23 s**
Stop Button Latency	Time between press and no audio playing: ≤140 ms	Average Latency: **80 ms**
Attachment Weight	Weight: ≤60 g	Weight: **86.3 g**
Attachment Size	Max Size: 100mm × 25mm × 35mm	Size: **76mm × 42mm × 37mm**
Attachment Battery Life	Minimum battery life: 6.6 hours	Average battery life: **5 hours**
Slide Matching Accuracy	Slide matching accuracy: 95%	Accuracy: **98.2%**
Graph Detection Accuracy	Graph detection accuracy: 95%	Accuracy: **100%**
Graph Description Accuracy	Graph description accuracy: 95%	Accuracy: **96%**

Table 2: User Testing Results

Requirement	Specification	Performance
Accuracy of Text Description (1-5)	Average User Score: ≥4	Average User Score: **4.9**
Accuracy of Graph Description (1-5)	Average User Score: ≥4	Average User Score: **3.9**
Ease of Use (1-5)	Average User Score: ≥4	Average User Score: **4.0**

3. How easy was it to use the device? **(Very hard, somewhat hard, neutral, somewhat easy, very easy)**

Each answer corresponds with a number 1 to 5, and we hoped for an average score of at least 4 for each question. Overall, with 10 participants, we received average scores of 4.9 for text accuracy, 3.9 for graph description accuracy, and 4.0 for ease of use.

We received many comments from our users about our device. The common theme of these comments was that graph description needed more detail. This is likely also why our one failing metric for user testing was graph description accuracy. Users wanted more information about interpretations of the graphs, axis labels, and general context about what the graphs represent or why they are being displayed. One other comment that came up was that there should be an indication for when the displayed slide has changed so that the user knows when a different slide is being shown.

7.6 Results for Latency During Lecture

For the start button latency, we directly measured the latency between the moment the button press is registered and the time the app begins the text to speech, and then printed out the measured result. We did the same for the stop button, and also measure the amount of full words that are spoken before the audio stops. After 10 trials, we received an average "start" latency of 7.23 seconds, and an average "stop" latency of 80 ms, both of which were faster than our design requirements. The standard deviation for "start" latency was 0.23 seconds, and the standard deviation for "stop" latency was 4 ms.

7.7 Results for Latency Before Lecture

Before the lecture, when the user indicates which class they are attending next, we measured the time it takes for our system to finish processing the presentation PDF. We specifically measured the time from when the button press is registered to when the ML models have finished processing the slide information, and then printed out the measured result. We wanted this latency to be under 10 minutes, the length of an average passing period, and we received an average latency of 32 seconds after 5 trials, which is a passing test. The standard deviation for the latency was 6 seconds.

7.8 Results for Power

We wanted the device and app to be able to run consistently for at least 6.6 hours, the average number of teaching hours in a school day. After conducting 6 trials, our device battery was depleted in 5 hours on average, which is a failing test. However, the device can be fully charged in about 30 minutes, and an average 10 minute passing period can provide about 100 minutes of charge, which can last an entire lecture.

7.9 Results for Weight

We measured the device weight by zeroing out a scale with a pair of glasses, attaching the device, then placing it on the scale. The device should weigh less than 60 g, and if it weighs more than this we will have to consider decreasing the weight of the component case or choosing lighter components. Our component case weighs 86.3g in total, which constitutes a failing test, and made many glasses sag when mounted on the sides. This failing test is likely due to an underestimation of the individual hardware component weights, and this could have been solved by choosing a computing system even lighter than the Raspberry Pi Zero.

8 PROJECT MANAGEMENT

8.1 Schedule

Our schedule has evolved over the semester to account for extra tasks that each of us have taken on during the iterative process. On the software side, Aditi took on extra tasks with respect to the Canvas scraping software and integration. Similarly, Nithya needed to pivot to a more accurate machine learning model for slide matching, which led to a shift in the completion of the slide matching portion, as well as take on some extra integration tasks to combine the detection and description systems into one pipeline. The integration took much longer than expected. See Fig. 18 and Fig. 19 for a detailed breakdown.

8.2 Team Member Responsibilities

See table 3 for the work division; primary responsibilities are **bolded** and additional/secondary responsibilities are listed.

8.3 Bill of Materials and Budget

Our bill of materials is included as Table 4. Our initial purchases included the Raspberry Pi Zero WH, the Unistorm Raspberry Pi Camera Module, and the PiSugar 2 Power Module. We were able to acquire an NVIDIA Jetson Orin Nano Developer Kit from ECE Inventory, and had push buttons in our personal supplies. Our next purchases included an extra camera module, glasses frames to test with, and several clips and attachment mechanisms. We also used 3D printers in TechSpark to make our component case. Our total cost ended up being $153.10.

8.4 Risk Management Plans

A major risk for our system is the latency of the ML models being too slow. Our main mitigation technique was separating the system into the "before lecture" and "during lecture" stages. This allows the bulk of the latency-limiting work (the graph and text extraction) to be done when it is not as important to receive the information right away. During lecture, when it is important to receive timely outputs, we will only need to fetch the pre-extracted data.

Our slide matching model was also a big source of concern in our project. When we were using the Siamese NN at first, our accuracy was very low, and rarely (if ever) matched the slides properly. So, we decided to completely pivot to a different model that uses object detection through YOLO-v5, which allowed us to finally receive a 98.2% accuracy. While this solution was not what we had originally envisioned, we still mitigated against the bigger risk of the slide matching, and thus our entire final product, not working at all.

Another risk is the graph extraction model not having a high enough accuracy, so our mitigation techniques were to limit the types of graphs we can support, as well as collecting a lot of data from pre-existing datasets. We limited the types of graphs to line graphs and scatterplots.

The power consumption of the device and the iOS app constitute another risk. To mitigate, we optimized the number of operations we perform. Regarding the power consumption of the Raspberry Pi, we shut down any unnecessary functionality like the USB and HDMI outputs.

We also wanted to make sure that there were no privacy concerns related to our device, especially because it scrapes lectures from Canvas. We do not want to accidentally scrape a lecture that students should not be allowed to see, which is why we have mitigated this concern by only scraping the lectures that the professors have published under a specific module: the lecture can only be scraped if the student can actually view it on the Canvas website.

Our final risk was managing time and tasks between all members. We all had different course loads, and different projects that we were also working on simultaneously, so we built in a lot of slack to mitigate for this. This was extremely useful in the end, because we did not allocate enough time for integration, so the slack we had built in for ourselves allowed us to actually get the integration done, whereas if we did not allocate this extra time, it would have been much harder to have a completed project.

9 ETHICAL CONSIDERATIONS

We thought through a number of ethical considerations when formulating our solution. We found this especially important because nobody in our team is visually-impaired, and thus we wanted to be sensitive to the community involved.

Because our product will include an audio aspect, we must consider the auditory health of our users. Users must be able to adjust the audio output so that they will not face any damage to their ears. However, we have already taken this into account through our use case itself, because the device audio must be quiet enough such that the user can hear both the audio and the instructor speaking.

The product will also attempt to improve the user's mental health. Studies have shown that visually-impaired students are at risk for developing more mental and emotional problems than sighted students [16], and one reason might very well be the increased risk of falling behind in school due to the aforementioned information disparities. By helping to correct these information disparities, our product can decrease the risk of falling behind in school.

Our product includes multiple electrical components, so we also need to take into account the mitigation of safety hazards due to possible technical failures. Since the prod-

Table 3: Team Member Responsibilities

Nithya	Jaspreet	Aditi
Collecting/Generating & Labelling Graph Detection Data (1000 images)	**Setting up Image Capture on Raspberry Pi with Button Input**	**Implementing the Flask server**
Coding, Training, & Testing Graph Detection Model	**Wireless Communication between Raspberry Pi and Jetson**	**Creating the iOS app, making it compatible with VoiceOver and accessibility guidelines**
Annotating Slide Matching Data (850 images)	**Uploading and Configuring all Machine Learning Models on Jetson**	**Creating the Canvas scraper system**
Coding, Training, & Testing Siamese Network, Multi-Digit Detection, and Contour Single-Digit Detection Approaches for Slide-Matching	**Custom Design and 3D Printing of Hardware Component Case**	**Integrating the hardware, software, and ML models**
Collecting & Generating Graph Description Data (1000 images)	Configuring Raspberry Pi for Image Data Gathering	**Implementing wireless communication between RaspberryPi and iOS app**
Coding, Training, & Testing Graph Description Model (both Keras and PyTorch Versions)	Assisting with Wireless Communication Between the Raspberry Pi and Jetson and the iOS App	**Extracting text descriptions from slides, parsing PDF as images for graph detection model.**
Assisting with Integration of Detection-Description System and Slide Matching System	Gathering Feedback through User Testing	Assisting with preprocessing for slide matching model
Collecting Images for Slide Matching		
Writing Reference Graph Descriptions		
Full System Testing		

Table 4: Bill of Materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Raspberry Pi Zero WH	Zero WH	Raspberry Pi	1	$23.29	$23.29
Unistorm Camera	8541707548	Unistorm	2	$16.89	$33.78
PiSugar 2 Power Module	PiSugar2V2.1	PiSugar	1	$35.99	$35.99
3D Printed Component Case	N/A	TechSpark	1	$7.50	$7.50
NVIDIA Jetson Orin Nano	945-137766-000-000	NVIDIA	1	$0	$0
Push Buttons	N/A	Reland Sun	2	$0	$0
Glasses Frames	1043-10344	Kangaroo Manufacturing	1	$8.99	$8.99
Super Glue	6008WF-2	3M	1	$2.99	$2.99
#8 Self Drilling Screws	K-055-100P	Weideer	1	$7.99	$7.99
Black Metal Alligator Clips	RI-Shimeyao-49	Shimeyao	1	$12.99	$12.99
Silver Metal Alligator Clips	YAN-01	Yanecty	1	$9.99	$9.99
Mini Alligator Clips	4	MosBug	1	$9.59	$9.59
					$153.10

uct will be attached to glasses and very close to the skin, any technical failures should not physically harm the user. In order to mitigate this, we have designed our product in such a way that all electronics are safely housed in an insulated component box. This way, there is no risk of any component heating up and burning or electrocuting the user.

Finally, the product should also be easy to use for visually-impaired folks in order to promote welfare. In order to best promote ease of use, we included labels on every button we used in the iOS app, and made sure that Apple's VoiceOver feature would narrate all aspects of the app. We also made the design decision to limit the amount of operations the user needed to take in order to upload the lecture PDF to our app. Our design is to have the user press a button on the app associated with a course, prompting the app's backend to scrape Canvas for the lecture PDF, instead of having the user download the lecture on their computer, email it to themself, then upload it on the iPhone app.

10 RELATED WORK

There are several products which provide similar functionality to our system. We discuss these below.

1. BeMyEyes – this is a free app which connects visually impaired users with sighted users through a video call, and the sighted user describes what the visually impaired user is looking at. Our approach has a few advantages over this system. First, BeMyEyes causes the user to be dependent on another person by physically calling them to receive a description; this is not feasible for our use case of assisting a blind user follow along with lecture content, since having the blind student place a call during a lecture would be distracting for this student, other students in the class, and the professor. Additionally, the BeMyEyes solution provides a completely manual description as opposed to our system, which provides an automated description.

2. Envision Glasses – This product reads text aloud to the user through an app, similar to ours. However, our system has a few advantages over this product. For one, this product is extremely expensive, costing about $1900 for the read edition and $2500 for the home edition, which is the version that performs audio descriptions similar to our product. For reference, the cost of our product is about $100. Secondly, our system will perform graph description, whereas the Envision glasses only perform scene description.

3. OrCam – this product is primarily meant to be used as a handheld device which can be pointed at text in a close proximity, and reads the text to the user. It can be attached to glasses, but compared to our system, this is a disadvantage since the way that the user clips the device onto their glasses may not provide an optimal FOV for the camera. This system does not perform graph description, and the product is also quite expensive compared to our system: it requires a subscription of about $528 yearly.

11 SUMMARY

In summary, we aim to provide assistive learning technology to visually-impaired users by devising a camera attachment system which is able to read text and describe graphs to the student in a lecture setting. Our system will reduce the information disparity between sighted and visually-impaired students, giving these students the tools they need to succeed in the classroom. Our final product was able to meet many of the use-case and design requirements, and overall achieved the goal of creating a device to enhance lecture experiences for visually-impaired people. However, there are a few obvious additions we could have made to the system if we had more time:

1. The device should beep or vibrate when the professor has changed slides, so that the user can know when they need to press the button the device again.

2. The text-to-speech should be smoother and clearer, especially during sections of lots of bulleted or unequally spaced text.

3. We should collect more data for our slide matching model in order to make it much more accurate.

We learned many lessons about product development, teamwork, and time management over the course of the project. The most significant of which are the following.

Towards the beginning of the semester, we anticipated a schedule with very little dependencies, and very little overlap between individual team members' subsystems. However, there was a lot of overlap during the end of the semester when we began to perform the final integration, especially between the software and ML models. We should have either expected this early on, and left more than just one week to perform all of the integration, or performed the integration as each ML model was being completed. This way, we would only have to integrate one piece at a time, rather than all pieces together simultaneously, which caused lots of software issues.

We also did not come up with as many backup plans as we should have, and we recommend future students (especially those working with ML models) to have a plethora of backup plans at their disposal. When we had to pivot from one model to another at the very end of the semester, we had to scramble to collect hundreds of images to label and train with, when we really should have thought of a better backup plan initially in the case of our model failing its accuracy tests.

Glossary of Acronyms

- CER - Character Error Rate
- CNN - Convolutional Neural Network
- CVAT - Computer Vision Annotation Tool
- IoU - Intersection over Union
- JSON - JavaScript Object Notation
- LSTM - Long Short-Term Memory Network
- ML - Machine Learning
- RPi - Raspberry Pi
- TTS - Text-to-Speech
- YOLO-v5 - You Only Look Once Network (for Object Detection)

References

[1] E. Huber, K. Chang, I. Alvarez, A. Hundle, H. Bridge, and I. Fine, "Early blindness shapes cortical representations of auditory frequency within auditory cortex," *Journal of Neuroscience*, vol. 39, no. 26, pp. 5143–5152, Jun. 2019.

[2] B. Kosinski and J. Cummings, *The Scientific Method: An Introduction Using Reaction Time*. W.H. Freeman and Company, 1999.

[3] "Schools and staffing survey." (2008), [Online]. Available: `https://nces.ed.gov/surveys/sass/tables/sass0708_035_s1s.asp` (visited on 10/13/2023).

[4] "Phone battery statistics across major us cities." (Nov. 11, 2015), [Online]. Available: `https://veloxity.us/phone-battery-statistics/` (visited on 10/13/2023).

[5] Łukasz Bola, K. Siuda-Krzywicka, M. Paplińska, E. Sumera, P. Hańczur, and M. Szwed, "Braille in the sighted: Teaching tactile reading to sighted adults," *PLOS One*, vol. 11, no. 5, May 2016.

[6] "Voice qualities." (), [Online]. Available: `https://archive.ncvs.org/ncvs/tutorials/voiceprod/tutorial/quality.html` (visited on 10/13/2023).

[7] "Glasses measurements: How to find your frame size." (Jul. 27, 2022), [Online]. Available: `https://www.warbyparker.com/learn/eyeglasses-measurements` (visited on 10/13/2023).

[8] S. Knight. "Android vs ios: How do smartphone platforms compare on accessibility?" (Oct. 2021), [Online]. Available: `https://info.webusability.co.uk/blog/android-vs-ios-how-do-smartphone-platforms-compare-on-accessibility` (visited on 10/13/2023).

[9] L. Whitney. "Ios vs android market share: Do more people have iphones or android phones?" (Jun. 2023), [Online]. Available: `https://www.techrepublic.com/article/ios-vs-android-market-share/` (visited on 10/13/2023).

[10] R. Kundu. "Yolo: Algorithm for object detection explained [+examples]." (Jan. 2023), [Online]. Available: `https://www.v7labs.com/blog/yolo-object-detection` (visited on 10/13/2023).

[11] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," 2015. [Online]. Available: `https://api.semanticscholar.org/CorpusID:13874643`.

[12] M. Thoma. "Pdf library benchmarks." (Aug. 2023), [Online]. Available: `https://github.com/py-pdf/benchmarks/blob/main/README.md` (visited on 10/13/2023).

[13] J. Solawetz. "What is yolov7? a complete guide." (Jul. 17, 2022), [Online]. Available: `https://blog.roboflow.com/yolov7-breakdown/` (visited on 10/13/2023).

[14] A. K. Poddar and R. Rani, "Hybrid architecture using cnn and lstm for image captioning in hindi language," *Procedia Computer Science*, vol. 218, pp. 686–696, Jan. 2023.

[15] "Understanding ranking loss, contrastive loss, margin loss, triplet loss, hinge loss and all those confusing names." (Apr. 3, 2019), [Online]. Available: `https://gombru.github.io/2019/04/03/ranking_loss/` (visited on 12/14/2023).

[16] L. B. Augestad, "Mental health among children and young adults with visual impairments: A systematic review," *Journal of Visual Impairment Blindness*, vol. 111, no. 5, pp. 411–425, Sep. 2017.

12 APPENDIX

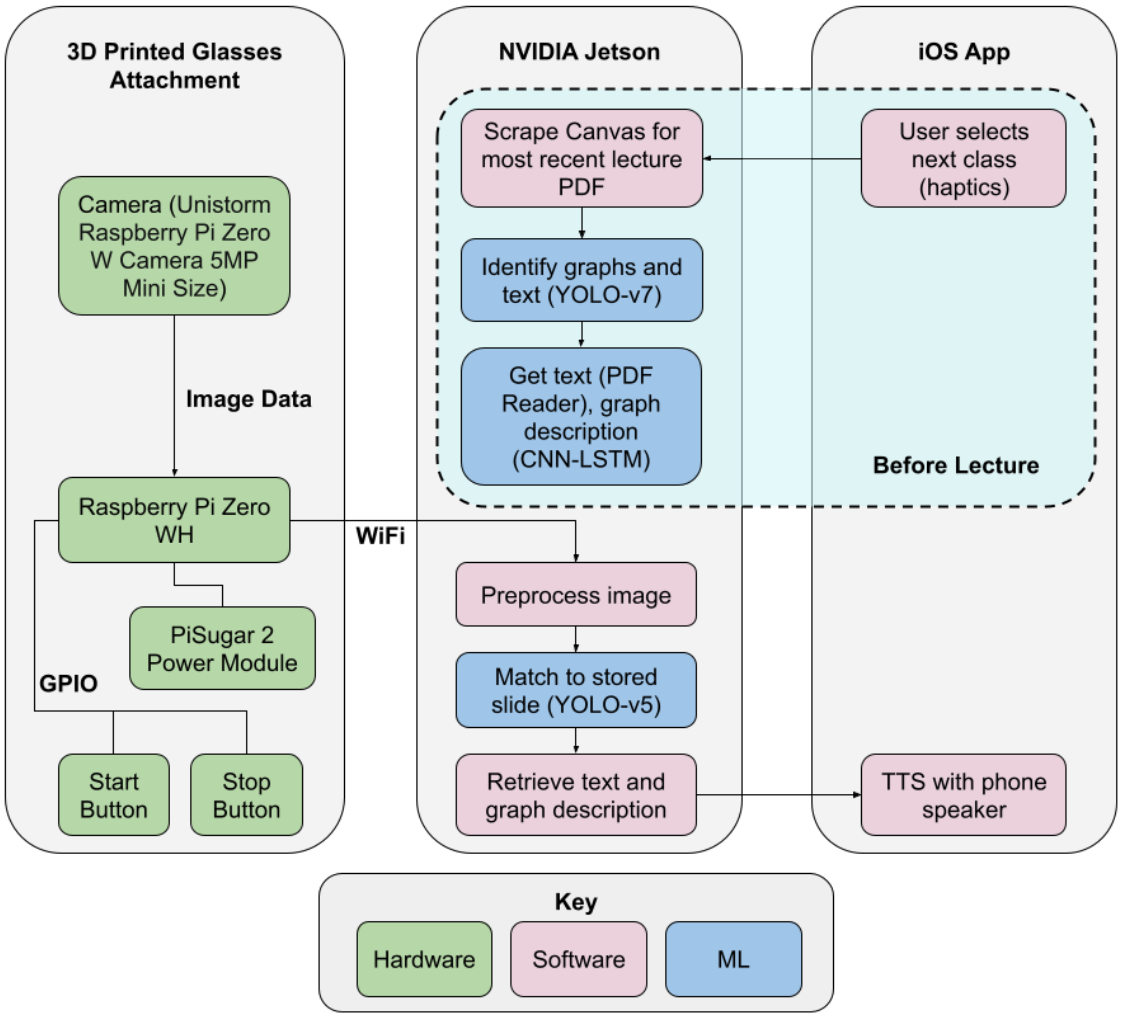

Figure 15: Block Diagram of System Architecture

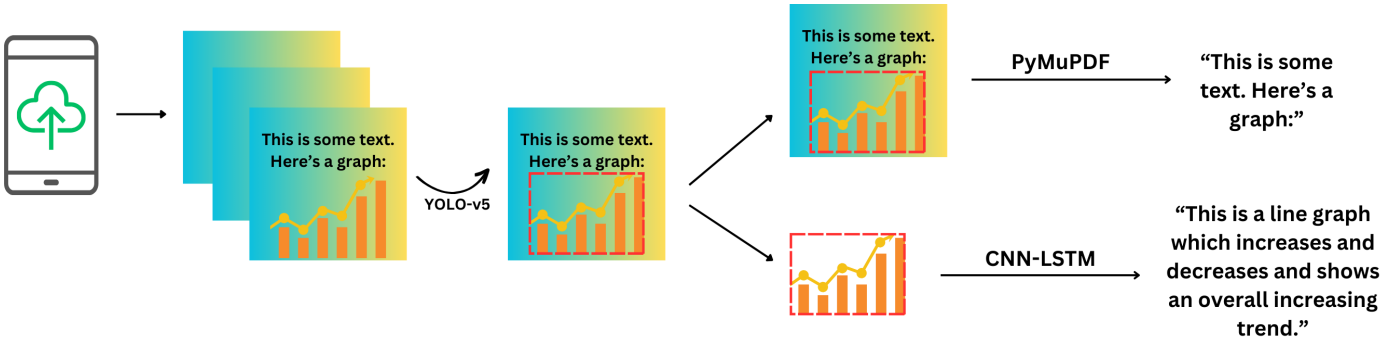

Figure 16: Visualization of Before-Lecture Software Pipeline

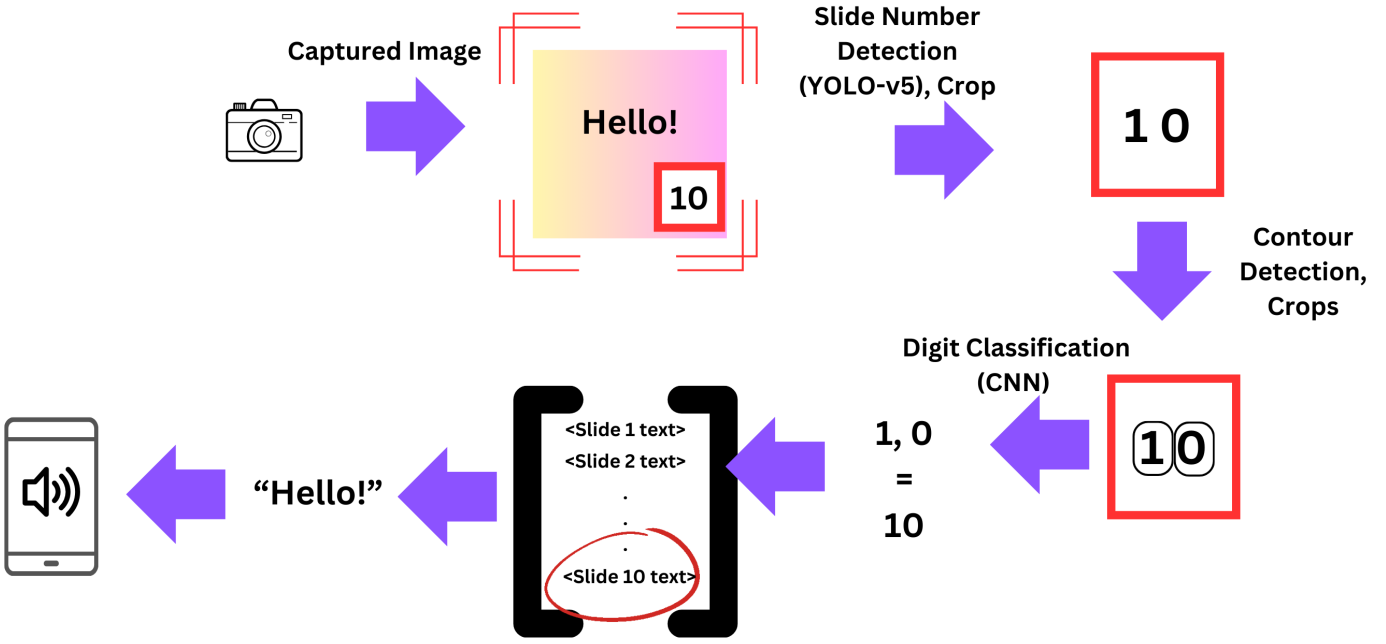

Figure 17: Visualization of During-Lecture Software Pipeline

Weekly Focus	Task	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
Initial software testing and ordering hardware	Pre/post-process data with CV techniques										
	Order all hardware										
	Create test project with simple text extraction model, check accuracy on dummy images										
Design a prototype	Set up simple iOS app and Flask server. Also talk to disability resources staff										
	Setup image to server pipeline										
	Research model for bounding boxes around graphs										
Refine the prototype	Gather and tag images for CV model										
	Test data transfer from camera										
	Start gathering and labeling references for graph description. Look into matching algorithm.										
Hardware Integration	Integrate hardware and app										
	Design glasses attachment										
	Continue gathering graph description data, finalize graph description model										
Finalize Prototype	Test the refined model on real people										
	Print and test glasses attachment										
	Validate whether the model performs well enough, make adjustments if necessary										
Modify Prototype Based on Feedback	Set up ML model on server										
	Refine hardware										
	Refine ML model based on people's feedback										
Final Adjustments	Integrate server and app (app-side)										
	Manufacture refined hardware										
	Integrate server and app (server-side)										
Overflow	Slack										
	Slack										
	Slack										
Final Testing	Test final device with real people										
	Test final device with real people										
	Test final device with real people										
Final Presentation + Report	Prepare final deliverables										
	Prepare final deliverables										
	Prepare final deliverables										

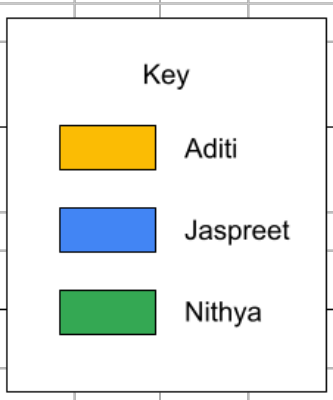

Figure 18: Original Schedule

Weekly Focus	Task	Week 10	Week 11	Week 12	Week 13	Week 14
Finalize Prototype	Add Canvas scraping software					
	Finish pipeline for sending images from Raspberry Pi to Jetson, and figure out WiFi on campus					
	Finalize and train graph description model					
Modify Prototype Based on Feedback	Gather and tag images for ML model					
	Design and 3D print the component case and textured button caps					
	Validate performance of ML models, and adjust as necessary					
Final Adjustments	Complete setting up software pipeline					
	Make necessary adjustments to final hardware design, and test subsystem					
	Refine ML model accuracy and speed					
Final Testing	Test final device with real people					
	Test final device with real people					
	Test final device with real people					
Final Presentation + Report	Prepare final deliverables					
	Prepare final deliverables					
	Prepare final deliverables					

Figure 19: Updated Schedule