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 Siamese Network, extract text and graph descriptions with PDFReader and a CNN-LSTM, and read them aloud to the user through an iOS app 2 times faster than current solutions.**

***Index Terms*—Accessibility, CNN-LSTM, Graph Description, iOS App, Machine Learning, Siamese Network, 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.

The first kind of information disparity can be displayed in the following scenario: imagine being a blind student in an engineering class, where the instructor has projected a scatterplot on the screen. The instructor, who forgets to explain the graph, begins by saying "This is a graph of the measured speed of a motor based on the input voltage. Using this data, we would predict that with an input of 15V, the motor speed would be 1000rpm". Even though the professor addressed the graph in their explanation, they did not explain the range of the axes or trend of the graph, which makes it difficult for the blind student to understand the reasoning for the professor's stated conclusion.

The second kind of disparity can be displayed in this scenario: Imagine, once again, that you are a visually impaired student in class. The instructor has already explained an important piece of terminology that is currently being displayed on the screen, but you have forgotten it. It would clearly be extremely helpful to you if you could read the slide in order to refresh your memory about the term. If you were a sighted student this would not be an issue, which is where the information disparity is apparent.

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.

We also spoke with someone who works as a Teaching Assistant at a blind school, who agreed with our analysis, and felt that the system would, in fact, be helpful to their students.

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 from sighted individuals 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 Competing Technologies

Competing technologies include free apps that connect visually impaired students with sighted volunteers through a video call [2]. The volunteer will then describe out loud what the student is "seeing". Other technologies are handheld cameras that can be pointed at text [3], which is then extracted as text to speech, and glasses that perform scene description [4]. Our system is advantageous compared to these approaches because it will perform not just text description, but graph descriptions. It will also be much cheaper, at about $100, when compared to the camera technologies mentioned above, which are in the $1000 to $2000 range. Our system will also allow visually-impaired users to be independent, which is not the case for the application where a sighted user will need to be involved.

1.3 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.

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 [5]. 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 [6].

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 [7], 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, scatterplot, bar, pie). 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 and shape 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 [8], 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 [9], 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. 12 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.

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 the YOLOv7 model to get bounding boxes around the particular types of graphs we are considering (bar graphs, scatterplots, and pie charts). At this point, we will have a processing step in which we white-out all of the pixels within the bounding boxes we have identified, so that any text within the graph itself, such as the title and axis labels, are not included as part of the extracted text from the slide.

  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 image will be pre-processed (deskewed, warped, grayscaled) before being passed to the matching algorithm, which will use a Siamese Neural Network to identify the most similar slide in the pre-uploaded slide deck. 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.

4 DESIGN REQUIREMENTS

4.1 Latency Requirements

4.1.1 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.

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

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 will provide some wiggle room, because we have not yet trained our ML models and thus do not know the exact time measurements, 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).
- 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.

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.
- 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.

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 [10].

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 × 25 mm × 35 mm) - (96 mm × 21 mm × 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 bar graphs, scatter plots, and pie charts. 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 (using the Siamese Network) 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, it makes sense to 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 plan to host 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 [11], and more United States residents own iPhones than Androids [12].

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-v7 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 [13], as they only need to pass the input through one network to produce output bounding boxes for detected objects.

5.8 Slide Matching Model

There are a few different methods we could have used to match the captured image of a slide during lecture with one of the pre-uploaded slides. We decided to perform the matching using a Siamese Network [14], from which we will get 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 between these embeddings. With these similarity scores, we will then identify the most similar slide and retrieve the corresponding text and description. An alternative approach we considered was to simply extract the text from the captured image using PyTesseract and then compare this text to that of each slide, extracted using the PDF Reader. However, we believe that the Siamese Network approach is superior for two reasons. First, the accuracy of PyTesseract may not be high enough to extract the text properly, so comparisons with the text from slides may be thrown off as a result. Additionally, using PyTesseract would mean we are discarding information such as images or font types which would be used by the Siamese Network to help make a better prediction, rather than just relying on the text alone.

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 [15], PyMuPDF provides a 97% accuracy 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. 1 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 1: Comparison of Speeds for Different PDF Readers

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. 2 provides a rough model of our component box case, with holes included for the camera and buttons:

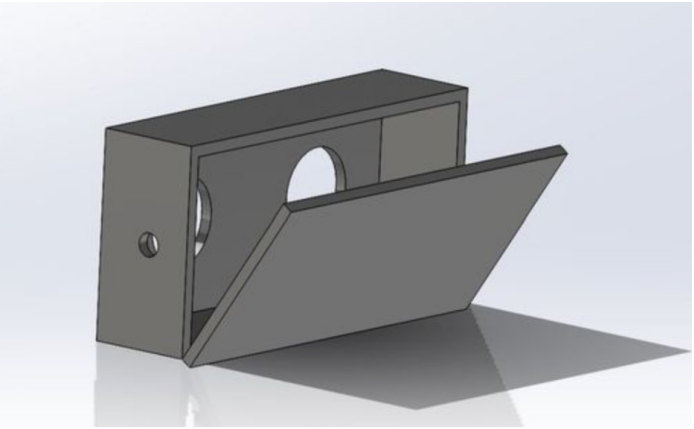

Figure 2: CAD Model of Component Box

6.2 Pre-Lecture Software Pipeline

See Fig. 13 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. 3 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 say “Organic Chemistry 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 Organic Chemistry course will be scraped.

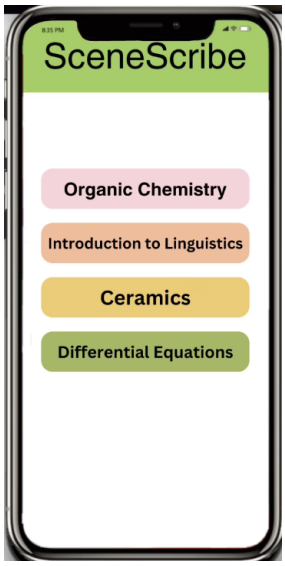

Figure 3: iOS App Layout

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 bar graph, scatterplot, or pie chart on a slide and get a bounding box around it. To do so, we will use the YOLO-v7 network, which will output the coordinates of the bounding box around the graph. See Fig. 4 below for the architecture of YOLO-v7 [16].

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. The second is the use of anchor boxes, which help pinpoint both the general shape and position of the detected object. 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. See Fig. 5 for the architecture of the CNN-LSTM [17].

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. 6:

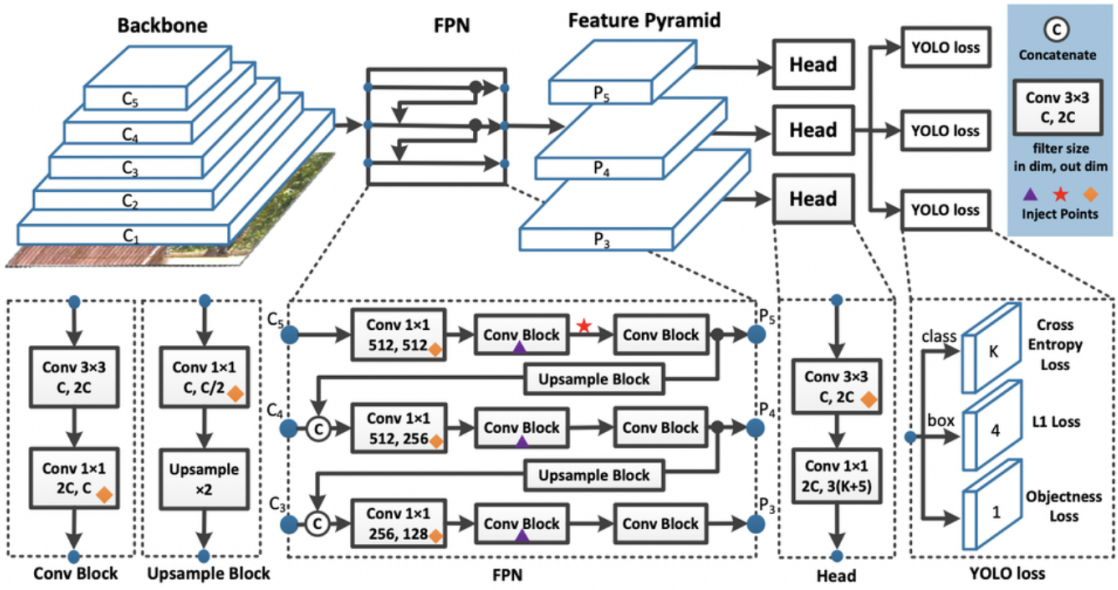

Figure 4: YOLO-v7 Architecture

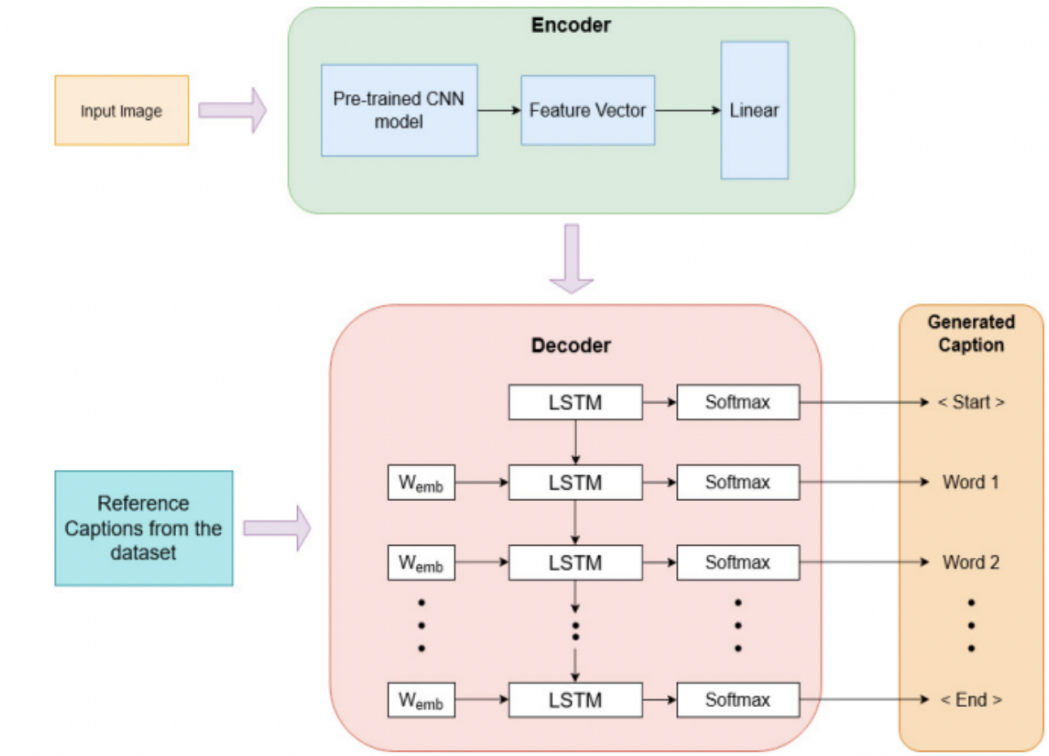

Figure 5: CNN-LSTM Architecture

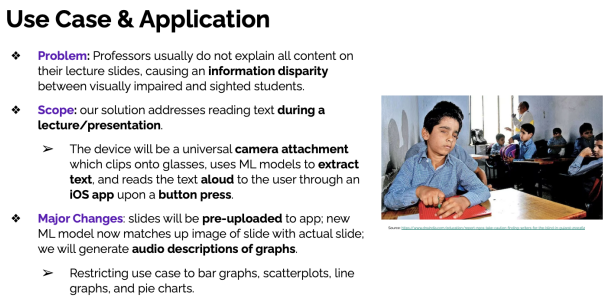

Figure 6: Example Slide for Text Extraction

We can extract the following text with PyMuPDF:

```
texts:  Use Case & Application
◆
Problem: Professors usually do not explain all content on
their lecture slides, causing an information disparity
between visually impaired and sighted students.
◆
Scope: our solution addresses reading text during a
lecture/presentation.
➢
The device will be a universal camera attachment
which clips onto glasses, uses ML models to extract
text, and reads the text aloud to the user through an
iOS app upon a button press.
◆
Major Changes: slides will be pre-uploaded to app; new
ML model now matches up image of slide with actual slide;
we will generate audio descriptions of graphs.
➢
Restricting use case to bar graphs, scatterplots, line
graphs, and pie charts.
Source: https://www.dnaindia.com/education/report-ngos-take-caution-finding-writers-for-the-blind-in-gujarat-2591162
```

Figure 7: Extracted Text from Fig. 6

6.3 During-Lecture Software Pipeline

See Fig. 14 in the Appendix for a diagram of the during-lecture software pipeline.

6.3.1 Slide Matching Model

First, we will preprocess the image by grayscaling it, deskewing it, and applying a canny-edge detection. Here is a before and after of the process:

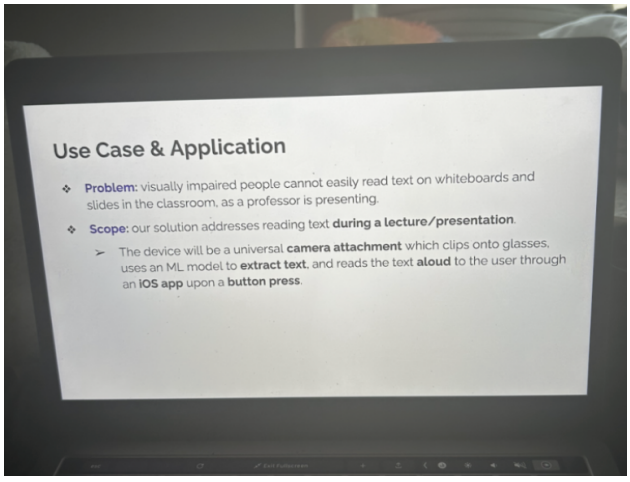

Figure 8: Slide Image Before Preprocessing

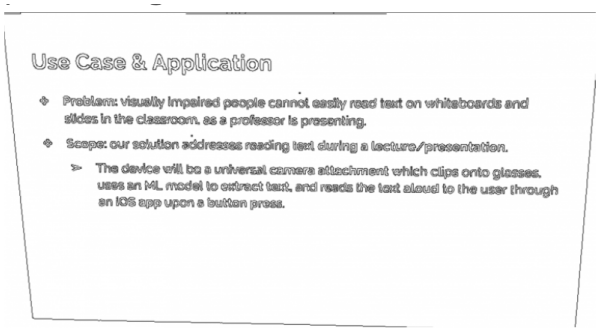

Figure 9: Slide Image After Preprocessing

It can then be compared with the pre-uploaded slideshow PDF using a Siamese NN model, and the best-matching slide can be identified. 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.

For the slide matching model, we will be using a Siamese Neural Network. See Fig. 10 for the architecture of the Siamese NN [18].

The architecture consists of two twin CNN subnetworks, and this network learns a similarity function between pairs of inputs by calculating embeddings for each input, and then computing the distance between these embeddings. To give an example application, face recognition uses this type of model to learn face embeddings and then compare an image to known faces to determine the identity. Similarly, we will compare the embedding for an image of a slide with the embeddings for each of the pre-uploaded slides to determine which is the closest match.

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

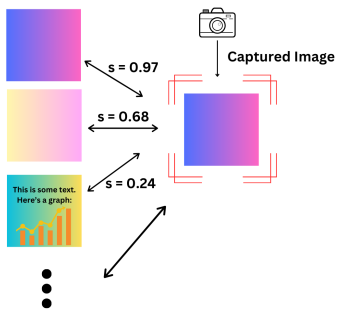

Figure 11: Similarity Scores for an Input Slide

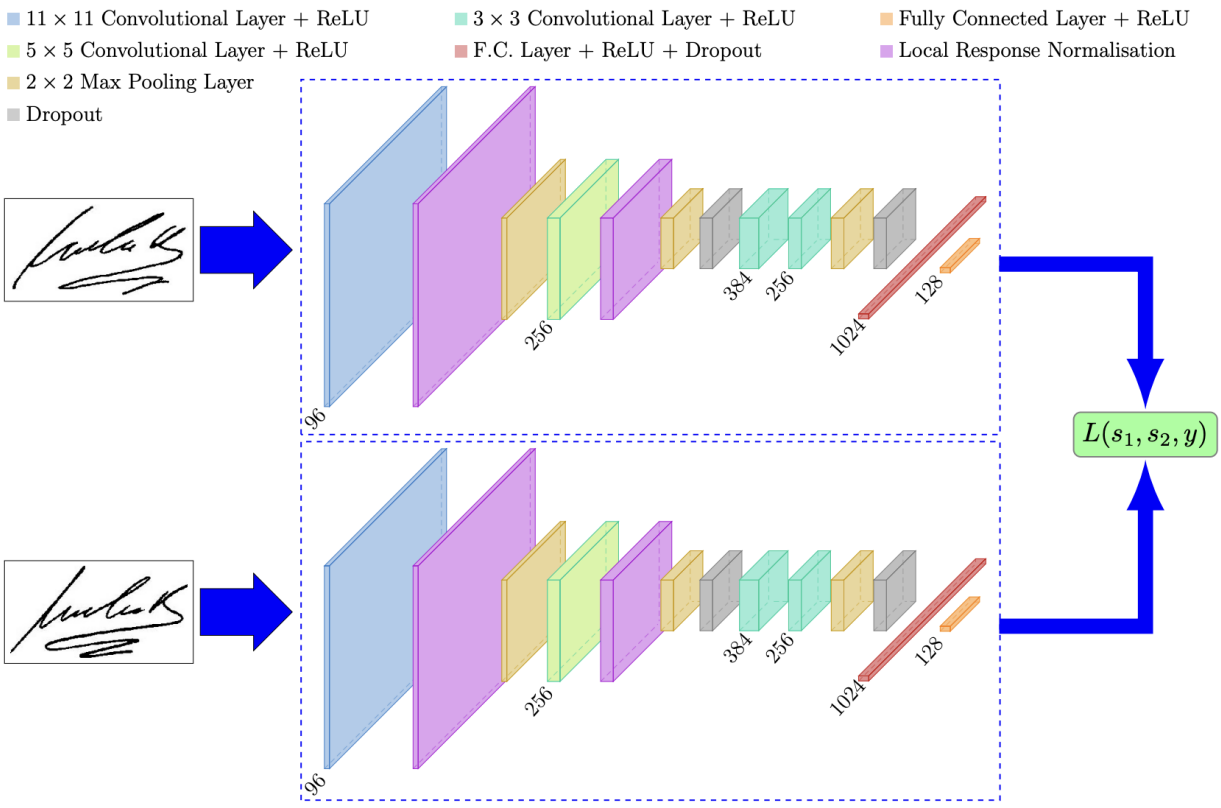

Figure 10: Siamese NN Architecture

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

7.1 Tests for Accuracy of Text Extraction

We will measure 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. We hope to achieve a 100% accuracy for text extraction.

7.2 Tests for Accuracy of Graph Description

Similar to the CER for the text extraction accuracy, we will use the similarity score between the reference and candidate embeddings to determine the accuracy of our graph descriptions. Our graph description templates were referenced in the use case requirements, so the outputs will follow that pre-assigned format. We hope to achieve a 95% accuracy for graph description.

7.3 Tests for User Experience

In order to test our system's user experience, we will create a test presentation. Slides will include all types of graphs we plan to use, slides with just text, slides with images, and slides with a combination of some or all of these elements. Then, we will have volunteers do the pre-lecture steps, then have them use the device for each slide of the test presentation. We will recruit at least one visually-impaired volunteer, and have them rate each slide's output as "Helpful", "Mostly Helpful", "Somewhat Helpful", or "Unhelpful" on a scale from 1-4, and average these ratings to calculate our metric. We will also recruit sighted volunteers to classify each slide as "Correct," "Mostly Correct," "Considerable Errors," or "Incorrect" on a scale from 1-4 and average these ratings as well. For both of these metrics, we will consider an average score between 1-2 to be satisfactory.

7.4 Tests for Latency During Lecture

For the start button latency, we can directly measure the latency between the moment the button press is registered and the time the app begins the text to speech, and then print out the measured result. We will do the same for the stop button, and also measure the amount of full words that are spoken before the audio stops. As mentioned in our requirements, we want the latency for the start button to be under 8 seconds and the latency for the stop button to be under 140 ms.

7.5 Tests for Latency Before Lecture

Before the lecture, when the user indicates which class they are attending next, we can measure the time it takes for our system to finish processing the presentation PDF. We will measure the time from when the button press is registered to when the ML models have finished processing the slide information, and then print out the measured result. We want this latency to be under 10 minutes.

7.6 Tests for Power

All of us have iPhones, so we will measure the amount of battery our phone consumes during the school day and average this value. Then, we will keep our app running in the background while using the device during a similar school day, measure the amount of battery our phone consumes, and check the difference in battery consumption. When we are testing with our sighted volunteers, we will also check how long it takes for the system battery to deplete with constant use of the device. Our system should not have to be recharged until 6 hours have passed since the last time it was at full charge.

7.7 Tests for Weight

We will measure 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.

8 PROJECT MANAGEMENT

8.1 Schedule

Our schedule is split among ourselves with very limited dependencies — almost none of our weekly work depends on someone else finishing their part of the project. We have also included room for slack, as well as time for completing the group deliverables and presentations, and enough time to account for school breaks. See Fig. 15 for a detailed breakdown.

8.2 Team Member Responsibilities

Jaspreet will be working mostly on hardware and hardware integration, Nithya will be working mainly on the graph description model and slide recognition models, and

Aditi will be working on Canvas scraping, the iOS app, and setting up server communication. We will all collect data for the ML models and work on major deliverables.

8.3 Bill of Materials and Budget

Our bill of materials is included as Table 1. Our purchases include 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. We also plan on using 3D printers in TechSpark to make our component case. Our total cost ended up being $83.67.

8.4 Risk Mitigation Plans

A major risk for our system is the latency of the ML models being too large. 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. If this mitigation technique is not enough, we plan on pivoting to a simpler model. 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 can also retrain the model with more data or to pivot to a more complex model depending on the tradeoff between latency.

Another risk is the power consumption of the device and the iOS app. To mitigate, we can optimize the number of operations we perform. Regarding the power consumption of the Raspberry Pi, we can shut down any unnecessary functionality like the USB and HDMI outputs. We can also increase the battery capacity if deemed necessary after testing.

8.5 Ethical Considerations

The system must be electrically safe, it should not overheat, and as a safety measure, the electronic components should not touch the user's skin directly, which is why we have encased them in a component box. The attachment should be light and comfortable to wear, so as to maximize user comfort. Our product should also only be used as a learning aid, not a substitute for accessibility or navigation tools. In order to limit complexity, our product will only be able to recognize English text, but to increase global accessibility, an expansion of the system can include different languages. For environmental concerns, our product should be energy efficient and should not require constant battery recharges. Finally, it should be affordable so as to appeal to a wider population.

9 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.

10 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. Challenges we anticipate include gathering enough training data for all 3 models (CNN-LSTM, YOLO v-7, and Siamese) and training them so they are able to achieve the accuracies set forth in our use-case and design requirements. We will tackle these challenges by gathering data early as well as augmenting our online dataset with synthetic data (generated by us), and using established methods (such as training

Table 1: 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	1	$16.89	$16.89
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
					$83.67

longer, tweaking learning rate and architecture, etc.) to reduce the bias and variance of our models.

Glossary of Acronyms

- CER - Character Error Rate
- JSON - JavaScript Object Notation
- LSTM - Long Short-Term Memory Network
- ML - Machine Learning
- RPi - Raspberry Pi
- TTS - Text-to-Speech
- YOLO-v7 - 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] "Be my eyes." (Oct. 2023), [Online]. Available: https://www.bemyeyes.com/ (visited on 10/13/2023).

[3] "Orcam." (Oct. 2023), [Online]. Available: https://www.orcam.com/en-us/orcam-learn (visited on 10/13/2023).

[4] "Envision." (Oct. 2023), [Online]. Available: https://www.letsenvision.com/glasses (visited on 10/13/2023).

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

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

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

[8] 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.

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

[10] "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).

[11] 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).

[12] 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).

[13] 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).

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

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

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

[17] 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.

[18] S. Dey, A. Dutta, J. I. Toledo, S. K. Ghosh, J. Lladós, and U. Pal, "Signet: Convolutional siamese network for writer independent offline signature verification," Sep. 2017. [Online]. Available: `https://arxiv.org/pdf/1707.02131.pdf`.

11 APPENDIX

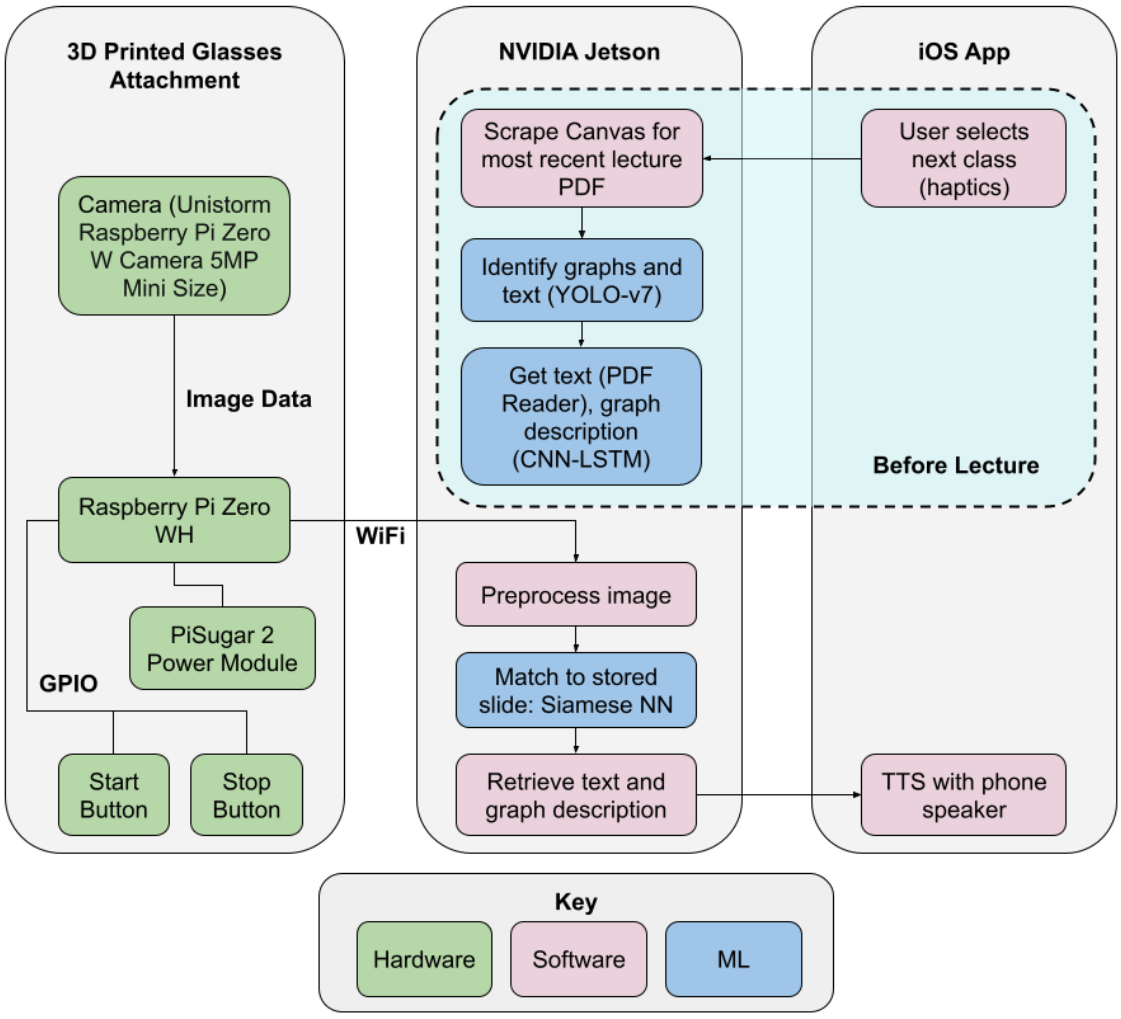

Figure 12: Block Diagram of System Architecture

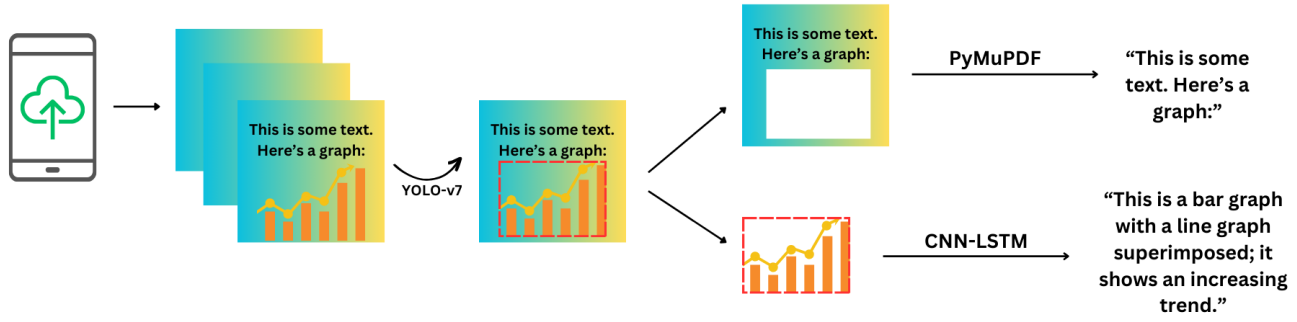

Figure 13: Visualization of Before-Lecture Software Pipeline

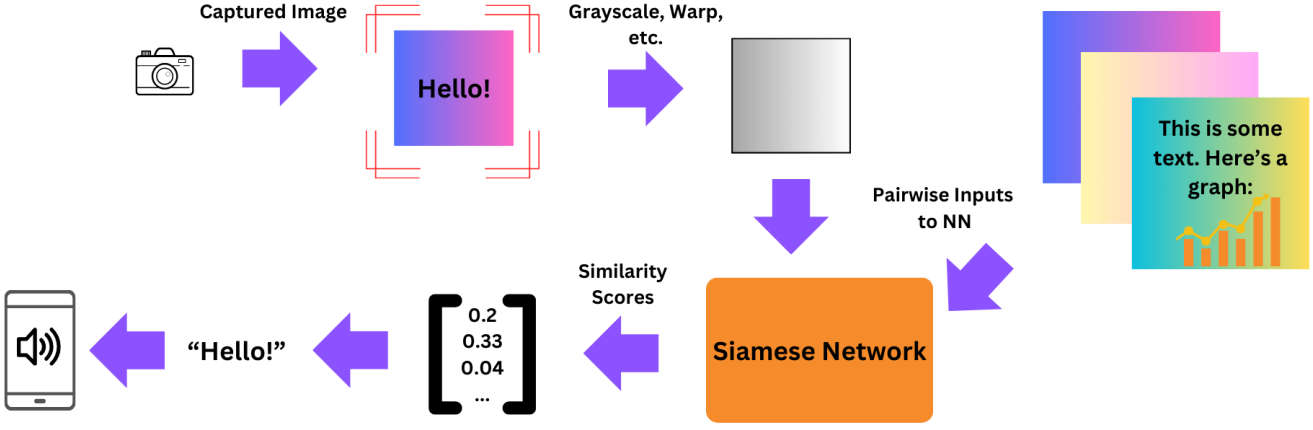

Figure 14: Visualization of During-Lecture Software Pipeline

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										

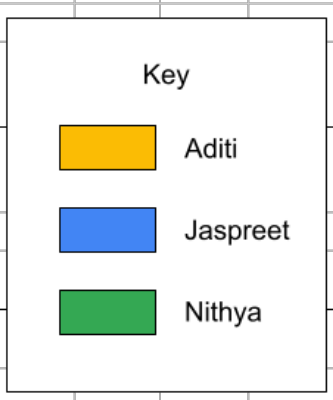

Figure 15: Schedule