iRecruit

Author: Mohini Banerjee, Shilika Gehlot, Jessica Meng: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—An interview assistant capable of providing software engineering jobseekers with the opportunity to practice for the interview process. Students are challenged with navigating fully virtual interviews and practicing how to conduct themselves during behavioral and technical interviews. Although there exist several written guidelines about common interview practices and questions, there is a lack of opportunity to practice a simulated interview with a "real" interviewer. iRecruit aims to give users a chance to practice for interviews through facial detection for behavioral interviews, and signal processing and machine learning for technical interviews.**

Index Terms—**Facial detection, Fourier transform, Haar cascades, Machine learning, Neural network, Signal processing**

I. Introduction

A common task people are faced with when searching for a job is interviewing. There exist several resources for assisting jobseekers with their interview process, such as lists of common behavioral interview questions and platforms to practice solving technical problems. Common technical interview platforms include HackerRank and LeetCode, where users are able to solve problems in a wide range of Computer Science topics such as Linked Lists, Dynamic Programming, and Strings. We want to improve upon current resources by creating a centralized platform where users are able to practice for both behavioral and technical interviews in a simulated environment. This way, users are able to gain an understanding for what real, virtual interviews are like.

For behavioral interviews, users are asked to video record themselves answering common behavioral questions. During the recording, iRecruit will provide real-time feedback on the user's eye movement and screen alignment. The goal is to detect the user's facial feature coordinates within 10 seconds, and alert the user after 5 seconds of subpar eye contact or screen alignment. For technical interviews, users are asked to audio record their skills, letter by letter. iRecruit will create a speech-to-text model to feed into a neural network that generates a technical question relevant to the interviewee's skillset. The goal is to identify the letter spoken with 60% accuracy and generate a question within 1 minute of post-processing.

II. Design Requirements

We have split the design requirements of our web application into 3 main components:

- Facial detection
- Signal processing
- Machine learning

Each component has a series of requirements that we are aiming to meet. For overall qualitative requirements, we want to make our code is consistent and well-documented. It should follow a coding standard that is readable and understandable for all team members. We also want to make sure that the iRecruit web application is able to save and remember a user's information, so that they are able to use a single profile each time they want to practice interviewing.

For the facial detection portion, there are three main requirements. The first one is the requirement for the initial setup phase. This initial setup phase is to allow the user to position themselves accordingly, and for the system to calculate a frame of reference that it will use for future off-center detection. The frame of reference should be calculated within 10 seconds, which should allow enough time for the user to find a position that is comfortable for them. A time less than 10 seconds may result in an insufficient frame of reference calculation, particularly for first-time users who are unaware of how the system works. The second requirement pertains to alerting the user of subpar eye contact or screen alignment. If a user's eyes or face is off-center, then the system should alert the user within 5 seconds. We did not want an immediate alert, because there are times where a user needs to look away from the screen for a very short period of time (1-2 seconds); for instance, for a screen break or to look at the time. The last requirement is for the accuracy of the facial detection portion. We aim to have approximately 80 percent accuracy, because we will be using the OpenCV library in Python. OpenCV has built-in Haar Cascades, which are pre-trained classifiers for features such as faces, eyes, and smiles. The accuracy of the Haar Cascades is around mid-90 percent, so we thought that aiming for 80 percent with our system that builds on top of Haar Cascades was a reasonable accuracy achievement.

For the signal processing portion, there are three main requirements. The first requirement is modifying the initial input. Our signal processing algorithm takes in an audio recording of the user speaking a single letter. The modification process will take two steps. First, we will remove the initial silence before the user starts speaking from the original file. This will get rid of unnecessary data (background noise) and shorten the input. Second, we will represent the remaining data as a feature vector. This will allow us to manipulate and transform the input to represent each letter in a way that a computer program can understand. The second requirement is to formulate the output. This output will be a feature vector that is fed into the machine learning algorithm. In order to make this output meaningful, the vector should have characteristics that

distinguish a specific letter from another, while having similar values for the same letter. The third and last requirement of this portion is testing the accuracy of our algorithm. This portion is difficult to test concretely for two reasons. First, the signal of different people recording the same letter will differ due to differences in pitch, loudness, and frequency. Second, the signal representing the same person saying the same letter will vary, as well. To ensure our algorithm is behaving as expected, we will compare the same letters spoken by the same person and check if the signals are similar.

For the machine learning portion, there are three main requirements. First, the input will be generated from the signal processing algorithm described above. We will reduce the dimension of the feature vector to make it easier for the machine learning algorithm to train. In order to build the algorithm, we will need to ensure we have ample training data. We plan on having approximately 20 feature vectors representing each of the 26 letters for a total of 520 samples of training data. This training data will be representative of the audio recordings of many different people to ensure that our algorithm is not biased towards a single person's voice. A subset of our training data will be used as validation data to prevent any overfitting. The second requirement consists of the output being a probability distribution across the different letters. The letter that the user spoke would be the one with the highest probability as determined by the algorithm. Third, we expect for the machine learning algorithm to have an accuracy of about 60%. We recognize that it will be difficult for our algorithm to pick up the differences between similar sounding letters such as "M" and "N". Speech recognition is a difficult task that many talented engineers have been working on for years. Since we are three college students with three months of time, we are aiming to produce a simplified version of the speech recognition algorithm that can correctly determine letters 60% of the time.

III. Architecture and/or Principle of Operation

We have split the architecture of iRecruit's system into 4 main parts:

- Web application
- Facial detection
- Signal processing
- Machine learning

The facial detection, signal processing, and machine learning components exist within the main web application. The facial detection portion will exist relatively independently, while the signal processing and machine learning portions will be integrated with each other. The web application is the baseline for the system, as this is where the user will be able to access the various components. The web application is split into three main components:

- Behavioral
- Technical
- Profile

When a user registers or logs into the system, they will be led to the dashboard, where they will be able to navigate to these three parts. If they choose to go to the "Behavioral" page, they will be presented with a randomly generated behavioral interview question and will be given the option to video record themselves responding to the question. iRecruit expects that the user is sitting in front of the camera and that their face will be visible in the video screen. When the user is recording, the facial detection portion will attempt to detect their face using Python's OpenCV library, specifically Haar cascades. This may require the user to position themselves accordingly and adjust their angle if necessary, as the Haar cascades are sensitive to the angle of the face. The facial detection algorithm allocates 10 seconds to an initial setup phase, to determine the frame of reference of the user's face and eye coordinates. Once the system has this frame of reference, it will refer back to this for each video frame. If the user's face or eye coordinates are not within a range of the frame of reference coordinates (also known as off-center), the system will alert the user within 5 seconds. This alert will be some type of visual and/or audio signal, so it is clear to the user that they are off-center and to reposition.

If a user chooses to go to the "Technical" page, they will be presented with a randomly generated technical question from our database that is relevant to the skills they have provided in the profile page. In our backend database, we have questions labeled easy, medium, and hard for each of the following categories - arrays, strings, object-oriented programming, linked lists, binary trees, recursion, and dynamic programming. With each question, the user will be provided with an example input and output that will help demonstrate what the programming solution is supposed to do. iRecruit will provide an additional input that will test the accuracy of the user's solution. After running their program with the question and input, they will be able to enter their output and iRecruit will let them know if it was correct or not. At this point, the user will be prompted to move on to the next randomly generated technical question.

The user's skill set, video recordings, and list of behavioral and technical questions answered will be saved in the "Profile" section of the web application. Here, the user will be able to view three things. First, they can review past video recordings if they want to look at their eye contact and screen alignment. Second, they will have a running record of all the behavioral and technical questions answered so far, as well as a summary of their results for each question. For the behavioral component, this will consist of a concise summary of their eye contact and screen alignment. For the technical component, this will consist of how accurately the question was answered. Third, they will be able to update and edit the recording of their skill set, so that they do not have to re-record each time they log in. The profile page will call the signal processing and machine learning algorithms in the backend each time a user decides to record their skill set.

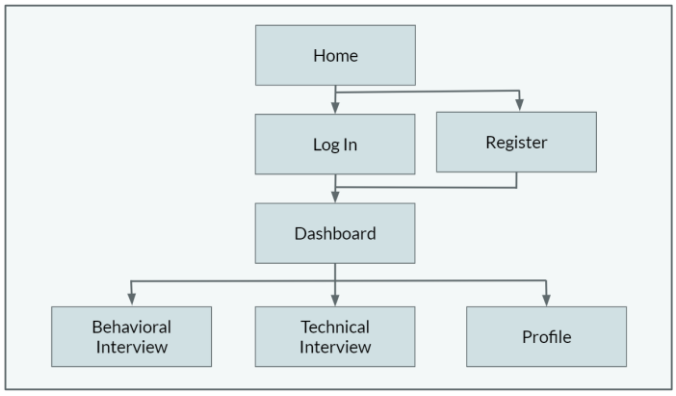


Fig. 1. High-level diagram of the system architecture.

Fig. 1. Is a high-level diagram of the system architecture that shows the different web pages we plan on having, as well as how the pages connect to each other. The behavioral interview, technical interview, and profile pages are where most of the backend processes (facial detection, signal processing, machine learning) happen, whereas the home, log in, register, and dashboard pages are more for sake of completeness of our web application and to provide an intuitive flow between different pages.

IV. Design Trade Studies

We considered various approaches to each of the components of the system and performed an initial research phase to weigh these approaches. There were a handful of options and algorithms that we took into account, but ultimately went with the ones that we thought would have the most available documentation and provide us with a high accuracy.

A. *Web Application*

For the web application, we chose to use Django, a Python web framework, to build it. Django is one of the more popular existing web frameworks that we have the most experience with. Many of the web components are already developed and it allows us to focus on the features we are trying to implement. There are four main reasons we chose to use Django. First, it is easy to use as it is based on the Python programming language. Since it is commonly used, there exists many tutorials and documentation for us to refer to. Second, it is fast and simple as making changes in the frontend or backend code does not require the whole system to restart. It also allows for separate testing of backend and frontend components. Third, it is extremely secure as Django security protects against clickjacking, cross-site scripting, and SQL injection. Lastly, it is suitable for any web application project of any size and capacity. It is scalable and can handle large amounts of data; it can be used on any operating systems including Mac, Windows, and Linux; it can incorporate multiple databases into the project to store information. Because of the above reasons, there were never any alternate web frameworks we considered to host our application. Django seemed like the best approach from the start.

B. *Facial Detection*

For the facial detection portion, we decided to use the OpenCV library in Python, particularly Haar cascades. There were two main Python libraries that we were considering for facial detection, which were dlib and OpenCV. dlib is a toolkit that contains machine learning algorithms and tools, and although it is principally a C++ library, it can also be used with Python. It has a Histogram of Oriented Gradients (HOG) feature descriptor, which is very powerful and actually more accurate than OpenCV Haar cascades. However, we ultimately chose OpenCV, because OpenCV is much more commonly used in Python. It also has more documentation and tutorials available, which we thought would be helpful because we were not familiar with facial detection before iRecruit. Haar cascades also have an accuracy of approximately 95 percent[9], which we believed was a sufficient baseline accuracy for us to build on top of for our aim of 80 percent accuracy. We were originally going to measure 3 things in the facial detection portion: eye contact, posture, and screen alignment. However, we decided to forgo posture, as there was no sufficient way to measure it. The first measurement we thought of was to attempt to detect the mouth, and if the mouth disappeared (user's head is down), that constituted subpar posture. However, if there is no mouth detected, there is also no face detected. Another measurement we thought of was to attempt to detect the shoulders and measure the distance between the shoulders and the center of the face, and if that distance was less than that of the frame of reference, that constituted subpar posture. However, if a user has long hair and the hair is covering the shoulders, shoulder detection would not be possible. Forgoing this measurement allows us to focus on implementing the eye contact and screen alignment portions, and making them more robust to meet accuracy demands. Facial detection accuracy will be calculated by the following equation:

$$Accuracy = \frac{Number\ of\ True\ Positive\ and\ Negative\ Tests}{Total\ Number\ of\ Tests} \quad (1)$$

True positive tests occur when the user does not have subpar eye contact or screen alignment, and the system does not alert them. False positive tests are when the user does not have subpar eye contact or screen alignment, and the system alerts them. True negative tests occur when the user has subpar eye contact or screen alignment, and the system alerts them. False negative tests occur when the user has subpar eye contact or screen alignment, and the system does not alert them.

C. *Signal Processing*

For the signal processing portion, we hit many roadblocks as we experimented with the best approach to reach the desired output. There are three main decisions we had to make. First, we were confused on whether to use the time domain or frequency domain to represent the signal. The time domain signal seemed like it provided us with more information, however the amplitudes of different signals representing the same letter were different. We recognized this was most likely due to differences in pitch, frequency, and loudness. Thus, we tried representing the signal in the frequency domain through the use of the Fourier Transform. This caused different problems as every signal, no matter the letter, looked similar as

each representation had a peak in the lower frequencies and another peak at the higher frequencies. The Fourier Transform tells us what frequencies are present in our signal. It also tells us how much power the time domain signal has at each frequency. By breaking a signal into its frequency components, it allows us to block out certain frequencies. We realized that although the information that the frequency domain gives is less intuitive than the time domain, ultimately it will help differentiate between letters.

The next two design tradeoffs are more specific. For starters, one decision we had to make was trimming the silence from the audio recording. Trimming it would distort the audio signal, but background noise silence may confuse the machine learning algorithm. Furthermore, we had to consider trimming the beginning of the signal, the end of the signal or both. The reason for keeping the noise at the end of the signal is to ensure that the length of every time signal is uniform. Trimming either the beginning or end will be harder to implement as it will need to be done from scratch, whereas if we trim the silence from both sides of the signal, we are able to use the built in trim function from the librosa library. We will need to experiment further to see which option will provide the ideal output.

The other decision involved using a windowing function to analyze the Fourier Transform. We decided to split the audio signal into 20ms chunks. In theory, this is equivalent to multiplying the signal by a rectangular window to extract the 20ms chunks. Once we realized that we were essentially performing windowing, we needed to analyze the tradeoffs between using different windowing functions. After doing some research on why windowing is necessary, we found that taking the Fourier Transform of a signal that is not perfectly periodic results in some discrepancies. The Fourier Transform assumes the time domain signal is a finite, periodic signal. When this is not the case, the endpoints of the signal are not continuous and this discontinuity is present in the frequency domain as high frequency components that do not exist in the original signal. Therefore, the Fourier Transform is not accurate as the different energy components are leaking onto each other. We found that windowing can help provide a solution for this problem. Windowing can reduce the impact of the discontinuity present in the time signal and thus provide a more accurate representation of the signal in the frequency domain. It consists of multiplying the time domain signal by a window of finite size that has a sinusoidal amplitude that approaches zero at the discontinuous parts. Out of all the possible windows, we saw that the Hamming window function best emulates this sinusoidal pattern as it has a wide peak but low side lobes.

D. Machine Learning

To implement the machine learning algorithm for our speech processing portion, we were debating two main approaches, neural networks and gaussian mixture modelling. The goal of a neural network is to program a simulation of connected human brain cells so eventually it can recognize patterns, point out similarities and differences, and behave like a human brain. As discussed extensively in the System Description, a complete neural network consists of the input and output layer, and one of more hidden layers. Increasing the number of hidden layers increases the amount of time it will take to train the algorithm, so when we start to finalize the number of hidden layers, this is something we will have to keep in mind. In addition to algorithmic decisions, we will need to finalize the number of training and testing data samples we use. It is common to use at least ten times as many training samples as the number of inputs. That means that for us, we will need to gather at least 260 hundred training samples of people saying the different letters. To improve the accuracy, this number could climb up to a thousand or more.

The Gaussian mixture model is a probabilistic model that represents subpopulations within an overall population that have a normal distribution. By gathering user input of people saying the different letters, we can create a model that would determine the probability of likelihood of a specific letter being chosen. We could then use this model to predict unknown utterances when the user is speaking a letter. The Gaussian mixture model will not require as many training samples as a neural network would as creating a probabilistic model in theory is less complex than trying to simulate the human brain. The two algorithms have different pros and cons. However none of us have any experience with Gaussian mixture modelling while two of us have experience with neural networks. Learning a new algorithm from scratch may not be the best use of our time, therefore, our team will first try the neural network algorithm as we are most familiar with it. If the classification is completely incorrect, we will look into using Gaussian mixture modelling. After deciding to go with the neural network, we discussed exactly what we are trying to classify even within speech recognition. The most advanced algorithms such as the once used to implement Siri, Alexa, and Google Assistant are able to identify full sentences. Due to our time constraint, we knew we'd have to narrow down our scope. Thus, we decided to make our speech recognition model identify the alphabet. This way, we are only trying to distinguish between 26 possibilities, the letters A through Z. Additionally, we discussed whether we would want to allow the user to speak a stream of letters into the input or one at a time. Because implementing any speech recognition, even something as basic as distinguishing between letters, is a difficult task, we decided to start with recognizing one letter at a time and will move on to a stream of letters if time allows.

V. System Description

A. Facial Detection

The facial detection portion of iRecruit will be implemented utilizing the OpenCV library in Python. When a user records themselves answering the practice behavioral question, the system will call the OpenCV VideoCapture class to begin a new capture of video frames at a rate of 30 frames per second. Additionally, the OpenCV VideoWriter class will be used for saving the video. For the alerts to the user, this will be done through OpenCV's putText() method and/or through the playsound library, to notify the user visually and/or audibly. There will be two parts that exist within the facial detection portion, one for eye contact and one for screen alignment.

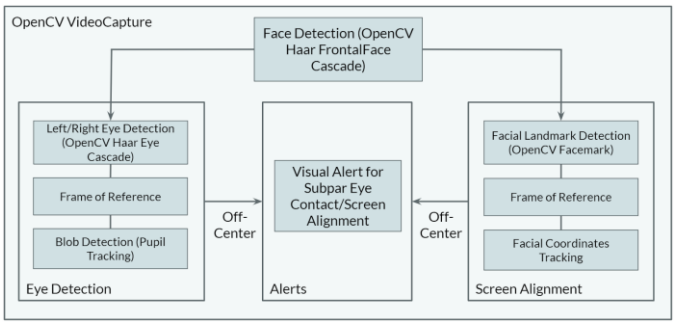


Fig. 2. Block diagram for facial detection architecture.

1. *Eye Contact*

The eye contact portion utilizes OpenCV Haar cascades as well as the numpy library. For each frame of the video, the system will use the frontal face Haar cascade to detect the user's face. The base image (video frame) is transformed into grayscale in order to detect the face, and the face is found on the original base image. After the face is retrieved, the system will use the eye Haar cascade to detect the user's eyes. The base face image is transformed into grayscale in order to detect the eyes. The height and width of the original face image is calculated using numpy to make the eye detection easier. This is because the eyes will always exist on the upper half of the face, and the left eye will be on the left half of the face, while the right eye will be on the right half of the face. After the eyes are retrieved, the system will track the irises/pupils within the eyes using a blob detector in OpenCV. The eyes are then transformed into grayscale and a threshold is set to determine the cutoff of which parts of the eye become black and white, which allows for the detection of the irises/pupils. When the irises/pupils are found, OpenCV moments are used to calculate the centroid. This center is calculated every frame, and the X and Y coordinates of the center are added to arrays over the period of the 10 second initial setup phase. The average of these coordinates is taken to find the frame of reference, which is what the system will use as the baseline of what constitutes centered.

2. *Screen Alignment*

The screen alignment utilizes OpenCV Haar cascades and the Facemark API. Similar to the eye contact portion, for each frame of the video, the system will use the frontal face Haar cascade to detect the user's face. The base image (video frame) is transformed into grayscale in order to detect the face, and the face is found on the original base image. After the face is retrieved, the system will use the Facemark API Local Binary Features (LBF) to determine the locations of all landmark facial features. This includes the eyebrows, eyes, nose, mouth, and edges of the face. These coordinates are calculated every frame, and they will be stored over the period of the 10 second initial setup phase. The average of the coordinates is taken to find the frame of reference, which is what the system will use as the baseline of what constitutes centered.

B. *Signal Processing*

The signal processing algorithm will be implemented using Python scripts. The input will be an audio file of the user speaking a single letter which will be read using the wavfile module within the scipy library. This module also automatically translates the audio into a float array which we can manipulate and modify. Once the array is stored, we will be manually trim the array to get rid of initial silence. This will be done using a specific threshold to determine exactly where the user starts speaking. Once we hit that threshold, we will truncate anything before that point. This updated data will be split up into twenty millisecond chunks. These chunks will be split using a Hamming window with 50% overlap. This will allow us to retain a better representation of the original audio data. The fourier transform will be applied to each chunk using the inbuilt fft function that the scipy library provides. In addition to applying the fourier transform, we will scale the values to render the numbers larger and more reasonable from the programmer's perspective. This two-dimensional vector - the dimensions being milliseconds and fourier transform value - will be the final output of the signal processing algorithm and the input into the neural network.

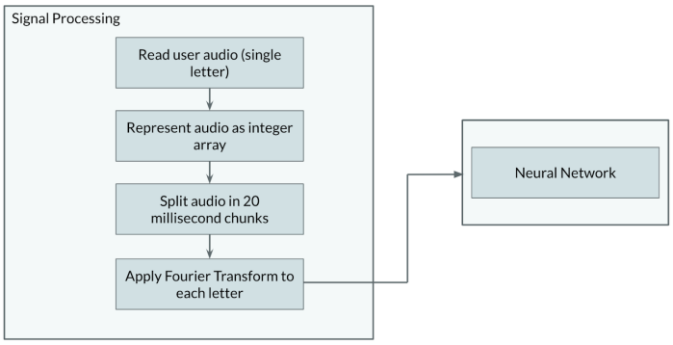


Fig. 3. Block diagram for signal processing.

C. *Machine Learning*

The machine learning algorithm that we will be implementing is the neural network. As described in the design requirements, the input will consist of a feature vector that represents the Fourier Transform of 20ms chunks of the time domain signal. There will be one or more hidden layers and an output layer that will be a probability distribution across the letters of the alphabet. Each hidden layer will have a length that is not necessarily the same as the length of the input layer. This length will be determined through trial and error, based on whichever gives the highest accuracy. These feature vectors in the hidden layer will be formed by performing a weighted linear combination between the feature vector in the input layer and the weights in a matrix α. The result of the linear combination will be passed through a sigmoid function in order to normalize the vector. Similarly, the probability distribution in the output layer will be formed by performing a weighted linear combination between the feature vector in the hidden layer and the weights in a matrix β. The result of the linear combination will be passed through a sigmoid function in order to normalize the probability distribution.

Training the model involves finding the optimal parameter matrices that minimize the Mean Squared Error given by this equation below:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)^2 \quad (2)$$

Y_hat, the predicted classification for each sample in the dataset, is found through the feedforward propagation described above with the current weight matrices α and β, which are found through a process called stochastic gradient descent. Y is the true classification for each sample in the dataset. N is the total number of samples. This process updates each of the weights in α and β by choosing one sample from the dataset and adding the current weight to the partial derivative of the mean squared error with respect to each of the weights. This is repeated until the MSE is successfully minimized.

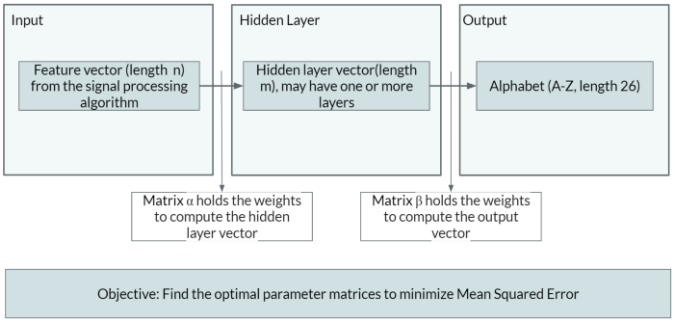


Fig. 4. Block diagram for neural network.

VI. Project Management

A. Schedule

Our full Gantt chart is in Appendix X. We made our schedule very detailed, so that each part was broken down into several parts. The schedule accounts for tasks that all team members are responsible for (e.g. abstract, proposal presentation), a research phase, and an implementation phase. The 3 main components of the requirements - facial detection, signal processing, and machine learning, were divided among each team member. Green indicates all team member responsibilities, purple is for Mohini, pink is for Shilika, and blue is for Jessica,

B. Team Member Responsibilities

Mohini - Mohini is in charge of the machine learning algorithm and ensuring that the accuracy of the neural network matches our expected accuracy of 60%. She will also be working with Shilika to form the input to the neural network, which is a result of the signal processing portion of our project. Mohini will also be designing the web application.

Shilika - Shilika is in charge of the signal processing algorithm. She will be working to ensure that the output is a reasonable representation of the audio that was spoken by the user and will continue to make enhancements and improvements to hone the output. Shilika will also be designing the web application.

Jessica - Jessica is in charge of the facial detection portion for the behavioral interview section. She will work on researching and implementing the facial detection algorithm and providing real-time feedback to users. This includes detecting facial features and alerting users of subpar eye contact and screen alignment.

C. Budget

iRecruit is a low-cost project, with the only purchase being AWS credits for EC2 to deploy the web application.

TABLE I. List of Purchases

Component	***Anticipated Cost***	***Status***
AWS EC2 Credits	~$10.00	In-progress

D. Risk Management

There were a couple of risk factors to consider for each portion of iRecruit that we need to consider. For the facial detection part, the biggest risk factor is the inability to meet our accuracy expectation of 80 percent. To mitigate this, we are using the OpenCV Haar cascades, which historically have an accuracy rate of about 95 percent. We believe that because the bulk of the initial detection is done using Haar cascades, the accuracy of that will continue to stay high, and any loss in accuracy will be due to the tracking and alert portions. Other risk factors include factors that may affect performance, such as contrast between different facial features, lighting, and background. For example, the contrast between the iris and the rest of the eye, may vary depending on the user. To account for this, we will allow users to set their threshold manually instead of having a hard-set threshold. This should also take care of the lighting. For the background, we will recommend that a user is positioned in front of a relatively empty background, so that the face is apparent on the screen.

There are three main risk factors to consider in the signal processing portion. The first risk factor is not being able to reach the accuracy expectations we have set for ourselves. We are aiming for a 60% accuracy on the overall speech processing algorithm. The output of the signal processing algorithm will have a big impact on the overall output of our speech recognition implementation. The second risk factor is ensuring that pitch and loudness will not have an effect on our signal processing. For example, different people speak at different volumes and have different pitches. These factors should not have a major input on the overall output. The last risk factor is distinguishing between letters that sound very similar, for example, letters such and ‘B’ and ‘E’ or ‘M’ and ‘N’. Ensuring that our algorithm is able to take into account the slight differences in the pronunciations of these letters will be key to making sure our algorithm is as effective as possible.

The biggest risk factor for machine learning is that our model may not work at all. As mentioned previously in this report, speech recognition is an extremely demanding task which requires many talented engineers spending many years working on it. If the input vectors are significantly different from our training data, the accuracy of our model may be extremely low. There are many factors beyond our control including the pitch, frequency, and loudness of the audio signal that may contribute to a lower than expected accuracy for the algorithm. Furthermore, we will need to gather enough varying training data to build a model that does not overfit to any particular data. Another minor risk factor relates to not being able to determine

the optimal parameters for our model. This includes the number of hidden layers, the number of epochs, and size of training and test dataset.

VII. Related Work

There are multiple current resources available for practicing behavioral and technical interviewing. For behavioral interviews, there are several articles that exist that give tips and commonly asked behavioral questions. These articles exist across many platforms, including many job-related sites, such as Indeed and Glassdoor. For technical interviews, there are several platforms that allow users to practice their technical and coding skills. This includes HackerRank and LeetCode, where users are able to select which topic they want to practice with and are given a question and an IDE to code on.

VIII. Summary

A. *Future Work*

If we do choose to continue working beyond the semester, one improvement we'd like to make has to do with the scope of the speech to text algorithm. Currently, we are planning on simply detecting individual letters. Future work would involve being able to detect entire words. This is a lot more complex, as we'd have to research phonetics as well as natural language processing algorithms. We'd also like to improve the time complexity of our speech processing algorithm as currently we are expecting it to take between 5 and 10 minutes to predict each letter. Another improvement we'd like to make is to offer more feedback in the behavioral interview section. Currently, we are offering eye contact and screen alignment, but would like to research other acceptable forms of feedback. Additionally, we would like to do more research into what constitutes good eye contact and good screen alignment as we are basing it off our personal experience. This would provide more valuable feedback to the user if we were able to specify the metrics of what constitutes as "good".

B. *Lessons Learned*

This is an important application to consider as many of us, as graduating seniors, are experiencing the hardships of the job recruiting process. Interviewing is rough and based on our personal experiences, we decided that having a centralized platform would make practicing for an interview less stressful. In terms of the design and implementation process of our project, we suggest having clear goals, deadlines, and vision for the end product before starting the implementation. Documenting the small success points along the way helps the end goal seem more feasible and attainable.

References

[1] "Dlib C++ Library." *Dlib*, dlib.net/python/index.html.

[2] "Gaussian Mixture Model." *Brilliant Math & Science Wiki*, brilliant.org/wiki/gaussian-mixture-model/.

[3] Guennouni, Souhail, et al. "A Comparative Study of Multiple Object Detection Using Haar-Like Feature Selection and Local Binary Patterns in Several Platforms." *Modelling and Simulation in Engineering*, vol. 2015, 31 Dec. 2015, pp. 1–8., doi:10.1155/2015/948960.

[4] "How Neural Networks Work - A Simple Introduction." *Explain That Stuff*, 17 June 2020, www.explainthatstuff.com/introduction-to-neural-networks.html.

[5] Jack, Srujan. "Face Detection Using Dlib HOG." *Medium*, 17 July 2020, medium.com/mlcrunch/face-detection-using-dlib-hog-198414837945.

[6] Lab, Sagara Idea. "What Is Django and Why Is It Used?" *Medium*, Medium, 4 Feb. 2020, medium.com/@sagarajkt/what-is-django-and-why-is-it-used-2dafdc75ce67.

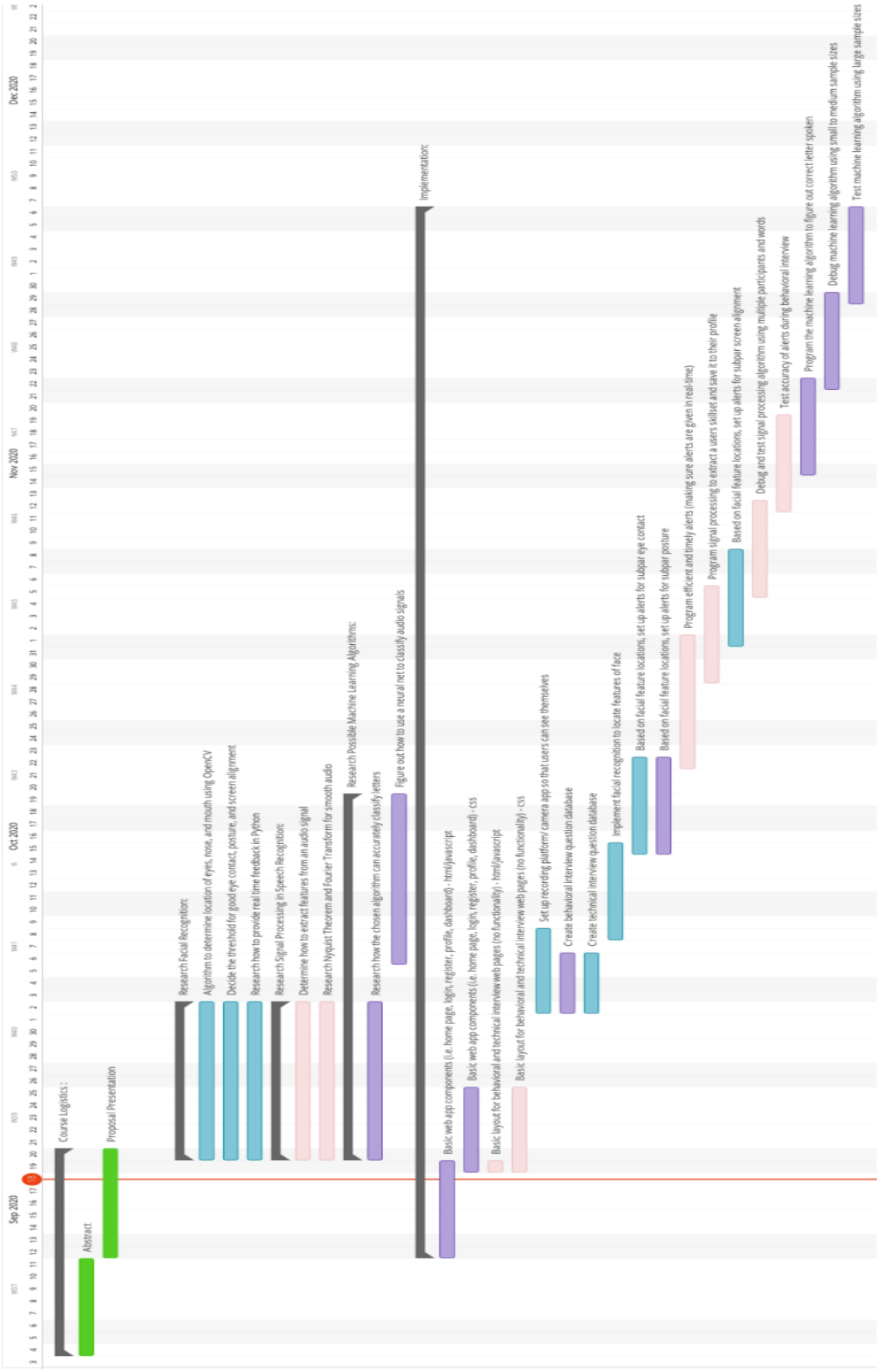
[7] Mallick, Satya. "How to Find Frame Rate or Frames per Second (Fps) in OpenCV (Python / C++) ?" *Learn OpenCV*, 12 Nov. 2015, www.learnopencv.com/how-to-find-frame-rate-or-frames-per-second-fps-in-opencv-python-cpp/.

[8] Scheidler, Pete. "Understanding the Basics of Fourier Transforms." *EnDAQ Blog for Data Sensing and Analyzing*, blog.endaq.com/fourier-transform-basics.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

[10] *Understanding FFTs and Windowing* . National Instruments , download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf.

IX. Appendix I. Schedule



X. Appendix II. Enlarged Block Diagrams for Facial Detection and Machine Learning