ASLearn

Authors: Hinna Hafiz, Aishwarya Joshi, and Valeria Salinas

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—ASLearn is a platform that serves as a convenient tool for users to learn and practice 51 American Sign Language terms through a web application interface. Our system will utilize computer vision and machine learning to process user inputs and detect what sign the user is making, ultimately providing feedback on the correctness of their sign. This will provide users with an experience to effectively learn ASL with correct and immediate evaluation results, keeping track of their progress as they work through different sign language modules.**

***Index Terms*—ASL, computer vision, LSTM, MediaPipe, neural network, web application**

I. Introduction

Hundreds of thousands of people within the U.S. alone rely primarily on American Sign Language (ASL) to communicate as a result of hearing loss, where millions more make up the American hard-of-hearing community. Encouraging people to learn sign language can help bridge the communication gap between hearing people and members of the hard-of-hearing community. However, learning sign language correctly can be difficult to do without an instructor, and many people may not have the opportunity to take ASL classes on a regular basis. ASLearn, our learning platform for American Sign Language, aims to combine the flexibility of remote learning with the interactivity of live feedback to give users an effective, engaging experience in learning ASL.

While there are currently many websites, mobile apps, and formal courses with live instructors for learning ASL, they have different tradeoffs that influence how feasible they are for people to use. With websites and mobile apps, users watch or read instructions on how to do signs and then have to practice them on their own, leaving students to figure out if they are signing correctly without any expert feedback. With formal courses, live instructors can provide the feedback lacking in online learning platforms, but such courses might present cost or scheduling barriers for many people. Thus, with ASLearn, we hope to reach people who have wanted to learn ASL but either found watching online tutorials to be too passive or could not commit to a formal course given their busy schedules.

Some key features of ASLearn include embedding the user's live webcam video feed into the online platform where users can see themselves attempting signs. In the 'learning mode' of our platform, the user will be presented with this video feed side-by-side with an instructional video and prompt. Using a combination of computer vision and machine learning, our platform will analyze the user's attempt at a given sign and provide feedback on whether or not they did it correctly, which will be displayed on an ASLearn web application interface. Another feature, the 'testing mode', will still contain the user's embedded video feed but no instructional video. Instead, 'testing mode' will only prompt the user to attempt signs that they have previously learned on the platform. Finally, there is a 'testing module' that quizzes users on a random subset of signs from our selected learning categories and records this performance to the user's profile, so they can see which signs they frequently get incorrect.

II. Use-Case Requirements

We defined use-case specifications around the computer vision of chosen signs, the accuracy of sign labeling, user distance from the camera, solution latency, and web application usability. These criteria combined, when met, will yield a successful platform that meets our intended use-case.

In regard to the computer vision of ASL signs, we have selected 51 signs that we want our platform to be able to teach and test users on. These signs make up an essential foundational understanding of ASL for beginners to work towards mastering. Specifically, we will be including the twenty-six letters of the English alphabet, digits 0-9, seven conversational signs, and eight signs related to learning. It should be noted that the alphabet and digits, thirty-six total signs, are static, whereas the conversational and learning signs are dynamic. The exact conversational and learning signs we are using include the following:

Conversational signs: how, you, my, name, yes, no, maybe, sign language (8 total)

Learning signs: school, major, ask, class, help me, what, word (7 total)

Thus, for one of our use-case requirements, we want our computer vision model to be able to detect that the user is attempting one of these 51 signs and then send data collected from the user's attempt to a neural network model that identifies what sign the user did. The model will be trained with open-source image and video data demonstrating correct ASL. We will determine whether the user has correctly done

the sign by comparing the identification generated by the model to the expected sign.

With respect to the accuracy of our platform, we chose a 97% accuracy metric for correct identification and feedback for user-generated signs. Our platform detects user signs after a user is prompted to attempt a specific sign; thus, we are able to compare the predicted label assigned to the user's attempt to the expected label requested in the prompt. Because our models are trained on substantial data that meets the community standard for proper ASL, along with the suitability of our model structure (discussed further in the implementation details) for video classification, our model will have a reasonable likelihood to achieve this accuracy metric. We allow for a 3% error rate to account for false positives, where our platform might indicate that an incorrect sign is correct due to its similarity to other signs or a camera angle that makes it resemble the prompted sign. Additionally, the error rate accounts for false negatives, which can occur as a result of environmental impediments such as lighting conditions, where our platform considers a sign to be incorrect even if it is done correctly.

A third use-case requirement we defined involves the distance between the user and the webcam recording them while they perform sign language gestures. This platform is intended for usage on a laptop or desktop computer, not a mobile device, so that users' hands can both be free to sign. In our research, we found that the typical distance between a user and their laptop is about 2 feet [1]. With this in mind, we require that the platform is able to accurately detect user signs at a distance of 3 feet or less from the camera, giving an additional foot to account for the user moving back to make sure they are fully in frame and facing the camera head-on.

Another use-case specification for ASLearn is latency, both with respect to web application responsiveness and the execution time of the deep learning models that predict what sign a user is making. For the web application, our research and academic work suggest that a reasonable latency metric is under 50 milliseconds [2], which is what we will aim for in terms of page and button responsiveness. Response times exceeding 50 milliseconds may negatively impact user experience. For both the deep learning model to determine what sign the user is doing and the web application to display that feedback to the user, we will require an overall latency of 2 seconds. Latency exceeding this 2-second requirement may make the platform frustrating for users. On the other hand, a shorter latency requirement may underestimate the amount of time needed to process input data and execute a model with said data to generate a label prediction.

Finally, the intention of our platform is to easily and conveniently learn ASL, so our usability requirements are specified with the intent of ensuring a smooth user experience. The two main components of this are that the site is easy to navigate and that the feedback given to the user after attempting a sign is easy to understand. In order to measure this, we conducted user surveys towards the end of the semester with about 5 users where they were asked to rate aspects of our platform from 1 to 5 (5 being good, 1 being poor). Our goal for this requirement is to have 90% user satisfaction in regard to navigating and understanding the site. We chose this number because we wanted to aim higher than the industry standard of about 75% [3] - based on the American Customer Satisfaction Index - for user testing because a significant aspect of our project is improving user experiences with online ASL learning.

To measure whether we achieve this 90% user satisfaction, we had 4 questions - related to user registration, platform navigation, the likelihood of continued use, and the likelihood of recommending the platform to others - within the user survey that require a 1-5 rating, meaning a total of 20 points per user survey. At 5 users, with 90% user satisfaction, we are aiming for 90/100 across all the surveys, rather than individual user satisfaction of 18/20 for all five users. This is so that our user experience results are not heavily influenced by one individual tester's experience, given the small number of users we plan to test our platform with. Note that some of these users will be experienced in ASL and others will be new to learning ASL. At the end of the survey, we will ask users if they had any additional comments to help us understand what users would like to see. If time constraints allow, we will apply this feedback to our solution to make it more user-friendly.

III. Architecture and/or Principle of Operation

The user video feed from the camera is embedded in our web application interface so that the user can view a reflection of how they are forming their sign language gestures in response to a prompt shown on the web app interface. Additionally, a ring light is used to maintain good lighting to lessen the likelihood of the user's hands not being detected in a dimmer environment. Figure 1 demonstrates this connection between our external setup and the internal implementation of our system. Observe that the video is also sent to a computer vision processing component that utilizes MediaPipe, an open-source framework that allows us to extract feature data from the user's hands. This feature data consists of absolute position coordinate data that provides information about the shape/orientation of the user's hand. This data is propagated to the machine learning component of the system to then be fed as inputs to a neural network model, which is selected from among 7 neural networks–see Figure 11 at the end of this report. Each model supports prediction generation for a specific subset of sign language gestures. The groupings are based on whether the signs are static (non-moving) or dynamic (movement required), as well as physical similarity.

Specifically, the categories for static signs include fist-shaped, one-finger, two-finger, three-finger, and open-hand for static. The two categories for dynamic signs are learning terms and conversation terms. Based on which subset the expected sign belongs to, the correct neural network is selected for execution.

Figure 2 demonstrates how information received from the web app informs the logic of selecting the correct model to execute for a given sign, as well as the feature data extracted from the user's sign attempt to input to the network. Executing the model generates a prediction for what sign the user is making. The prediction is sent back to the web app and compared to the expected sign, our system will generate feedback for the user to inform them whether their gesture was correct (the prediction matches the expected sign) or incorrect (the prediction does not match). Within the web application itself, further user data relating to their progress on signs (ratios of correct to incorrect prior attempts) will be tracked and stored within a SQLite database. Various modes of learning will also be provided to the user (practicing signs with versus without instructional help, taking tests, etc.). These details are further explored within our discussion of the system implementation.

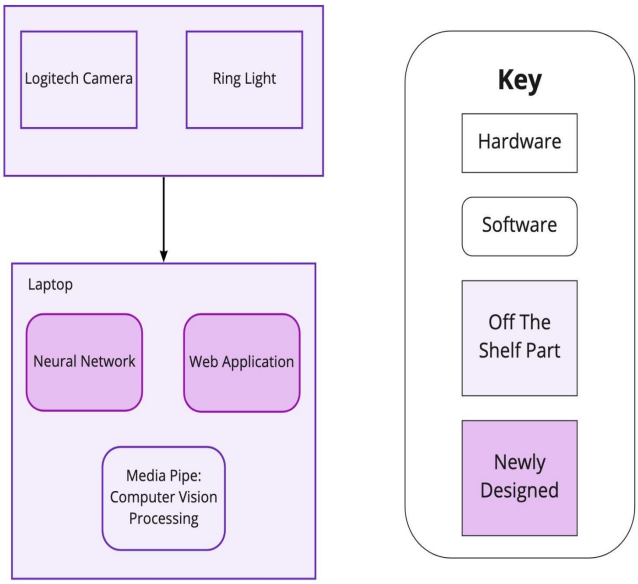


Fig. 1. Solution Approach Diagram

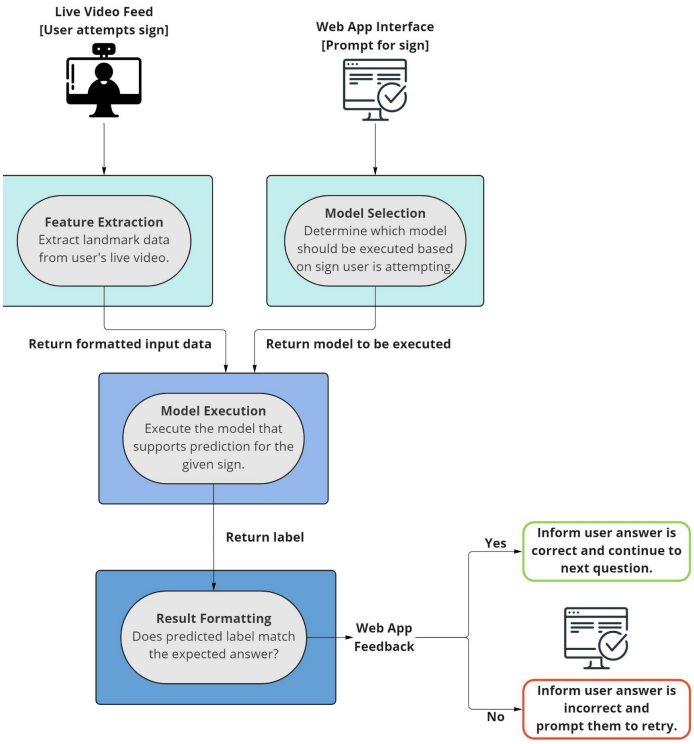


Fig. 2. Software Block Diagram

IV. Design Requirements

One of our design requirements involves the user load that our platform can handle. Because we have decided to keep our system as a local tech stack, we have limitations on how many users can log in at once. From our previous experience, the average number of users that a locally hosted application can handle is 5 users, which aligns with our own experience using local hosting for web applications. However, concurrency is not the main priority of our solution, due to having only one physical system handling the computer vision and the machine learning models. Thus, we are focusing on an individual user's experience, where our solution must be able to perform accurately and efficiently under the load of 1 sole user.

For the machine learning model in our system, we require the input video length from the user to be 5 seconds or less, to maintain uniformity with our training data. This is so that our machine learning model does not process too much data at once, which could inhibit neural network training and cause sign evaluation to slow down. However, we also don't want to have too little data where there isn't enough meaningful information about the sign being demonstrated, hurting our accuracy.

From each frame, we will extract 42 landmarks (21 points from the left hand and the 21 points from the right hand) that we get from MediaPipe. For image data, we must extract landmark data from the image and duplicate the data 10 times with noise applied (a process described further in our system implementation). If some landmarks are missing (due to errors in MediaPipe detection or single-handed signs), the missing landmarks will be padded. Thus, we require our designed

solution to be able to retrieve landmark data for the user's hands, which can either be 42 total landmarks or less than that with padding. Note that we are solely extracting features and data related to the user's hands, despite ASL also relying heavily on emotive facial expressions and signs that touch other parts of the body. The reason for this is due to the additional complexity of dealing with signs that involve contact with the face or body.

For model execution speed capabilities, we are requiring our system latency to be under 2 seconds. This means that for the machine learning model subsystem, the time it takes to process the user image input and output a prediction must be fairly low as well, while also maintaining our accuracy requirements. This will require us to observe tradeoffs between increasing the size of our model to achieve greater accuracy despite that this also increases model execution time. Given our requirement for user feedback latency (time between the completion of a sign and telling the user whether it was correct/incorrect), we must ensure that model execution time is not severely worsened in conjunction with negligible accuracy improvement.

We additionally have some requirements for the datasets we are using to train and test the neural networks, particularly the ones we find online. First, we require that they are correct to standard ASL. Second, we require at least 1,000 images for every static sign, and 5 videos for every dynamic sign from outside sourced data. For static signs, we determined the value based on-course experience in machine learning and researching the data used in similar ASL recognition projects. For dynamic signs, we decided on such a low amount because we saw in our preliminary research that dynamic ASL datasets are very rare, and if they do exist are very small. Thus, in addition to the 5 videos per dynamic sign we find in datasets, we will be creating our own training data so that we have at least 50 additional videos per dynamic sign.

V. Design Trade Studies

A. Hosting Locally vs. Cloud

For our web application, we will have a local tech stack instead of deploying it to the cloud. Deploying it to the cloud will heavily increase the latency in our web application (due to the separation of our system components into different servers that must interact with each other). Because the user is going to be attempting signs in front of a camera from which a video feed is embedded inside of our web app, we are sending the video data input to our computer vision component and machine learning component. If the components were deployed to the cloud, it would increase the latency of our user feedback generation (indicating to the user whether the sign they made was correct or not), as data must be passed between each component and ultimately back to the web app. Hence, we decided having a locally-hosted web application would help in keeping latency down but we are aware that doing this means we won't be able to handle multiple users at once nor have a web domain that the public can visit.

B. LSTM cells vs. GRU cells

Currently, our neural network models will utilize LSTM cells in order to take into account temporal information that comes with the sequencing order of frames within a video. Another type of cell we considered is the GRU (Gated Recurrent Units). LSTM cells have 3 gates: the input gate that stores information in long-term memory, the forget gate that removes information from long-term memory, and the output gate that produces information to share with future time steps [4]. GRU has 2 gates: the update gate which allows a subset of past info to be carried forward and the reset gate which allows a subset of past info to be ignored [4]. Based on past studies, GRUs tend to execute faster due to more simplicity, and LSTMs will be more likely to provide greater prediction accuracy; on the other hand, GRUs may provide better accuracy for smaller datasets [4]. Because we have over 4,000 videos and image data, we ultimately decided to use LSTM layers in order to prioritize prediction accuracy, given that a major component of success for our learning platform is to provide accurate feedback to users as they practice sign language. Moreover, some studies have been inconclusive as to which cell type will always result in better performance, and this result may vary based on the task the models are being trained for [5]. We decided not to use GRU cells due to this uncertainty, as a negligible improvement in model execution latency would not warrant poorer prediction accuracy.

C. MediaPipe and LSTM vs. CNN and LSTM model

An approach we examined for our neural network is to have a CNN and LSTM neural network. This combination is a common approach for image and video classification [5, 6]. We decided not to use a convolutional neural network (CNN), and instead decided to rely on MediaPipe. The main purpose of a CNN prior to the LSTM layer would be to extract meaningful features from the image(s) or video(s). MediaPipe, however, is backed by CNN(s) and completes this feature extraction for us, generating landmark coordinate data from the hands. Therefore, there is no need to do another CNN layer on top of it.

D. LSTM vs. Dynamic LSTM

Another approach we examined was to have a dynamic LSTM neural network [7]. A dynamic LSTM model would help us in grabbing frames dynamically, sending them to MediaPipe, and having the features be sent immediately to the neural network in real-time. So rather than waiting for a certain time frame to end after a user does a sign, they can receive immediate feedback. The main idea behind it is to send the data points into the machine learning model, calculate its LSTM layer for the frame, and then wait until the next frame comes to continue doing the LSTM layer. After getting all the required frames, we will move on to the rest of our machine learning model which should take more than a second to predict the label of the input frames. The main problem we saw from this design was knowing when a sign is complete, and we debated the possibility that if a hand isn't detected in

the frame then the sign is done. In the end, we didn't go with this neural network model mainly because there is no reason, as of now, to need a dynamic LSTM neural network, especially since the majority of our signs are completed within a 3-second window. However, if we want to extend our project further by including phrases in our learning modules, this is a great method to use in the future.

VI. System Implementation

Before exploring the details of our solution approach, we will once more clarify the connection between our web application, our computer vision model, and our machine learning model, as demonstrated in Figure 1. Initially, the video feedback displayed in the web application is sent to the computer vision model backed by MediaPipe, which will generate data points on hands present in the video feed frame by frame. These data points are then fed into our machine learning component to generate a prediction label. This label is sent back to our web application to inform the user if the sign is correct or incorrect.

Data pre-processing and model training is done in a g3s.xlarge EC2 GPU graphics instance with a Deep Learning Amazon Machine Image (AMI) that has Tensorflow pre-installed. We used both data we created as well as datasets found online containing video and image data demonstrating correct American sign language gestures to compose our training, validation, and testing datasets. Each example (video or image) will be assigned a label based on the sign it demonstrates.

A. Computer Vision Component

We use MediaPipe to process and extract landmark data of the hands detected in the videos or images. For each frame that hands are detected, there are 21 landmarks per hand (resulting in a total of 42 landmarks) that will be stored as NumPy arrays. If only a single hand is detected and just 21 landmarks are present, the other 21 must then be padded as zeros. The landmark data is formatted differently for static signs and for dynamic signs. For static signs, inputs consist of landmarks from a single frame/image of the sign gestures. Given that there are 42 landmarks (with three features per landmark), this input is flattened into an array of dimensions (1 x 126). For dynamic signs, the feature data will be composed by extracting landmarks from multiple frames evenly spaced throughout a video input. Our current approach grabs 30 frames from the video data, where there are, again, 42 landmarks (with three features per landmark). Given this format, the data is formatted into an array of dimensions (30 x 126). For both static and dynamic signs, padding is emplaced in the feature data vectors when landmarks are not detected.

For each example, there are three cases of accepted formats: video of a dynamic sign, video of a static sign, and image of a static sign. Other examples that do not fall into these categories will be discarded (since, for example, data from dynamic signs as images are definitely incorrect and thus not appropriate for our model to learn from). In order to create much more image data to supplement the online datasets we use, we decided to create videos of static signs and parsed them into individual frames to be saved as jpg images. This proved to be a better approach than extracting and noisifying landmark data from pre-existing images, and allowed us to generate thousands of static sign image samples efficiently. There is far less availability of sign language video data online than image data. As such, we had to create individual video samples for each dynamic sign in order to have enough training data for our models.

Once formatted for training and testing, the data is stored and fed to the machine learning component of the system to carry out model creation and tuning, as demonstrated in Figure 3. The data was stored on our hard drive to back up, as well as in GitHub repositories, such that it does not occupy excessive space on our devices locally. Once the models were trained, they were integrated into the web app for real-time evaluation.

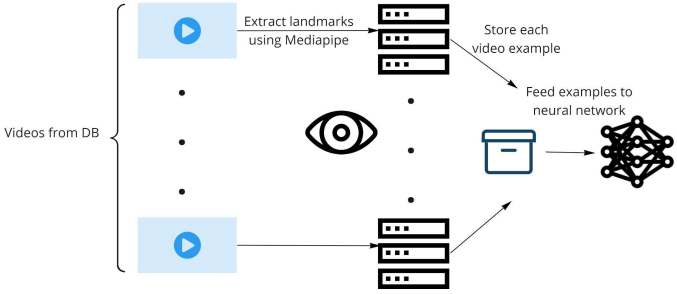


Fig. 3 Feature Extraction Pipeline

The user submits a video or image input for their attempt at a given sign specified by a prompt in the web app interface and the landmark data from their input will be collected in the same manner as described above for static and dynamic signs respectively. Once a batch of landmark data is collected, this will be passed to the machine learning component of the system to evaluate and assign a prediction label.

B. Machine Learning Component

The models are instantiated, trained, and saved using Tensorflow on an EC2 instance. As per Figure 4, the model structure for static signs is created as a Sequential model instance with five dense layers, as well as two dropout layers to help mitigate overfitting. As per Figure 5, the model structure for dynamic signs is also created as a Sequential model instance, but instead with three LSTM layers followed by three dense layers [12]. The LSTM layers allow the model to take temporal information (ordered sequencing of the video data) into account for classification. To do this, LSTM cells propagate information forward as well as to each other (older time steps inform future time steps). Further, the model conducts categorical classification, where the output generated is a set of probabilities indicating the likelihood of each possible label being the correct prediction. We then use argmax to select the highest probability, and the predicted sign is the label associated with this probability.

The models are trained and saved using the Tensorflow API and Keras API. For static signs, the flattened array of

landmarks for a given frame is fed to the model's input layer. For dynamic signs, each of the frames for a given example is treated as a time step fed in at each node of the model's input layer. This training is done through the Tensorflow model.fit() method, whereas testing and validation metrics are generated through the model.evaluate() method, which allows us to observe the model prediction accuracy. The method model.save() allows us to preserve the model's weights and structure after training is complete. The method keras.models.load_model() allows us to load up this saved model format for evaluation of signs in the web application. We trained our models and observed their training and testing accuracy every 10 epochs (beginning with 10 epochs, up to and including 1000 epochs).

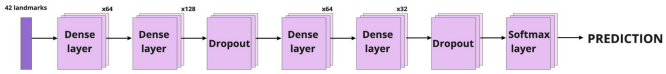


Fig. 4 Static Model Structure

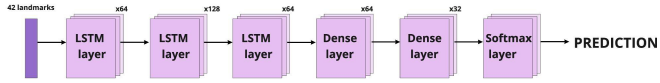


Fig. 5 Dynamic Model Structure

C. Web Application

We are using JavaScript and HTML to create the web application component of the ASLearn platform. Users will be able to log in so that their ASL curriculum progress can be stored in a database in association with their account. With this, they can see which signs they have tested on and the percentage of times they have done the sign correctly. The pages of the web app will consist of login, registration, homepage, courses, learning mode, testing mode, and a profile page. Because we want to create a web application that allows for real-time responsiveness, we are utilizing AJAX so that the user can receive feedback on their sign without reloading the page.

The home page will show multiple courses available for the user to take, each of which will be a sign language topic such as the alphabet, numbers, conversation, etc as seen in Figure 12. If a user clicks a sign language topic on the home page, like the alphabet, they will be guided to the specific course page for that topic. Inside the course page, the user will see the individual modules they can access as seen in Figure 13. For example, if a user is viewing the alphabet course page, the individual modules they would see will be A, B, C, etc. Each individual module would have learning and testing mode options for the user to select. The pages for learning and testing will be similar in format. As seen in Figure 14, the testing page will give an indication to attempt a certain sign and will have real-time video feedback embedded into the page so the user can see themselves doing the sign. On the other hand, the learning page is going to have real-time video feedback, an instructional demo video showing how to do the sign correctly, and text blurbs giving hints to the user on how to do the sign correctly as seen in Figure 15. We are also going to have a separate course module called testing where a user is prompted to pick the topics they want to be tested on like in Figure 16. They will be given 10 random questions, depending on the topics they chose and will be shown a testing page for each of the questions. In the end, they would be shown their results as seen in Figure 17. Lastly, the user has a profile page where they can see how they have done for each individual sign that they have tested on as seen in Figure 18. For example, if a user has 50% this means that out of the two times they have been tested on the sign, they have only gotten it right once. The purpose of this page is to motivate users to practice the signs that they are still having trouble with.

For both the testing and training pages, we are calling on our machine learning model using JavaScript. As the user attempts a sign, the real-time video feedback of their attempt is captured and sent to the machine learning model. Apart from the video feedback being sent to the computer vision model, the web application would also be in charge of sending an indication of what sign the user is testing/training on. We are sending a string of what the expected sign is (e.g., 'A', 'B', etc.), to the machine learning model to help it select the correct neural network it needs to make a prediction for the user's attempt.

The computer vision model will receive the real-time video feedback to format as landmark data that can be fed to the machine learning model's corresponding neural network, which will generate a predicted label for what the sign is identified as (e.g. 'A', 'B', 'C', etc.). This label is sent back to our web application, where we are going to check whether the predicted label we received is the same one as the one we sent to the machine learning model. The results we receive from this comparison will be displayed to the user in the web app. If the user's sign is correct, the boundary of the real-time video feedback will light up as green. If incorrect, the boundary will glow red. We also have a message box at the bottom to inform users if they were correct or not as well. Having the boundary of the real-time video feedback box light up will capture the user's attention and make these correct results clear. However, we do have the message box as well as an additional indicator of sign correctness.

As for how the computer vision model is receiving video feedback of the user's attempt at a sign, a 5-second timer will indicate to the user when they should start their sign and how much time they have left to make the sign. The timer is going to start once the user clicks a button on the page indicating that they want to begin attempting their sign. This eliminates the problem of needing to know when the user makes the sign and wants it to be checked for correctness. The timer is beneficial to both our computer vision and machine learning models since it helps us restrict how many frames we are sending to the models. The user can choose to either let the timer run out for the submission to finish recording or they can

press the stop recording button once they have finished their sign submission.

When doing the web application component, some problems did arise with respect to the integration between the web application and the machine learning component. Our original idea was to record the user's sign submission and send this to the machine learning component, but this ended up being extremely complicated especially since our recording was a Blob object. Therefore, we decided to start real-time predicting once the 'Start Recording' button has been pressed. The recording button ended up giving us the opportunity to know when a prediction starts. We also had trouble saving the data with respect to a user for each individual test that they made, so we ended up creating a model called Test where it created a new test object, saving all of the questions and answers, and linking it to the current user that's logged into the account.

VII. Test, Verification and Validation

In regard to testing, we have broken down our plan into subsystems and overall system tests. This is to ensure that the tools we are using match our needs and that the ASLearn platform we create is aligned with our intended use-case and specifications.

For subsystem tests, we examined the Hands library data from MediaPipe, where we verified that when hands are in frame and moving with signs, they are detected with 21 landmarks each as specified in the MediaPipe documentation. We ensured that this works both with live video feed and when given pre-existing video or image data. In terms of the data format, the 21 landmarks were verified to have x,y,z coordinates associated with them, both in MediaPipe demos and in our own experience testing the Hands library. Finally, we verified the correctness of testing data by referring to online ASL guides and relying on Hinna's expertise given that she took an ASL course last semester. Similarly, for online datasets–see Table III at the end of this report–we verified the correctness and quality of the ASL sign data by using reputable sources and manually looking at examples from the dataset.

As for ASLearn platform testing, we conducted tests based on the use-case requirements detailed in section II of this paper. To reiterate what these requirements are, we tested the user distance from the camera, the overall platform latency, the accuracy of the platform sign detection, the ability to handle left/right-hand dominance based on user preference, and user satisfaction with the web application user interface.

A. Tests for User Distance from Camera

For the user distance from the camera, which we specified to be within 3 feet, we conducted signs at various distances (i.e. 1.5 feet, 2 feet. 2.5 feet, and 3 feet) and check how well ASLearn is able to determine if the sign is correct or not. A passing test will maintain prediction accuracy in determining sign correctness at any distance within and including 3 feet. If our tests for this requirement fail, we direct the user to be within whatever distance we have determined to be successful and also use a bounding box to hone in on the user's hand motion.

B. Tests for Platform Latency

A second set of testing was conducted in regard to the platform latency, which we have decided should be within 2 seconds. This latency requirement is specifically for how long it takes our model to predict the correctness of the sign and the web application to display feedback responsiveness of the ASLearn platform itself. In order to test this, we timed how long it takes for the site to give the user correct feedback, starting from the point at which the user submits their sign, which should take no longer than 5 seconds, as explained in Section III of this report. If we found that our latency consistently exceeds 2 seconds, we had the contingency plan of adjusting our prediction generation algorithm to be a faster but potentially less accurate method; for instance, using a CNN for faster execution and reduced data formatting complexity. Using a CNN would simply take in image data at the input layer, though it may pose lower accuracy due to interference from extraneous colored pixel data surrounding the hands performing sign language.

Figure 6 demonstrates our analysis of model latency variation for dynamic signs as we increased the number of frames we rely on for formatting the feature data of each video sample. For our system, we decided to use 30 frames, as this allowed us to achieve lower model latency while not undermining accuracy too much. We were able to achieve a maximum model testing accuracy of about 94% for both dynamic models (conversation and learning) at this 30 frame metric.

In testing our system after integration, we found web app latency (for submitting a sign and formatting user feedback) is less than 0.5 seconds. This includes the latency for model execution and latency for formatting and sending a request to the model execution server and receiving a prediction response. Model execution time involves feeding an input to the neural network and generating a prediction at the output layer, which is less than about 0.18 seconds. The latency for sending a post request involves formatting the user's feature extracted data as part of an HTTP POST request, creating the request, sending it, and then receiving the response containing the prediction generated by the model selected for execution. These steps together make for a latency of about 0.3 seconds. Overall, the 0.5 second latency is well below our initial requirement of 2 seconds.

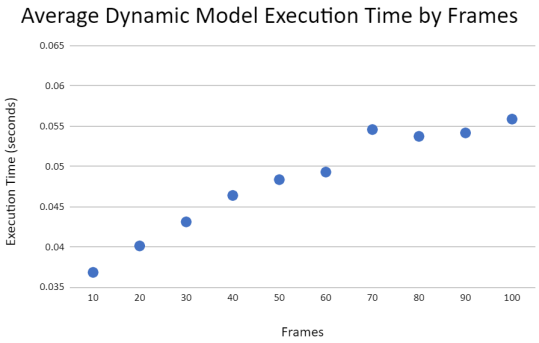


Fig. 6 Average Model Execution Time

C. Tests for ASLearn Accuracy

As previously mentioned, we have specified that the ASLearn platform should be 97% accurate when determining the correctness of user sign. As such, we observed the training and testing accuracy our models were able to achieve across various training epochs. Figure 7 demonstrates how all of the models achieved almost 99% training accuracy, which verifies that the models are indeed learning from the data. Figure 8 demonstrates our initially found model testing accuracies (generated by evaluating the models on data they were not initially trained with). The dynamic sign models both achieved about 94% accuracy, and the fist model achieved about 93%. On the other hand, the remaining static sign models still had room for improvement. We observed that the low accuracy may be due to the models overfitting to the data found through online sources. For some signs, the data was created by one person, and often in the same lighting for most samples. Thus, we created more of our own static sign image data to add to the training set in order to improve the testing accuracy of the one-finger, two-finger, three-finger, and open-hand models. We created these images in various lightings (i.e. low light, natural light, studio light, etc.). Figure 9 demonstrates the improved testing accuracies achieved for each of these models, where each had an improvement of at least 8.5% to at most a 22% for the one-finger model, as observed in Figure 10. Overall, creating and adding more of our own data added more diversity of examples to the data set we were relying on from online sources. The one-finger model may have experienced the highest improvement due to its slightly lower physical complexity in comparison to the other multi-finger signs. Given that we did not quite achieve a 97% accuracy for each of the models, we decided to take additional measures to improve accuracy during real-time performance by communicating to the user that they should be in a well-lit space and have little to no impediments on their hands (e.g. bracelets, rings, etc.). Moreover, it seems that training accuracy defines a limitation on how much testing accuracy we are able to achieve. We cannot simply keep training for more epochs in hopes to improve test accuracy, as once the models hit almost 100% training accuracy consistently, this can indicate overfitting, which is difficult to mitigate.

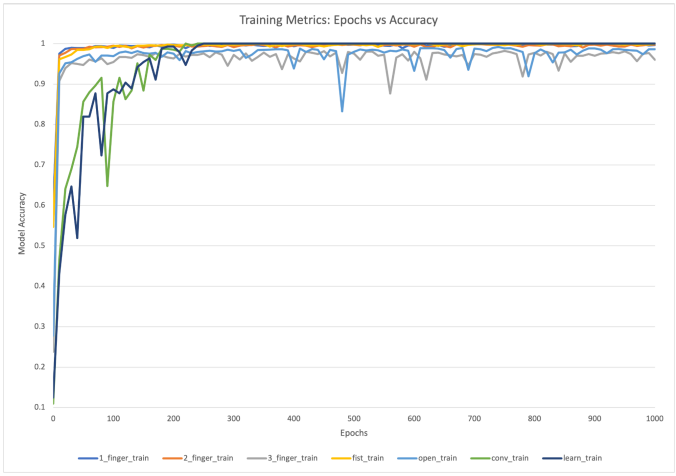


Fig. 7 Model Training Accuracy

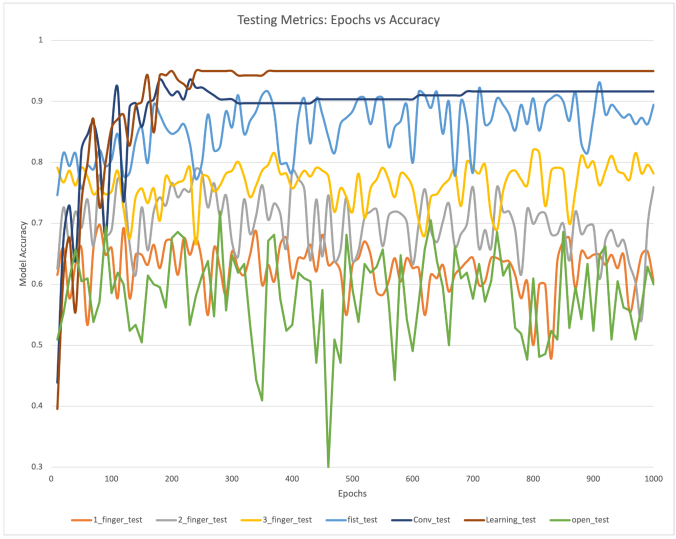


Fig. 8 Initial Model Testing Accuracies

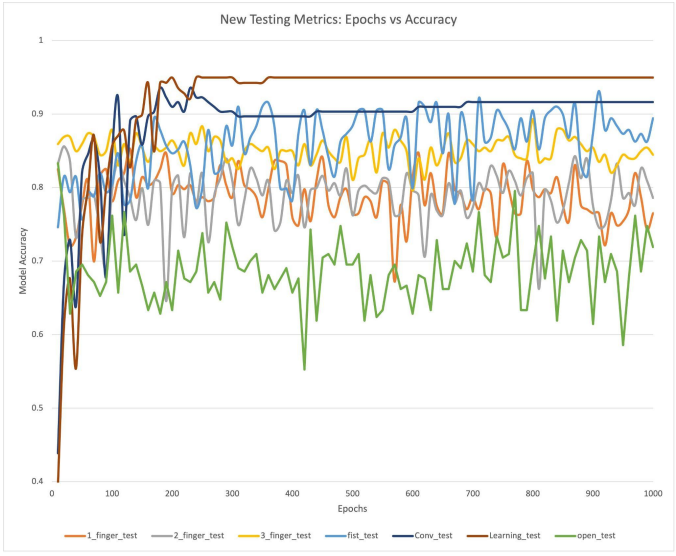


Fig. 9 Improved Model Testing Accuracies

	Max Test Accuracy Before	Max Test Accuracy After	Percent Improvement
one-finger	69.79%	85.25%	22.15%
two-finger	78.93%	85.62%	8.48%
three-finger	82.04%	89.32%	8.87%
open-hand	71.90%	79.52%	10.60%

Fig. 10 Maximum Achieved Model Test Accuracy Before and After Addition of Training Data

D. Tests for Left or Right Handed Signs

An additional aspect of ASLearn that will need to be tested is accuracy based on whether the user is right or left-handed. To test this we will do signs on the platform using our right hands, and then again using our left hands, where we expect that the model remains at least 97% accurate regardless of which hand is used. However, if this test fails, we will communicate that our platform is currently only suited for right-handed signs, given that the majority of data available online is right-handed and that all our group members are right-handed.

E. Tests for User Satisfaction

Finally, we conducted user satisfaction surveys after having users test the ASLearn platform based on certain usability metrics. We had users in the later weeks of the semester rank 4 features of ASLearn, as described in section II, - from 1 to 5. Overall, we planned to have at least 90% user satisfaction based on the industry standard, where we aimed for a total score of 90/100 across all surveys. However, the survey results were lower than expected, with most of the complaints being about the accuracy of the static signs and not being graded correctly. With the time that we had left in the semester when we conducted the surveys, we decided to add more training data for the static signs to improve the overall experience. Another comment left from one of the users was to add a profile page to our web application which we ended up doing to show the user their progress in learning a sign. Unfortunately, we were unable to conduct a second round of user satisfaction surveys to see how the changes have affected the user experience.

VIII. Project Management

Given the time constraints and complexity of this project, good project management will be essential to keeping our team on track for a successful demo at the end of the semester. We have assigned Hinna to be our group's project manager, with Valeria in charge of note-taking at meetings, and Aishwarya in charge of updating WordPress with team milestones.

A. Schedule

To manage the time constraints of the semester, we have created a detailed Gantt chart, which can be found at the end of this report in Figure 19, to plan out tasks for every week of the semester. The chart is color-coded based on the team member who is primarily responsible for each task, with various tasks involving all team members.

In referring to Figure 19, our schedule did not change drastically from our design document to the final project report. The only major change was adding more weeks of training our neural network. By training, we mean creating more training data for our static and dynamic signs to improve the accuracy of our seven neural network models.

B. Team Member Responsibilities

In regard to team member responsibilities, we have divided the labor among our three group members with the more intensive tasks involving effort from everyone. Aishwarya was mainly responsible for setting up the initial deep learning model, working with the AWS EC2 instance, embedding the camera feed into the web application, and conducting distance tests. Hinna was responsible for making the ASLearn instructional materials, verifying the correctness of testing and training data, as well as testing latency and left/right-handedness. Valeria did the web application UI design, the deep learning model with Aishwarya, and the testing of the lighting/environmental requirements. Tasks that all three of us worked on include fine-tuning the deep learning model, making testing data for the model, integrating the web application component and the machine learning component, and conducting user tests.

C. Bill of Materials and Budget

Refer to Table II for a detailed list of the tools we plan to utilize for our design and the purpose each component will have. Refer to Table III for a list of the open-source datasets we plan to use for model training, validation, and testing.

D. AWS Usage

We requested 150 credits for AWS. Our initial idea was that we would use these credits to help train our neural networks. Instead, due to time constraints, we decided to only focus on training our neural networks. Since our training is data-heavy, most of our usage went into parsing our video and image data in addition to training the Tensorflow models. We ended up using 42.92 credits in total. After our initial training of the neural networks, we decided to stop using AWS. There wasn't that much of a difference between how much time it took to train locally versus in AWS. Furthermore, it took a long time to just move our data locally into AWS. Thank you to Amazon for allowing us to use their platform to help train our machine learning models.

E. Risk Management

The main risk of our project was that there was no data

available online on the dynamic signs that we wanted to include in our project. Because of the lack of data online, we decided to create our own training and testing datasets for all the dynamic signs. Instead of creating a Python program that iterates through YouTube to find videos of the signs we want, we ended up creating the data ourselves. We made over 70 videos for each of the 15 dynamic signs. These videos were five seconds long and the signs were done at different angles to account for user error when they are testing themselves. Apart from creating our own dynamic sign language dataset, we also created our own static sign language data. As we continued training our models with online datasets, we realized that some of these datasets were incorrectly doing the signs. Because of this, we ended up having to scrap most of the online data that we found and create it ourselves. We made the decision to make the data ourselves because we knew we would be doing the signs consistent with the ASL standard. The total amount of static data we created was over 100 images for each of the static signs.

IX. Ethical Issues

In examining the potential impacts of our project, a few ethical issues can be raised, with two main broad groups that are most vulnerable if our platform fails or is misapplied: people with physical or learning impediments and those in the hard of hearing community.

The first group is vulnerable because our platform requires users to be able to do signs that they learn relatively quickly in our 'testing' mode, which provides engagement and interactivity despite it being a remote, online learning platform. However, if someone has a physical disability particularly involving the motion of their hands or a learning disability that results in slower understanding, our testing mode might be incredibly frustrating or even impossible to use, leaving such users out of the equation. However, these users can still use our 'learning mode' and live embedded video to learn new ASL terms and see themselves attempt to sign these terms side by side with the instructional video.

Additionally, some of these issues center around the fact that our project aims to teach people in the hearing community how to communicate with those in the hard-of-hearing community. So, if our platform doesn't meet the ASL standard when we are teaching users different signs or if the process is so frustrating that users decide to stop trying to learn ASL - through our platform or at all. The consequences of this and of our platform failing, in general, most deeply affect the hard of hearing community. Our platform aims to help hearing people learn how to communicate with deaf people, thus any failure in the platform would result in those in the hard of hearing community to continue bearing the brunt of bridging the communication gap with hearing people through lip reading, text communication, and doing non-sign language gestures.

Another ethical aspect of our project to consider involves the data we used to train and test our models. An ideal dataset would have high resolution 3-5 second videos of all 51 signs we are doing, with at least 100 iterations of each in both left and right-handed signs, and contain a variety of different signers (i.e. different skin tones, hand sizes, environmental factors, etc). Currently, we are using a variety of datasets as the 51 signs are not all contained in one. We have a dataset for the alphabet, the digits (0-9), and some communicative, dynamic signs. Particularly for the communicative signs, we have nowhere near 100 iterations and the dataset video quality is fairly inconsistent, but there are a variety of signers. For the letter and numbers datasets, there is less variation in signers but there are at least 100 iterations of each sign.

To mitigate these problems, a potential solution is to have more people help us create our training data over time, with us putting a hard focus on having a variety of hand sizes and skin tones. Creating more data would help us have an algorithm that isn't biased toward a specific skin tone or hand size allowing more users to be able to use our platform and get correct feedback. We also want this data to be in environments where the light isn't perfect so that users can use this platform anywhere.

As for our users with physical or learning impediments, we can create a special mode for these users. In this special mode, testing will be more lenient in regards to the timing of the video submission, giving users 15 seconds rather than the 5 seconds limit. Another possibility is to not have a timeout running and instead give the users the ability to stop the recording once they are done. This way, the users are not constricted to doing the sign in 5 seconds which can be tough to do for someone with impediments.

Hence, if this project were to be deployed, we would need to work on these ethical issues before it is free to the public. As a platform that is meant to help bridge the gap between the hard of hearing community and the hearing community, these issues need to be fixed. We also still need to consider the limitations of American Sign Language data if we want to add more topics to the lesson plans.

X. Related Work

There are existing examples of utilizing TensorFlow to create neural network models and MediaPipe to generate landmark coordinate data from video or image inputs of sign language or other tasks. In one specific example involving ASL identification for the letters J and Z, the TensorFlow models were able to achieve at least 96% training accuracy using 612 videos and 100% training accuracy on 91 videos. This poses promising results for our implementation, as it sets a precedent of successful sign identification with a similar model structure (LSTM layers and dense layers) and set of tools [8].

Another example utilizes an LSTM followed by dense layer structure to train a model on time sequenced data recognition [9]. Overall, this is a standard approach used for the neural network structure when carrying out recognition tasks where temporal data also informs the resultant classification. In another previous work, comparison between different model structures/approaches was also conducted, where using LSTM

layers demonstrated better accuracy results (at least 86%) compared to other machine learning approaches such as Hidden Markov models. The project further utilized MediaPipe and TensorFlow to extract landmark data and create the models [10].

XI. Summary

ASLearn holds the potential to foster better inclusivity for those who are part of the hard-of-hearing community. Making ASL curriculum and access to quick, correct feedback easily accessible will hopefully also encourage people to learn ASL when they may not have considered it to be feasible before.

A. Overall System Performance

Our system was able to meet the design specifications we set for ourselves at the beginning of the semester. While we do have a fully functional, locally-hosted platform that can detect sign correctness for static and dynamic signs, there are some limitations to it. The first limitation is the user must use the platform in places with good lighting and have a good camera in order for the model accuracy to be at its highest. The user should also be within 3 feet of the camera to use the feedback system and sign into it. Apart from that, the model accuracy is averaging 80% among all the neural network models. From the testing that we have done, the platform tends to detect sign submission to be incorrect even when it isn't, making it frustrating for the user. In order to improve this, more training data needs to be added in. As more training data gets added, the model accuracy improves.

B. Future Work

If we continue working on this project, our main priority would be to add facial recognition. Facial recognition is important in ASL since it's a way for people in the community to show their feelings and be able to express themselves. Adding facial recognition would consist of redoing all of our data, specifically our data for dynamic signs, to add landmarks of the face present in the video. These landmarks we can probably also get from MediaPipe using the facial recognition feature that they already have.

Apart from that, another aspect that we can add to our project is checking if the dominant hand is doing the primary motion for two-handed dynamic signs. Making sure this occurs is imminent to correctly do these two-handed dynamic signs. We are also thinking about deploying the web application to the cloud so that more users have the ability to use our platform. By deploying on the cloud, this means that we would have a domain name that users can visit and it also means that we would be able to handle multiple users at once. Finally, adding more dynamic signs is something that we consider doing for the future so that users can learn how to do phrases like "Where's the bathroom?".

C. Lessons Learned

Throughout this semester, we learned that scheduling and having a plan is extremely important. We found that revisiting our schedule every week, and seeing where our progress currently stands, helped tremendously in prioritizing tasks for the next week. We also found that starting early, from figuring out what project we wanted to complete to starting the presentation slides, is the best way to go about completing such a huge project. Because this project had two main separate components, communication and documentation were key to making the integration between these two components work. Furthermore, one of our main lessons is to vet the data you are using before training your neural network on it. Ideally, the datasets should be diverse and comprehensive. But regardless of that, the main priority with the datasets is to vet them to make sure that the datasets are up to standard before training.

Glossary of Acronyms

AJAX - Asynchronous JavaScript and XML
AMI - Amazon Machine Image
ASL – American Sign Language
CNN - Convolutional Neural Network
EC2 - Elastic Compute Cloud
GRU - Gated Recurrent Unit
HTML - Hypertext Markup Language
LSTM - Long Short-Term Memory
UI - User Interface

References

[1] "The Ergonomic Equation." Make Ergonomics Simple: Tips for Adding Ergonomics to your Computing. Ergotron. Accessed March 4, 2022. https://www.ergotron.com/en-us/ergonomics/ergonomic-equation#:~:text=Position%20the%20monitor%20at%20least,larger%2C%20add%20more%20viewing%20distance.

[2] "Understanding Latency - Web Performance ." Web Performance | MDN, February 18, 2022. https://developer.mozilla.org/en-US/docs/Web/Performance/Understanding_latency.

[3] *U.S. Overall Customer Satisfaction*. (n.d.). The American Customer Satisfaction Index. Retrieved May 7, 2022, from https://www.theacsi.org/the-acsi-difference/us-overall-customer-satisfaction/

[4] Yang, Shudong, Xueying Yu, and Ying Zhou. "LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example." IEEE Xplore, 2018. https://ieeexplore.ieee.org/document/9221727.

[5] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." arXiv, December 11, 2014. https://arxiv.org/pdf/1412.3555v1.pdf.

[6] Paul, Sayak. "Keras Documentation: Video Classification with a CNN-RNN Architecture." Keras, June 5, 2021. https://keras.io/examples/vision/video_classification/.

[7] Rosebrock, Adrian. "Video Classification with Keras and Deep Learning." PyImageSearch, July 15, 2019. https://pyimagesearch.com/2019/07/15/video-classification-with-keras-and-deep-learning/.

[8] Damien, Aymeric. "Dynamic Recurrent Neural Network (LSTM)." WizardForcel. Accessed March 4, 2022. https://wizardforcel.gitbooks.io/tensorflow-examples-aymericdamien/content/3.08_dynamic_rnn.html..

[9] Pandurangan, G. (R. (2021, July 12). *New Machine Learning (ML) approaches for American sign language (ASL) recognition*. LinkedIn. Retrieved March 4, 2022, from

https://www.linkedin.com/pulse/new-machine-learning-ml-approaches-american-sign-asl-pandurangan-/
[10] *Recurrent neural networks (RNN) with Keras : Tensorflow Core*. TensorFlow. (2022, January 10). Retrieved March 4, 2022, from https://www.tensorflow.org/guide/keras/rnn
[11] Domènech, A. (2020, July 28). *ASL recognition in real time with RNN*. upcommons.upc.edu. Retrieved March 4, 2022, from https://upcommons.upc.edu/bitstream/handle/2117/343984/ASL%20recognition%20in%20real%20time%20with%20RNN%20-%20Antonio%20Dom%C3%A8nech.pdf?sequence=1&isAllowed=y
[12] Renotte, Nicholas. (2021, June). ActionDetectionforSignLanguage. https://github.com/nicknochnack/ActionDetectionforSignLanguage
[13] ASL Alphabet. (2018). https://www.kaggle.com/grassknoted/asl-alphabet
[14] Khalid, Muhammad. (2020) Sign Language for Numbers. https://www.kaggle.com/muhammadkhalid/sign-language-for-numbers
[15] Li, Dongxu and Rodriguez, Cristian and Yu, Xin and Li, Hongdong. (2020) "Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison". https://dxli94.github.io/WLASL/

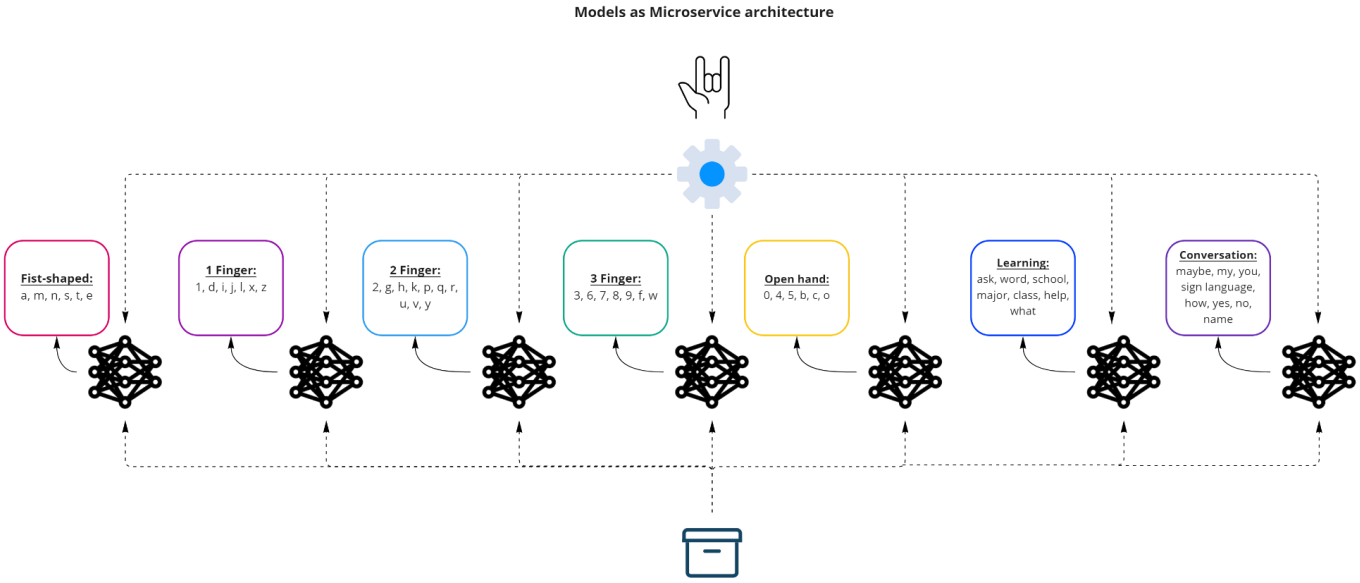


Fig. 11. Neural Networks as Microservice Architecture

TABLE II. BILL OF MATERIALS

Material	**Model Number**	**Price**	**Manufacturer**	**Source**	**Description**
Camera	Logitech Pro HD Webcam C920	$68.49	Logitech	Amazon	We will be using this camera to act as the web camera for the ASL platform. Using an external camera than the laptop builtin camera will allow for a higher quality video feed and consistency across our testing data.
External Drive	Toshiba Canvio Basics 1TB Portable External Hard Drive USB 3.0	$44.52	Toshiba	Amazon	We will be using this external drive to save all of our training and testing datasets.
Ring Light	Video Conference Lighting Kit 3200k-6500 K	$20.99	Selfila	Amazon	We will be using this ring light to help with our lightning situation as a risk mitigation if we see that doing signs in normal lightning makes it difficult for our computer vision model to extract the points.
AWS EC2 Instance	P3.2xlarge instance	$3.06 per hour	Amazon AWS	Amazon AWS	We will complete model training on an AWS cloud instance that contains a NVIDIA Tesla V100 GPU.
Laptop	N/A	$0	N/A	N/A	Our laptops will be used to create the deep learning model, the web application, and to host the ASLearn platform.
MediaPipe	Hands Library	$0	Google	Google	MediaPipe will be used to extract hand landmarks for user signs.
MediaPipe	MediaSequence	$0	Google	Google	MediaSequence will be used to extract frame data from sign language videos.
Python	TensorFlow, Numpy	$0	Python	Python	We will be using Python for our machine learning model, specifically using TensorFlow to create, train, and save our neutral networks and NumPy to format data from MediaPipe that will be fed into the neural network.
Web application	JavaScript, AJAX	$0	JavaScript	JavaScript	We will be using JavaScript to make our html templates and using AJAX to deal with the basic functionality of the webapp for switching pages, watching instructional demo videos, showing real-time video feedback, and sending this feedback into our computer vision model.

TABLE III. TRAINING DATASETS

Dataset	**Link**	**Signs in set**	**# of data per sign**	**Augmentation? Y/N**
Word-Level Deep Sign Language Recognition (WLASL)	https://dxli94.github.io/WLASL/	Dynamic signs	~15	Yes
ASL Alphabet (Kaggle)	https://www.kaggle.com/grassknoted/asl-alphabet	Letters from alphabet	~3000	No
ASL Numbers (Kaggle)	https://www.kaggle.com/muhammadkhalid/sign-language-for-numbers	Numbers (0-9)	~1500	No

Fig. 12. Web Application: Home Page

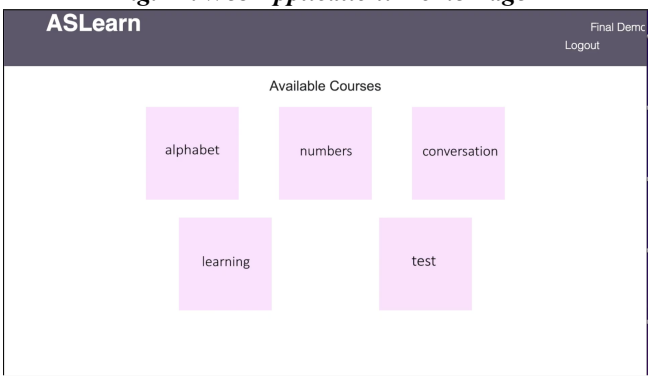


Fig. 13. Web Application: Course Page

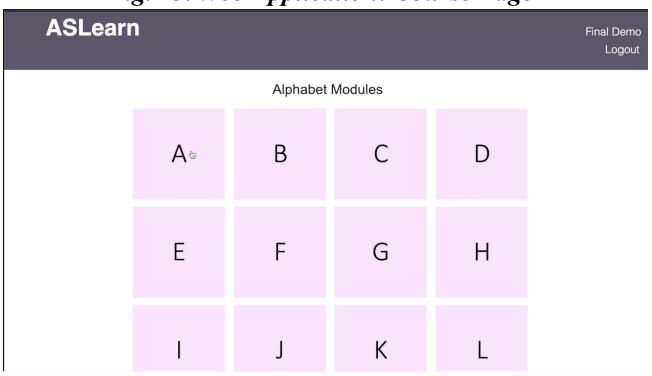


Fig. 14. Web Application: Testing Page

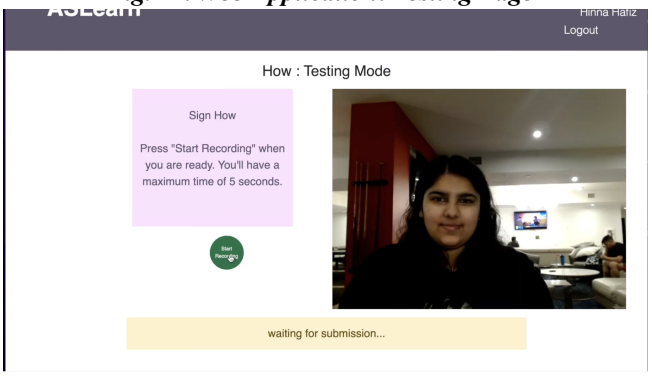


Fig. 15. Web Application: Learning Page

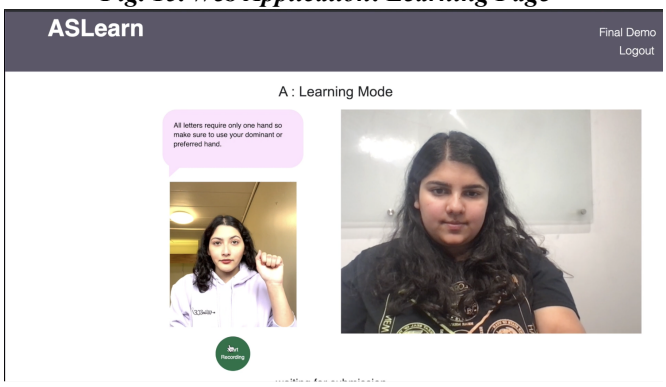


Fig. 16. Web Application: Picking Topics Page

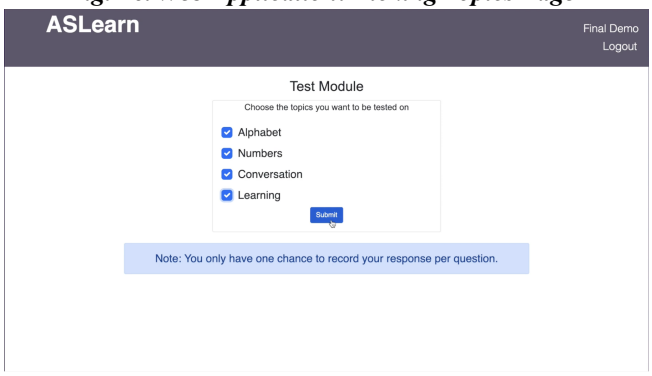


Fig. 17. Web Application: Results from Quiz Page

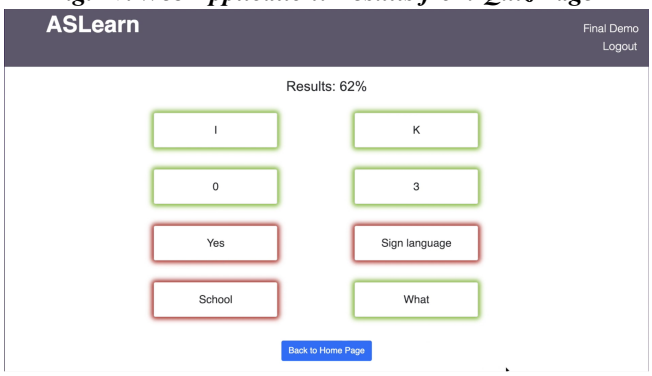


Fig. 18. Web Application: Profile Page

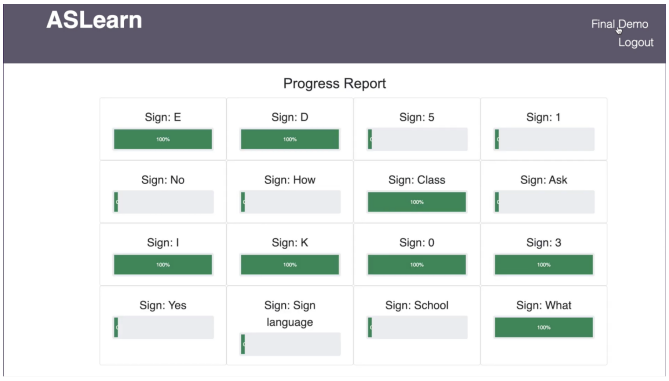
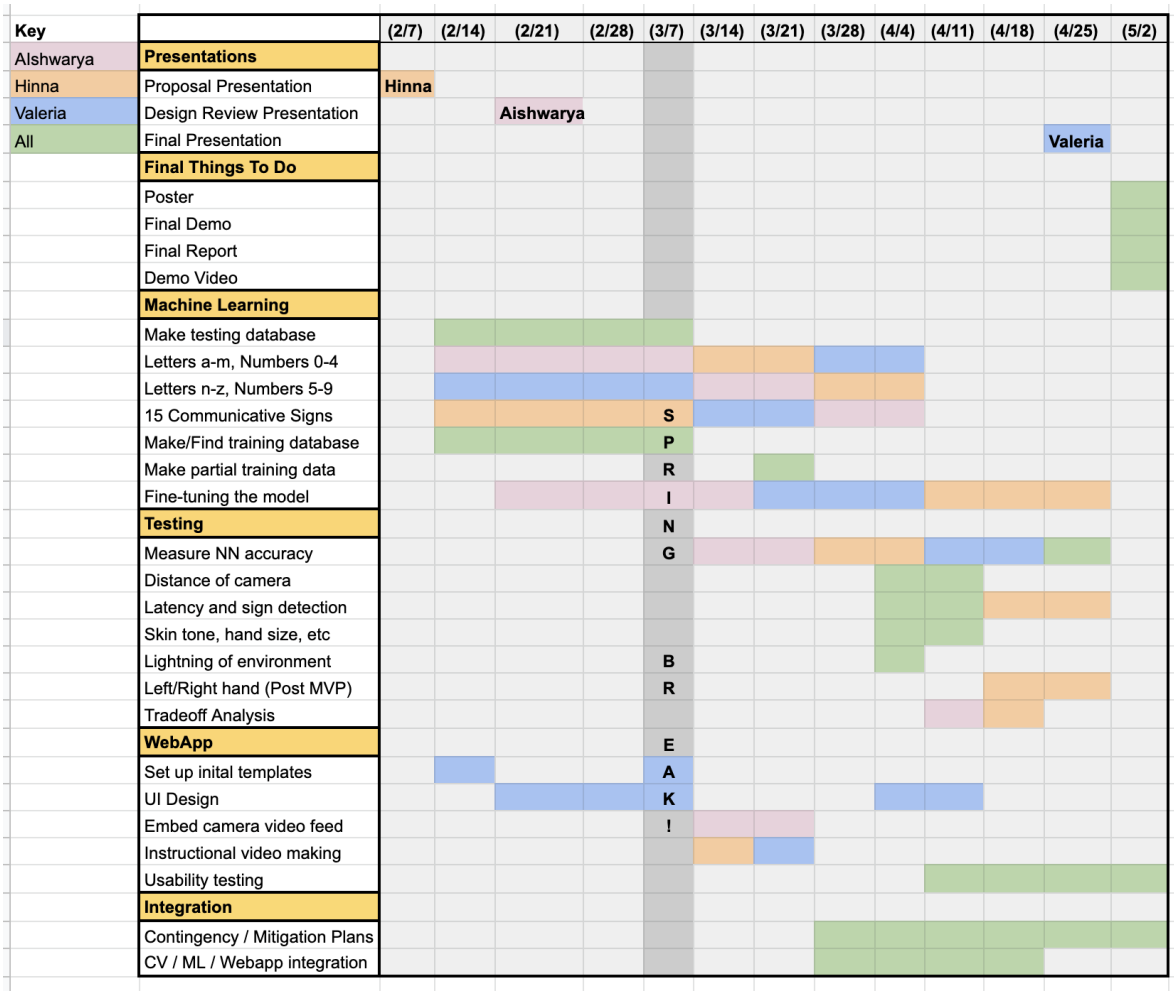


Fig. 19. Project Management Schedule



Key		(2/7)	(2/14)	(2/21)	(2/28)	(3/7)	(3/14)	(3/21)	(3/28)	(4/4)	(4/11)	(4/18)	(4/25)	(5/2)
Alshwarya	**Presentations**													
Hinna	Proposal Presentation	**Hinna**												
Valeria	Design Review Presentation			**Aishwarya**										
All	Final Presentation												**Valeria**	
	Final Things To Do													
	Poster													
	Final Demo													
	Final Report													
	Demo Video													
	Machine Learning													
	Make testing database													
	Letters a-m, Numbers 0-4													
	Letters n-z, Numbers 5-9													
	15 Communicative Signs					**S**								
	Make/Find training database					**P**								
	Make partial training data					**R**								
	Fine-tuning the model					**I**								
	Testing					**N**								
	Measure NN accuracy					**G**								
	Distance of camera													
	Latency and sign detection													
	Skin tone, hand size, etc													
	Lightning of environment					**B**								
	Left/Right hand (Post MVP)					**R**								
	Tradeoff Analysis													
	WebApp					**E**								
	Set up inital templates					**A**								
	UI Design					**K**								
	Embed camera video feed					**!**								
	Instructional video making													
	Usability testing													
	Integration													
	Contingency / Mitigation Plans													
	CV / ML / Webapp integration													