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 American Sign Language. 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.**

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

I. Introduction

Hundreds of thousands of people within the U.S. alone rely on American Sign Language (ASL) to communicate as a result of hearing loss. Encouragement 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 specifically 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 schedule.

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 this video feed side-by-side with an instructional video and prompt. Another feature, '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, and their performance is recorded in the background (number of correct/incorrect attempts). 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.

II. Use-Case Requirements

We defined specifications around computer vision of chosen signs, accuracy of sign labeling, user distance from the camera, 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 will detect user signs after a user is prompted to attempt a specific sign; thus, we will be able to compare the predicted label assigned to the user's attempt to the expected label requested in the prompt. Because our model will be 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 incorrect predictions as a result of human error or environmental impediments that may occur as the user attempts signs.

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. An additional foot of distance is added in case the user moves further away to make sure both their hands are in frame.

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 a latency of 2 seconds. This is separate from the responsiveness of the web application's general interactive features. Latency exceeding this 2-second requirement may make the platform frustrating for users. To maintain convenience, we want the latency of user feedback generation to not be much longer than feedback from an in-person instructor. On the other hand, a shorter latency requirement than 2 seconds may underestimate the amount of time needed to process input data and execute a model with said data to generate a label. Additionally, while we found that our model initially has between 0.07-0.12 seconds of latency based on some experimental data, we anticipate that once the model has been tuned for improved accuracy, there will be significantly more latency due to greater complexity of the model structure.

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 plan to conduct user surveys towards the end of the semester with about 10 users and have them rate aspects of the platform from 1-10. Our goal for this requirement is to have 90% user satisfaction in regard to navigating and understanding the site, as that is industry standard for user testing. To measure whether we achieve this 90% user satisfaction, we will give a 20 question survey to all users who try our web application during the testing phase. Currently, we are planning on testing 10 users, 5 of whom are experts in the ASL field and the other 5 are beginners taking the ASL stuco course at CMU. This survey will ask questions such as "Was the feedback you received quick?", "Did you understand the feedback?", and "How difficult was it to navigate through the pages?". The answers to these questions will be ratings on a scale of 1 to 10. 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. From there, we are getting the 90% by adding up all the numbers we got for each question and seeing if it's greater than 90. If it's not, we are planning on continuing to develop the web application and make the appropriate changes.

III. Architecture and/or Principle of Operation

The user video feed from the camera will be 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. This video will also be sent to a computer vision processing component that utilizes MediaPipe, an open-source framework that will allow us to extract feature data from the user's hands. This feature data will consist of absolute position coordinate data that provides information about the shape/orientation of the user's hand. This data will be propagated to the machine

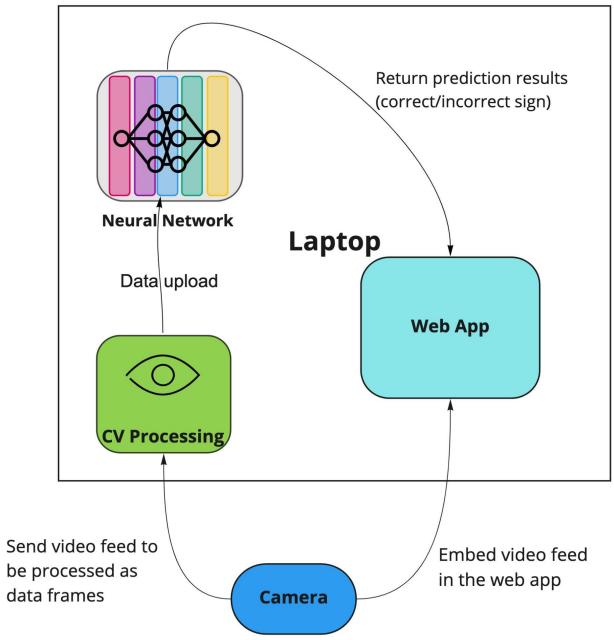

Fig. 1. Solution Approach Diagram

learning component of the system to then be fed as inputs to a

neural network model. This neural network model is selected from among 5 neural networks - see Figure 5 at the end of this report - that each support prediction generation for a specific subset of sign language gestures. Based on which subset the expected sign belongs to, the correct neural network will be selected for execution, and it will generate a prediction for what sign the user is making. In comparing this prediction 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).

IV. Design Requirements

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 data input into 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. Ideally, we want this latency to be less than 2 seconds, and keeping the tech stack locally for now can help us achieve this metric.

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 research, the average number of users that a locally hosted application can handle is 5 users (1). This aligns with our own experience using local hosting for web applications. Due to having only one physical system handling the computer vision and the machine learning models, we are not prioritizing support for a large number of concurrent users. In addition, we are focusing on an individual user's experience, and users are not interacting with each other on the web app. Therefore, concurrency is not a major aspect of our application that we need to worry about.

For our machine learning model, the average raw video input is 3 seconds long among the training and testing data, and, from preliminary testing, an average user does not take more than 3 seconds to make a sign. Therefore, we require for all data to be 3 seconds long to maintain uniformity. Because we tested it will take 3 seconds for the average user, we are not taking into account users with disabilities like arthritis. Real-time video feed will be fed to MediaPipe at 30 frames per second (fps). We will re-sample the videos we have at a lower frame rate, allowing us to only have landmark data from 10 frames sent to our neural network rather than 150 frames. Limiting to 10 frames will help us train our neural network and will facilitate getting a predicted label faster. Considering we are receiving 3-second clips, we don't want our machine learning model to have to process too much such data that neural network training and evaluation slows 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. We believe that it is sufficient to use 10 frames, especially given the feedback we received from Professor Matt Gormley who teaches Introduction to Machine Learning at CMU.

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. We will exclusively retrieve landmark data for the hands, and all the signs we will train the models to identify only involve the hands. The reason for this is due to the additional complexity of dealing with signs that involve contact with the face or body. Such signs would require a far larger amount of landmark data to be fed to the neural networks. Apart from that, we would need to calculate a distance/position error margin for users touching a certain part of the face for the sign.

To observe model execution speed capabilities, we tested our baseline model structure to determine how long it takes for a model to generate a prediction. With the experimental data that we currently have, we found that average model execution time is 0.07 to 0.12 seconds over 15 trials. This baseline model that was passed the experimental data only contained one LSTM layer and one Dense layer. As of now, we are planning on only using these two layers for our neural network. As we tune the models, however, it is likely that we increase the number of layers in order to improve prediction accuracy. This will require us to observe how increasing the size of our model to achieve greater accuracy 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.

In the datasets we are using to train and test the neural networks, the images and videos are of correct signs. As a risk mitigation plan, if we see that our neural network model is not accurate we are going to make our own database of incorrect images for signs. Adding incorrect images can help detect if there are false positives (incorrect signs being falsely identified as a particular sign) since that may be a likely source of error. Because of this, we added time to our schedule to make 30 incorrect images for each sign to add into our testing database.

For training our neural network, we are taking advantage of

the AWS credits available to us and using an EC2 instance for training. Training a neural network takes a long time to do, which ends up occupying our computer resources and impedes us from using our computers for other work. For this reason, we decided to take advantage of the EC2 instance to conduct model training. In doing so, we will also be able to train multiple models concurrently, allowing us to take advantage of all the training time that we have. Furthermore, we are using a GPU inside our EC2 instance which will significantly speed up our training time. Using a GPU helps us perform multiple, simultaneous computations which is helpful when we are trying to train five neural networks at the same time (2).

V. Design Trade Studies

A. 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 [3]. GRUs have 2 gates: the update gate that allows a subset of past info to be carried forward, and the reset gate which allows a subset of past info to be ignored [3]. 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 [3]. Because we have such large datasets of video 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 that this result may vary based on the task the models are being trained for [4]. 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.

B. 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.

C. 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 into 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. 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 (as well as model training) will be carried out in a p3.2xlarge EC2 instance with a Deep Learning AMI that has Tensorflow preinstalled. We will be using open-source datasets 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.

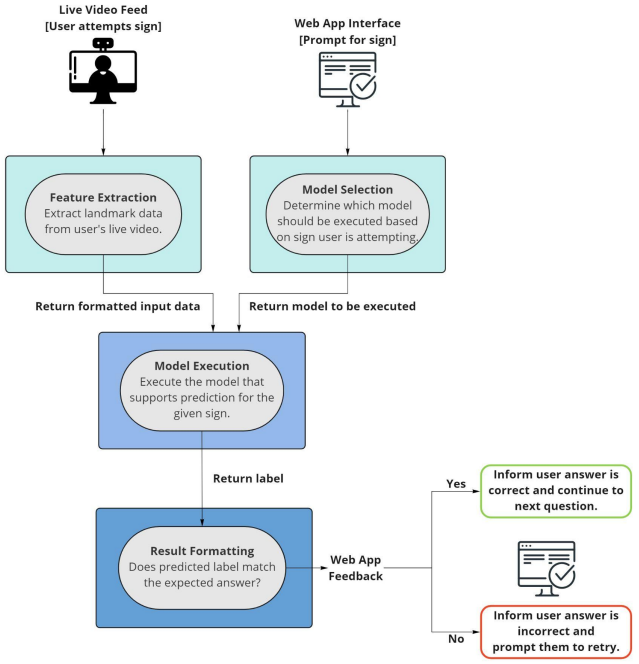

Fig. 2. Software Block Diagram

A. Computer Vision Component

We will 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 will be 21 landmarks for each hand (resulting in a total of 42 landmarks) that will be stored as flattened numpy arrays. If only a single hand is detected and just 21 landmarks are present, the other 21 must then be padded as zeros. For each example, we must collect 10 frames of flattened landmark data vectors. The numpy arrays for each example will be stored and split into subsets of training, testing, and validation data to use accordingly.

For each example from the open-source datasets, 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 useful for our model to learn from).

Because pre-existing videos may have slightly different durations (for example, one person may take 5 seconds to complete a sign, while another person takes 3 seconds), we will need to resample at different rates in order to select a total of 10 frames from the time the sign starts to the time it is completed. For instance, a duration of 5 seconds will require a resampling rate of 2 frames per second. We will use MediaSequence in order to conduct this resampling on video data. For pre-existing image data, we must extract landmark data from the image and format it as a flattened numpy array, duplicate the array 10 times and noisify the data to closely mimic variation observed normally in video input (greater presence of noise at beginning and end of a video, versus less noise towards the middle of a video due to the user stabilizing their static sign). Once the data has been formatted for training and testing, it will be fed to the machine learning component of the system to carry out model creation and tuning.

For real time evaluation, there will be a specified window of time (3 seconds) within which the user must attempt a given sign specified by a prompt in the web app interface, and the landmark data from their input will be collected. We will only collect data within this time window and as long as a hand is detected. If a user takes less time than they are allotted, data collection will stop. The user's input will be resampled to acquire 10 frames. Once a batch of landmark data from the 10 frames is collected, this will be passed to the machine learning component of the system to evaluate and assign a prediction label.

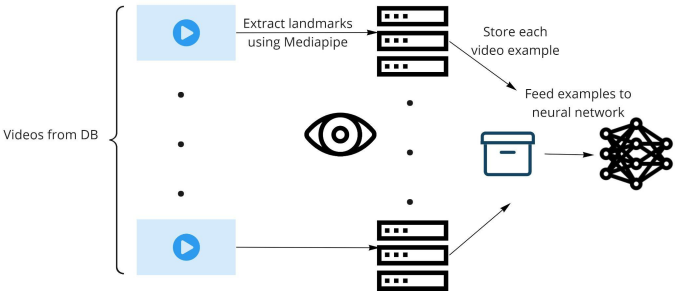

Fig. 3 Feature Extraction Pipeline

B. Machine Learning Component

The models will be instantiated, trained, and saved using Tensorflow on an EC2 instance. The initial model structure will be created as a Sequential model instance with at least one LSTM layer followed by at least one dense layer. The LSTM layers will allow the model to take temporal information (ordered sequencing of the video data) into account for classification. LSTM cells propagate information forward as well as to each other (older time steps inform future time steps). Further, the model will conduct categorical classification, where the output generated will be a set of probabilities indicating the likelihood of each possible label being the correct prediction. We will then use argmax to select the highest probability, and the predicted sign will be the label associated with this probability.

The models will be instantiated using Tensorflow. The initial model structure will be created as a Sequential model instance with at least one LSTM layer followed by at least one dense layer. The LSTM layers will allow the model to take temporal information (ordered sequencing of the video data) into account for classification. LSTM cells propagate information forward as well as to each other (older time steps

inform future time steps). Further, the model will conduct categorical classification, where the output generated will be a set of probabilities indicating the likelihood of each possible label being the correct prediction. We will then use argmax to select the highest probability, and the predicted sign will be the label associated with this probability.

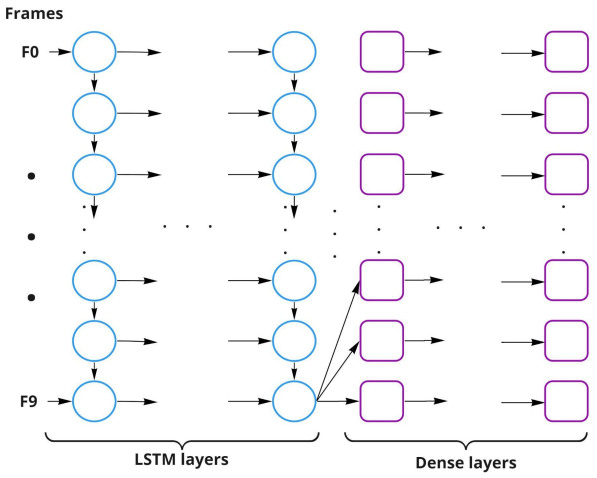

Fig. 4. Neural Network Structure

The models will also be trained and saved using the Tensorflow API. Each of the 10 frames for a given example is treated as a time step fed in at each node of the model's input layer. Training data will be passed to the model using the Tensorflow model.fit() method, whereas testing and validation metrics will be generated through the model.evaluate() method, which will allow us to observe the model prediction accuracy. The methods model.save_weights() and model.load_weights() will allow us to preserve the model after training is complete (so it can be used later in real time sign language evaluation). Table I denotes the different hyperparameters we will have to tune and the ranges for these values that the tuning process will explore. The learning rate initial value is the default learning rate that the provided Tensorflow optimizer uses. Further, because our training dataset is very large for some signs, we will begin with 10 epochs and increase this should we need to improve model accuracy. At the same time, increasing some of these parameters too much may result in overfitting to the training data, so we must exercise the range to determine the ideal combination of hyperparameters for model training.

TABLE I. HYPERPARAMETERS FOR TRAINING

Hyperparameter	Initial Value	Range [min, max]
Number of frames	10	[10, 30]
Learning rate	1e-3	[1e-4, 1]
Number of epochs	10	[10, 1000]
Number of LSTM layers	1	[1,5]
Number of dense layers	1	[1,5]

While using this ASL learning system, a user will be generating signs in real time in front of a web camera that we must identify and return feedback for. There are two primary system steps for this, selecting the correct model to execute and collecting the data in real time to pass to the model. Initially, the web app indicates the sign being requested (this will be based off of the user's selection for what sign they want to practice, or what sign the app requests the user to do during a test, etc.). The requested sign will be the label that indicates which of the 5 models we must execute, as well as the label we compare the model's prediction against to determine whether the user made their sign correctly. The web app itself will conduct this comparison and format it accordingly for the user to understand (format a response to display on the webpage for the user to know whether they did the sign correctly).

C. Web Application

We are using JavaScript and HTML to create the web app 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 lesson plans they have completed. The pages of the web app will consist of login, registration, homepage, courses, learning mode, and testing mode. 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. 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. 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. 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 both real-time video feedback and an instructional demo video showing how to do the sign correctly.

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. As of now, we will send 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. Having the boundary of the real-time video feedback box light up will capture the user’s attention and make these correct results clear.

As for how the computer vision model is receiving video feedback of the user’s attempt at a sign, a 3-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.

Apart from that, we do have a risk mitigation strategy if we see that the accuracy of our models is too low. If this issue arises, we plan to have a bounding box that can restrict what portion of the frames the computer vision model looks at to collect landmark data. The bounding box will help restrict the view and have its main focus be on the user’s hands. Within the real-time video feedback, we are going to display this bounding box to the user and instruct them to make their signs within it. The feedback that we get within the bounding box will then be sent to our computer vision model, hopefully improving it by restricting the portion of each frame from which it must extract hand landmark data.

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 will examine the MediaPipe data received using the Hands library on a web camera, the format of the data before it is sent to our deep learning model for processing, and the correctness of our created testing data along with the training data found online. For the Hands library data from MediaPipe, we will verify that when hands are in frame, they are detected with 21 landmarks each as specified in the MediaPipe documentation. We will ensure that this works both with live video feed and when given pre-existing video or image data. In terms of the data format, we expect that the 21 landmarks will have x,y,z coordinates associated with them, which is verified both in MediaPipe demos and in our own experience testing the Hands library. Finally, we will verify 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 will verify 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 will conduct tests based on the use-case requirements detailed in section II of this paper. To reiterate what these requirements are, we will be testing 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 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 will conduct 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 have 97% accuracy in determining sign correctness at any distance within and including 3 feet. If our tests for this requirement fail, we will 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 will be conducted in regard to the platform latency, which we have decided will be within 2 seconds. Note that 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 related to this prediction rather than the responsiveness of the ASLearn platform itself. In order to test this, we will time how long it takes for the site to give the user correct feedback, where we will start our timer once the user has stopped their sign, which should take no longer than 3 seconds, as explained in Section III of this report. If we find that our latency consistently exceeds 2 seconds, we have the contingency plan of adjusting

our prediction generation algorithm to be a faster but potentially less accurate method. For instance, we may try to use a CNN for faster execution (and reduced data formatting complexity), which would simply take in image data at the input layer; although, it may pose lower accuracy due to interference from extraneous colored pixel data surrounding the hands performing sign language.

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. The 3% where this could be inaccurate would mostly be due to environmental impediments such as poor lighting or the user wearing jewelry or other accessories that could interfere with hand recognition through MediaPipe. Thus, we will conduct tests in various lightings (i.e. low light, natural light, studio light) and with distracting accessories (bracelets, rings, nail polish, etc) to check if our platform remains at least 97% accurate in the sign predictions. If this test fails, we will communicate to the user that before testing signs on the platform they should be in a well-lit space and have little to no impediments on their hands.

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 will be conducting user satisfaction surveys after having users test the ASLearn platform based on certain usability metrics. Our plan is to interview at least 10 users in the later weeks of the semester and have them rank features of ASLearn - like navigability and how intuitive it is to use - from 1 to 10. Overall, we plan to have at least 90% user satisfaction based on the industry standard. If the survey results are lower than expected, we will do what we can to modify the user interface based on user feedback in order to improve the overall experience. If there is enough time between the given user feedback and the final demo, we will potentially 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 6, 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 6, it should be noted that we have planned for the last four weeks of the semester to heavily involve integration and contingency planning. In regard to integration, our solution has a lot of different parts - MediaPipe, neural networks, web application, and external camera feed - that need to be combined in the overall ASLearn platform. Thus, by allowing multiple weeks for integration we can account for obstacles that could arise due to data formatting, incompatible tools, etc. As for contingency planning, in Section VII it was mentioned that we will have risk mitigation and contingency plans in place if any of our validation tests fail. So, the contingency task on the schedule is to allow time for us to alter our solution based on that risk mitigation so that by the end of the semester we still have a product that meets our requirements and specifications.

B. Team Member Responsibilities

In regard to team member responsibilities, we have divided the labor between our three group members with the more intensive tasks involving effort from everyone. Aishwarya will mainly be 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 will be 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 will be doing the web application UI design, the deep learning model with Aishwarya, and testing of the lighting/environmental requirements. Tasks that all three of us will be working on include fine-tuning the deep learning model, making testing data for the model, 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. Risk Mitigation Plans

The critical risk in our project mainly concerns processing the input and detecting the correctness related to our chosen dynamic signs. The reason why this is a huge risk is because most of the similar ASL recognizing deep learning models only include the alphabet and numbers, which are largely static signs. Thus, by creating a model to also include 15 dynamic signs, we are doing something a little less common and more complex. As a result, we have less training data available online for those signs and thus must augment the data we currently have in order to better tune our model.

For risk mitigation in this sense, we are considering creating our own training and testing datasets. Our best solution right now is to create a Python program that can iterate through YouTube and find videos that include our desired signs in the title. From the concise list that we will receive, we will double check that the videos contain the signs and that there is sufficient duration of time consisting of the video subject making the sign for our neural network's 3-second restriction. Otherwise, another option that we are considering is to do one-to-one mapping instead of an LSTM network for dynamic signs. However, we are still going to use LSTM for static signs because we want consistency and clearness in the signs that the users are making.

IX. 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].

X. 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.

Upcoming challenges to our design will include achieving our goals for model prediction accuracy due to requirements for formatting input data and collecting data to be used for tuning our models. Thus far, we have considered many ways that we may mitigate these risks, such as various sign detection methods, planning our approach to hyperparameter tuning, and considerations for making sure we have enough data to train and test the models with. Thus far, we have reviewed many solution approaches and hope to continue perfecting our design as we progress through developing the implementation.

Glossary of Acronyms

AJAX - Asynchronous JavaScript and XML
ASL – American Sign Language
CNN - Convulational 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] 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.

[4] 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.

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

[6] 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/.

[7] 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..

[8] 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-/

[9] *Recurrent neural networks (RNN) with Keras : Tensorflow Core*. TensorFlow. (2022, January 10). Retrieved March 4, 2022, from https://www.tensorflow.org/guide/keras/rnn

[10] 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

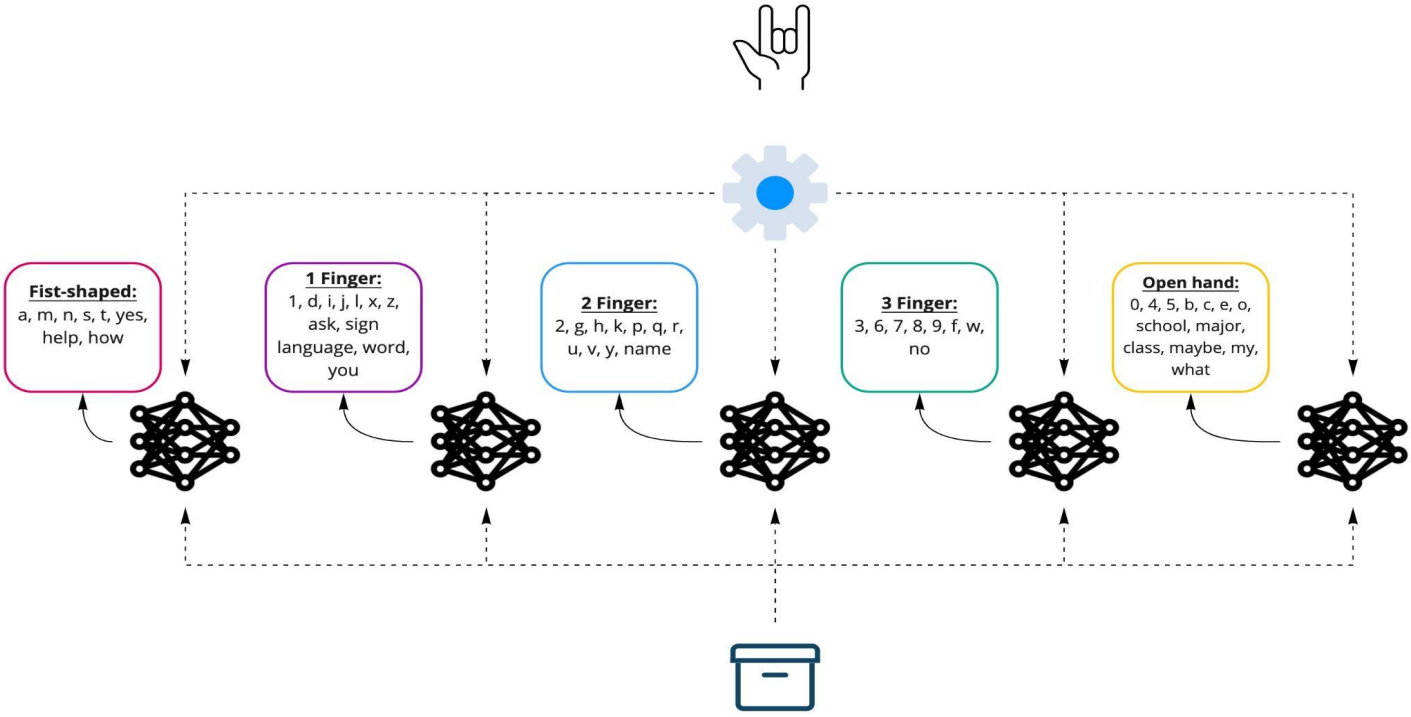

Fig. 5. Neural Networks as Microservice Architecture

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. 6. Project Management Schedule