Mesa - A Smart Meeting Table

Team C2 Design Report

Authors: Raunak Sanjay Gupta, Arman Hezarkhani, Olivia Weiss
*Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Mesa is a smart meeting table that aims to better facilitate collaboration, communication and record keeping in meetings. It allows users to write on an accompanying touch screen, converting their handwriting into text and digitally embedding it into documents and presentations hosted on the table through Google Cloud APIs. State of the art neural networks are used for the classification of letters. Users can interact with the smart table using a touch screen and simple hand gestures, making it very easy to use.**

***Index Terms*—Cloud, Collaboration, Computer Vision, Gesture Recognition, Object Character Recognition**

I. Introduction

More often than not, meetings default to one or a few members who present ideas to their teammates and coworkers. While others are able to present their ideas and discuss the problem at hand, it is challenging for them to directly contribute to the development process. The goal of our project is to address three common problems with working in a group: collaboration, communication, and record-keeping.

Our approach to solving these problems is with Mesa: the smart table. This table facilitates collaborative meetings by having an interactive dashboard capable of displaying documents such as slides and images. The table also has a notes widget which records and displays notes that a user takes. The user can use hand gestures to interact with these documents, and to take a screenshot of the current table display. Using these tools, meeting collaborators can display and interact with each others work, making comments and keeping track of their progress along the way.

Finally, there is a touchscreen component of the smart table, which serves multiple purposes. The touchscreen acts as the interface to the notes widget displayed on the smart table dashboard. The user can write their notes on the touchscreen and, using OCR, our table translates the handwriting into text and sends this text to the notes widget. That way, users can take notes as they usually would on a notepad. The touchscreen also takes care of account information for the table, so the user can input any relevant granular data for the table to use. All of these tools allow members of a meeting to behave as they would normally, but have a more meaningful and interactive discussion with each other using our table dashboard.

II. Design Requirements

This section discusses the key design requirements of this complex system. We present the main functionality we aimed to implement, and then discuss metrics that we used to test and validate our achievements with respect to the requirements. We also detail the requirements from the hardware and software we used.

A. Gesture Detection

- The average time taken to detect and correctly classify a hand gesture is no more than 2 seconds. We call this the *gesture detection latency*. This is to ensure that the table responds quickly to user input, and no considerable lag that can impact consumer experience is noticeable. We manually tested and measured the *gesture detection latency* over all the supported hand gestures, with the average response time being significantly less than 2 seconds in all the 10 trials that we conducted.
- We require $85-90\%$ accuracy in classifying the various hand gestures. Each hand gesture has its own required accuracy target, due to the inherent difference in the complexity of detection associated with each gesture. These accuracy requirements were judged against a plain white background. Please refer to the following table for the accuracy we were able to achieve for the different hand gestures.

Gesture Type	Target Accuracy	Achieved Accuracy
Clench	90%	92%
Two-finger tap	85%	87%

Table 1: Different gestures, their targeted and achieved accuracy

B. OCR Segmentation and Classification

- Because we are performing OCR on characters written on a touchscreen, our letter segmentation is close to perfect. This is especially true because our use case requires users to print their sentences rather than writing in cursive. This means that we create a bounding box for each letter based on when the user places and picks their finger. There exists a small amount of segmentation error, for reasons such as overlapping letters, but otherwise our segmentation is ideal. Therefore, we are able to segment letters with 100% accuracy, disregarding user error.
- Our word segmentation is based primarily on spacing. Letters that are grouped more closely together are part of the same word and letters that are grouped farther apart mark the end of one word and the beginning of another. This is based on the size of the letters and the expected average spacing. Although the touchscreen provides reliable information about this spacing, it is possible a user's handwriting could have words spaced too closely together. Because of this, we account have two separate requirements, dependent on the user's handwriting:
  1) Words are spaced farther apart than letters. In this case, we have 100% accuracy. This case is ideal.

2) Words are spaced closer together than letters, equally close, or have variable spacing. This does not fit in the scope of our project.

- Based on some common text classification papers, classification accuracy can on average range from around 87% - 97% [1][2][5]. Our classification model is not as in depth as the models from these papers due to time constraints. However, our classification challenge was simplified by the automatically-centered bounding box and the binary colored text on a white background. Therefore, we required our classification to be the average of these accuracies: 92%. Based on user testing, our final classification accuracy was 95%.

Action	Achieved Accuracy
Letter Segmentation	100%
Word Segmentation	100%
Classification	95%

Table 2: Different OCR components and their final accuracy

C. Overall product

- We aim to allow users to collaborate on and display their work documents while they use the table. To this extent, we allow users to interact with the documents that they upload through the dashboard. This is exclusively reliant upon Google Firestore and Google Cloud Storage, which guarantee 99.99999% success.
- The goal of our project is to have a smart, interactive and easy to use table. With that goal in mind we will design and fabricate our table such that it facilitates collaboration with up to 4 people at a time in an effective manner. To test its ease of use and success at promoting collaboration, we conducted a series of *user studies,* inviting different people such as students to test it and rate it on the desired features.

III. Architecture

A. Interactive meeting table

This subsystem is composed of a few different pieces of hardware. For the table, we use a TV screen, which is mounted onto a table. This TV screen is used to display the web dashboard in a Chromium browser. The table is controlled using a Raspberry Pi 3 B+, which is running the web UI. The Raspberry Pi 3 B+ is also connected to a camera mounted on top of the table, through which it receives a continuous video stream of the table surface. A lightweight Tornado based web server, written in Python, receives these image frames, and run the gesture detection classifier, communicating both between the web dashboard and the Google Cloud to take the appropriate action.

B. Touch screen tablet

The tablet is one of the main ways the user interacts with the table. Specifically, the user can control what is displayed on the table through the touch screen and also write notes onto the touchscreen, which triggers the OCR procedure to segment the letters and classify them accordingly.

C. Google Cloud

Google Cloud is the mechanism through which we will perform all our heavy computation and store all user data. Specifically, we will use Google Cloud Firestore for persistent data storage and Google Cloud Storage for blob file storage. We built a dashboard to interact with our cloud backend.

D. Web dashboard

This web dashboard is the main way that users interact with the cloud back-end. They can view a gallery of screenshots and upload and download files and presentations.

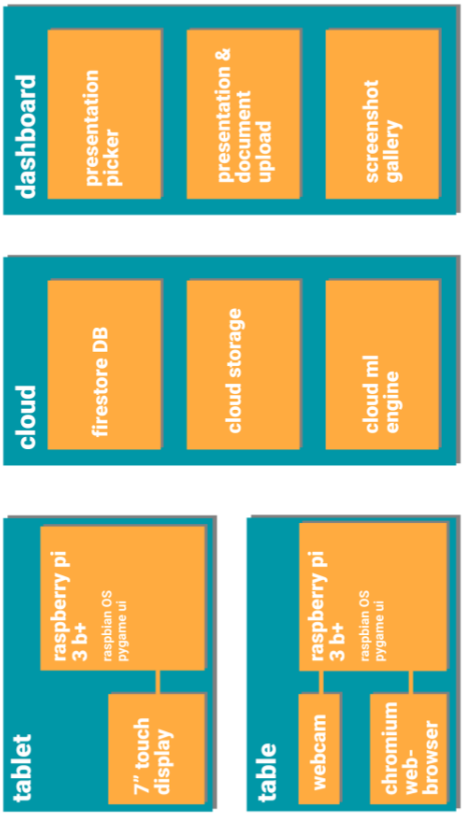


Fig. 1. Mesa - Overall System Architecture

IV. Design Trade Studies

A. OCR Study

There were a few different trade-offs to evaluate when deciding to implement the OCR for this project. First, we had to decide how to segment the handwritten letters. We considered a few options for this:

1) The user would write directly on the table, and we would filter out the background using a screen-shot of the table display without the writing on it

2) The user would write on a ”notepad” displayed on the table with a white background
3) We would build a pen that could track when it was touching the table and its current coordinates using hardware

We evaluated each of these options. The first option seemed overly complicated and not likely to work. The image that would be taken by our camera to filter out the background of the writing would be different than the image being displayed on our TV for a variety of reasons, including camera angle and lighting. We didn't want to dedicate our time to trying to solve a segmentation problem this way when we should be focused on classification. The third option had the potential to work, but again was very time-intensive. Considering we had multiple other demanding aspects of our project, we didn't want to add the hardware of a pen to the list.

This brought us to the second option: writing on a notepad with a white background on the table. This idea still had its issues: the notepad could be anywhere on the table, and the user could write on it from any angle. However, once we simplified the problem to this extent, we realized we could simplify it further by having the user write on a touchscreen instead. This would not limit our handwriting feature any more than the second option would, but it would make segmentation as simple as keeping track of pen touches and lifts. This is the idea we settled on.

After deciding on the segmentation plan, we had to settle on a classification plan. Originally, we were going to use a deep convolutional neural network for text classification, as is convention in the field. However, upon further research, it turned out the convolution would not be necessary. The purpose of convolution is to check all areas of a picture for an item to classify [3]. Our segmentation is the ideal case, with centered, binary text, so this would not be necessary. Instead, a simple neural network would accomplish the same task and require less computation power and setup.

B. Cloud System Study

When analyzing cloud platforms, we weighed our options with a few variables in mind. First, we considered our needs. We needed a REST API, a persistent database, file storage, and an instance to run our ML Model. Additionally, we considered how complex the authentication would be. Many times, implementing complex cloud architecture is difficult due the need to authenticate every request between instances. Lastly, we considered cost of the cloud provider.

After analyzing the big-three cloud providers, Amazon Web Services, Microsoft Azure, and Google Cloud, we landed on Google Cloud Platform. Google Cloud Platform meets all of our needs. Regarding authentication, it uses service accounts that can be stored in environment variables, making it easy to authenticate cross-platform. Lastly, they provide free and easy-to-use cloud credits. Another very important added perk is that we can use the same platform to house, run, and authenticate our Google Suite integration.

C. Gesture Detection Study

To perform gesture detection we primarily experimented with a few different computer vision based approaches before deciding to settle on using a camera. We decided to stray away from building our own classifier based on Machine Learning techniques as we are only aiming on detecting a small number of hand gestures that are fairly simple - which should be achievable using a purely computer vision based approach.

The first approach we experimented with was using a Leap Motion. The Leap Motion is a small device built primarily for VR/AR applications that uses an infrared and depth sensor to detect hands. This device allowed us to get hand joint data and implement code to detect hand gesture swipes. However, after experimentation and running a few tests, we came to the conclusion that the Leap Motion would not be viable for our use case as the sensor was only able to detect hands to a depth of 0.3m and had a lateral vision of around 0.25m - which was significantly less than the range we would have needed when the Leap Motion was mounted on top of the table.

We then proceeded to experiment with the Kinect - a device that has a camera and depth sensor, allowing for the tracking of body joints. The Kinect has a much longer range. (of up to around 3-4m) Although this solved the problems we had with the Leap Motion's range, it introduced other significant challenges. After writing some preliminary test code we learned that the Kinect can only track hand joints accurately only if the entire body is being tracked. Since tracking only the hands would lower accuracy significantly, we decided to switch to a purely computer vision based approach to detect the finger tips.

We finally decided to settle on using a Logitech HD Web Camera. This allowed us to get video frames from the camera, after which we find contours and find the convex hull of the largest contour to isolate the shape of the hand. We combined this with a K-means color segmentation approach with $K = 2$, to have a more robust hand detection algorithm. Using $K = 2$ allowed us to best cluster all skin colored hand points into one cluster, and the other points into another cluster, which was ignored. We then identified a clench and tap based on the number of convex hull points detected - a clench led to 0 convex hull points, where as a tap resulted in 2 convex hull points corresponding to the two open finger tips.

D. Hardware and software choices

In this section we quickly describe the trade offs we made in choosing specific hardware and software packages. Firstly, we decided to use 2 Raspberry Pi 3 B+'s for powering our entire system. The reasons we chose these were because we wanted a Linux based system that had stronger WiFi support (for quicker and more efficient communication between components) and more heavy compute capability. Out of all the options we considered the Raspberry Pi 3 B+ turned out to meet all these requirements.

To implement the hand gesture detection we decided to use a Logitech HD web camera to get realtime video frames,

which we then processed using a K-means color segementation algorithm and a convex hull algorithm to detect the hands.

To interface with the Raspberry Pi Touch screen we decided to implement a UI in Python using Pygame. By using a high level language such as Python, we were able to focus our time on the image segmentation and OCR, rather than writing low level driver code to interface with the touchscreen.

V. System Description

A. Touchscreen Implementation

The user can write notes on a Raspberry Pi 7 inch touchscreen display. These handwritten notes are shown in Pygame, which runs on the Raspbery Pi touchscreen. The user can use a button displayed on the touchscreen to indicate that the note is finished and ready to be processed. On the press of this button, the touchscreen Raspberry Pi sends an image of the Pygame screen, along with additional information, to our Cloud Functions endpoint and is redirected to our Cloud ML Engine Instance. This instance runs a neural network in Tensorflow to perform OCR on the handwritten text. The results of this OCR is saved in the Firestore DB, which is accessed later by the smart table to display the note on the dashboard.

There are two main components to the OCR itself: segmentation and classification. The segmentation aspect of this challenge is relatively simple. Since the user is writing on a touchscreen, segmentation information is saved and updates in real time. The user beginning to touch the screen indicates the beginning of a new letter. Up until the user stops touching the screen, the touchscreen Raspberry Pi keeps track of the minimum and maximum X and Y coordinates seen so far. Once the user stops touching the screen, this information becomes the bounding box of that letter. Once a new line is drawn, if the line is close enough to the previous bounding box, that bounding box is expanded to include that line – this accounts for letters with multiple strokes. On the press of the button, the bounding boxes are used to take screenshots of the letters. Next, a combination of Gaussian blurs and image processing are used to format the screenshots to match the formatting of the EMNIST dataset. The results of this preprocessing are sent to Cloud Compute Engine for classification. The letter order is maintained using the X coordinates of the bounding boxes. Word separation is also maintained by assuming spaces significantly larger between words than between letters.

Text classification was the challenging aspect of the handwriting OCR. However, since the letters are be so perfectly segmented, the classification is done without the use of a convolutional neural network. Convolution is mainly used to classify an item that could be anywhere within an image. However, in this case, the letter is always at the center of its bounding box, so a regular neural network can accomplish the same task. Our neural network has three layers: an input layer, a hidden layer, and an output layer. The input layer has 28 x 28 = 784 neurons. The hidden layer is approximately half the size of the input and output layers: 400 neurons. After experimentation, the activation function is ReLU, which significantly improved performance from Sigmoid. Finally, there is a Softmax output layer that has 62 neurons (10 digits + 26 lowercase letters + 26 uppercase letters).

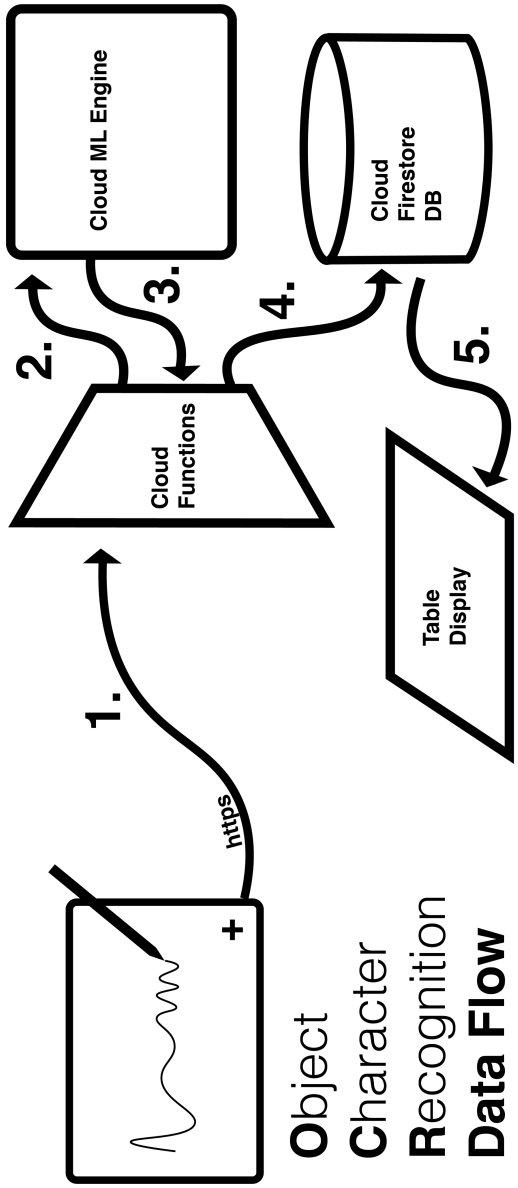


Fig. 2. Mesa - Touchscreen Data Flow

Fig. 2 shows in detail the setup for the touchscreen to interface with the rest of the system. There are two flows of information for the touchscreen: the OCR flow and the granular user input flow.

The OCR system begins with a user writing on the touchscreen with their fingers. When they press the + button, the Python Driver software creates an HTTP request containing a screenshot of the UI containing the handwriting, along with a JSON object containing XY pairs to segment the characters. This HTTP Request is made to the Cloud Functions REST API (see arrow 1). The Cloud Functions API then redirects this Request Body to the Cloud ML Engine Instance (see arrow 2) which runs the OCR and sends a response to the

Cloud Functions API (see arrow 3). This response is finally stored in the Firestore Database using the Firestore Client Library, which is a simple wrapper over the HTTP protocol (see arrow 4). To display the results of this process, the table polls every 10 seconds (see arrow 5) to refresh the display with any updated Notes objects. The granular user input flow is also fairly similar. When a user inputs information, this information is bundled into an HTTP Request (see arrow 1), and stored in the Firestore Database (see arrow 4). The table is polling for updates to this information every 10 seconds (see arrow 5) and displays the updated data as it is changed.

B. Interactive Table Implementation

This subsystem consists of the table display itself, along with the Logitech HD web camera that are both connected to a Raspberry Pi 3 B+. This subsystem is in charge of performing the hand gesture recognition and interfacing directly with the screen mounted on top of the table.

To perform the gesture detection, we decided to use a purely computer vision based approach. We have a gesture detection script, which is run as part of the a local Tornado based Python web server. This gesture script is implemented in Python and uses OpenCV for primitive image manipulation. The algorithm to detect gestures begins by first extracting the current frame and blurring it using a Gaussian Kernel. We then apply a K-means based color clustering algorithm, with $K = 2$ to isolate the hand in the frame. (one cluster is approximated as the skin color of the hand, and the background is classified as part of the other cluster) The segmented image is then thresholded using a binary mask to extract the shape of the hand. The next step involves finding the largest contour (the hand), and finding the convex hull of the contour to detect the finger tips, and the convex defects to find the points between fingers - allowing us to construct a visual map of the hand. This step forms the basis for detecting the various hand gestures we implemented. Here we describe the general approach we will follow to a couple of gestures:

A clench is easily detected by noticing a significant change in the spacing between the convex hull vertices and the convex defects, along with a sharp decrease in the height of the convex hull vertices (which corresponds to finger tips). Secondly, a 2 finger tap is detected by checking the number of convex hull vertices that are present.

This approach to hand gesture detection worked very well on plain solid backgrounds, as background subtraction in this case was straightforward. However, once we moved to testing on more complex and noisy backgrounds such as the documents presented on the table, background subtraction became significantly more complicated, leading to erroneous results with the hand gesture detection. This turned out to be a significant challenge that proved to be very hard to fix. Many of the approaches to background subtraction relied on a constant/non changing background, which did not fit our use case. Therefore, to improve the accuracy and robustness of our hand detection algorithm, we decided to incorporate a small rectangular red foam that the user holds as they interact with the table. By adapting the K means color segmentation algorithm to first detect this red foam (that is in significant contrast to the hands) and then the user's hands we were able to significantly improve the accuracy of our hand detection algorithm. This just required minor changes to our algorithm, as we first filtered out pixels that were not classified as "red" to isolate the red foam, before proceeding to detect the hands in the neighboring pixels.

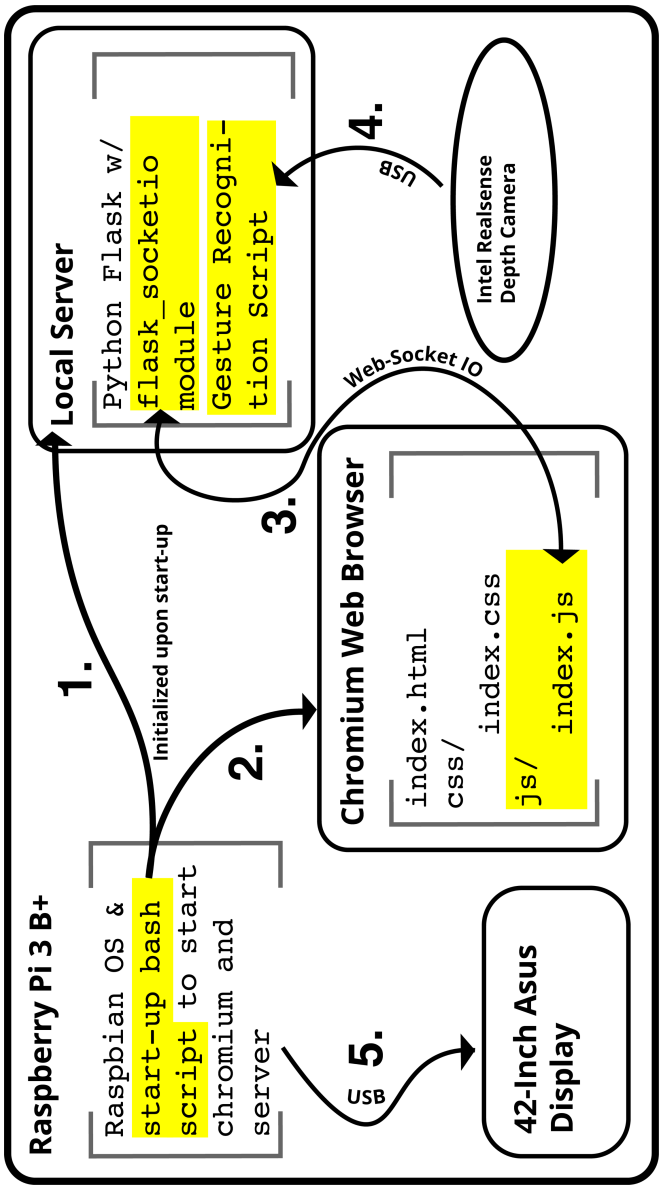


Fig. 3. Mesa - Gesture Detection Data Flow

Figure 3 shows the data flow and how the components within this subsystem interact. When the Raspberry Pi is booted up, we first use a start-up bash script to start running both the Tornado webserver and the Chromium web browser. (refer to arrows 1 and 2 in Fig 3) Upon startup the Chromium web browser establishes a web socket connection with the web server to communicate. (refer to arrow 3) In the Tornado web server we use a separate thread to run the hand gesture

detection code. This thread interfaces with the Logitech HD web camera, (refer to arrow 4) reading in the video stream frame by frame and run the hand gesture detection code which was detailed before. Whenever a gesture is detected, the web server calls the appropriate Google Cloud API endpoint function to indicate that a certain gesture was detected to trigger the appropriate action. This change is also communicated to the Chromium browser running in the display to reflect the change on the table screen itself. The Chromium web browser is displayed on the screen through a direct wired HDMI connection with the Raspberry Pi. (refer to arrow 5)

C. Cloud System Implementation

Initially, we wanted to have one single Rest API running on Cloud Functions, with the intention that it control every other cloud component. However, we realized that this wasn't necessary, and that it would add a whole new aspect to the project with no new technical benefit. Instead, we decided that we'd use the API libraries to communicate with the Firestore and Storage tools.

D. Web Dashboard Implementation

The meeting table is run using a Raspberry Pi 3 B+ running a Raspbian OS. The table consists of a HDMI enabled television screen – laying horizontally – as the main display and is powered by a wall outlet. Upon start-up, this Raspberry Pi instantiates a chromium web-browser and a local Python web-server. The chromium web-browser runs in Kiosk Mode, allowing it to be rendered in a full screen mode. Upon start up, it immediately requests static files from our CDN, allowing it to render our table User Interface. The local web-server is built using the Python Flask web framework. The server polls data from our webcam, and runs the gesture-recognition script to draw conclusions about the user's interactions with the table. When the server decides what gestures have been performed, it pushes the changes that need to be made to the UI using a websocket connection to our Chromium web browser.

The Chromium web-browser is also in HTTP communication with our backend in Google Cloud. It polls every 10 seconds to query the database for any new Notes that could have been made on our tablet.

VI. Project Management

A. Schedule

Our schedule ended up changing from our initial prediction in a few ways. First, we spent much longer than we expected on initial research. We learned that research and design play vital roles in approaching a challenging project such as this one. Also, we spent longer than expected on both the gesture detection and OCR components of this project. There were many more phases of testing and iteration than we initially accounted for. Finally, we started working on some components in parallel, because multiple of the components relied on each other to be advanced.

See Appendix A for a full schedule.

B. Team Member Responsibilities

To break up tasks, we divided our work into three overarching areas, each of which was assigned to a team-member. Olivia was primarily responsible for OCR, Raunak was primarily responsible for gesture recognition, and Arman was responsible for our modules interfacing with Google Cloud and integration in general.

As a part of OCR, Olivia primarily worked on training a Neural Network to do OCR and implementing a touch screen UI in Pygame to perform letter segmentation.

For gesture recognition, Raunak was primarily responsible for writing software that detected hand-gestures using a Logitech HD web camera. Raunak was also responsible for feeding the results of his gesture recognition software to the Chromium Web-browser using web-sockets on Python's Tornado web framework.

For our Google Cloud usage, Arman was primarily responsible for integrating all of the parts, by designing and implementing a multi-tiered cloud and software architecture, including a REST API, a No-SQL Database, a Cloud Storage Bucket, and a Machine Learning Model running on a cloud instance. As a secondary responsibility, Arman designed and implemented the Table and Dashboard UIs.

C. Budget

The break down of our budget and the parts used are detailed in Appendix C.

D. Risk Management

To minimize our risk, we designed a risk-management plan. This plan included a *bare-bone*s MVP, as well as minimum deliverables for each of our respective responsibilities.

1) MVP: Our MVP was defined as a table capable of displaying and interacting Sticky Notes. Specifically, as part of our MVP users can put new sticky notes on the table by writing with their fingers on a tablet and pressing send. Our back-end then runs the OCR on the hand-writing and creates a new sticky note. Users can then interact with these sticky notes using very simple hand-gestures.

This design simplifies our work by getting rid of the need to store static files in a storage bucket, create a dashboard, recognize two forms of each character, and detect complex hand-gestures.

2) OCR: For the OCR, the main way we reduced error was by focusing on getting the classification to work any amount, and then improving the classification accuracy from there. This was done both in preprocessing and in the neural net itself.

3) Gesture Detection: For Gesture Detection, we could make our work extremely complex by allowing complex backgrounds, creating complex gestures, and allowing multiple hands to work at once. In our most simple version, though, we considered having just two gestures (clench and tap) on a simple white background. To account for complex changing backgrounds, we added some background subtraction and enhanced our robustness and accuracy by introducing the use of a red foam in the user's hands.

4) Integration: For Integration, we could have a dashboard, store and display screenshots, and integrate G Suite tools. In our most simple version, though, we did not have any of that. Instead, we focused on setting up the barebones cloud infrastructure (REST API, Database, and Cloud ML Engine Instance), and design and implement the table UI.

VII. Related Work

There are two main competitors on the market: Jamboard by Google [4] and Surface Hub by Microsoft [6]. The tools are extremely similar: Jamboard integrates with GSuite Tools and Surface Hub integrates with Office Tools; Jamboard and Surface Hub both allow for more than one use to interact with the tool; Jamboard and Surface Hub both have web dashboards and web/mobile clients; and Jamboard and Surface Hub are both vertical touch-screens.

When analyzing these tools, we learned a lot from them and pulled from their strongest features. For example, we have GSuite integration, which allows for more then one person to interact, and even have a web dashboard. However, we differentiate from Jamboard and Surface Hub in two places: web and mobile clients and a vertical touch-screen. We decided not to create a web and mobile client for purely logistical reasons- we do not have nearly enough time to do so. The creative decisions came into play when deciding not to have a vertical touch-screen.

Jamboard and Surface Hub both have vertical touch-screens, but we decided to go with a horizontal screen that users interact with using hand-gestures. With this, we had two big design decisions: the horizontal display and the interactions. We decided to have a horizontal display because that's how people have been meeting for thousands of years. It's natural and it's agnostic to the professional field or culture. As for the interactions, we decided to go with hand-gesture based interactions for two reasons: cost and user experience.

Touch-screens of that size are about an order of magnitude more expensive than the camera we worked with, so this decision leads to a much cheaper product. As for the user experience, we decided it makes more sense to interact with the table as if there are objects on it, rather than as if it's a touch screen.

VIII. Summary

This section focuses on detailing a summary of what our project managed to achieve. It also touches upon any future work that could be incorporated into the project to make it more robust and successful. Finally, this section also details some lessons that we learned as a team as we worked on this project.

To summarize, we built a smart meeting table that focuses on improving collaboration, communication and record keeping in a meeting. Specifically, we built a table that one can interact with using hand gestures to perform basic tasks such as interact with widgets such as advancing through and manipulating meeting documents and taking screenshots. In addition, we built a robust and accurate segmentation and classification algorithm using a neural net, to allow users to write their own hand written notes during the meeting, which are then digitally embedded into the meeting notes. Finally, we implemented an easy to use and intuitive UI for users to manage these meetings and have a place to view and interact with all their notes.

There are a few things we would like to consider for future work. Firstly, with regards to the classification of letters and words, we realized we could improve our accuracy by basic methods such as an auto-correct API, and built in checks to account for common mistakes between the classification of letters. Had we had more time, I think these two simple steps would have significantly improved our classification accuracy. Secondly, after some research I believe that our hand gesture detection algorithm could have been more robust to changes in the background by focusing on CV algorithms based on feature detection and extraction. If we were to work on this in the future, we would work on improving the accuracy and robustness of our hand gesture detection, by using feature based detection and maybe even incorporating some ML techniques such as a convolutional neural network.

There are a lot of things we learnt, both individually and as a team, as we worked on this project. In this report we focus on some of the msot crucial things. Firstly, with regards to the OCR we learnt that pre-processing images can significantly impact the accuracy of classification. Before fine tuning our pre-processing we were getting an accuracy of about $40-50\%$, and simple changes in how we handled the pre-processing immediately bumped us up to over 80% accuracy. Secondly, we learnt that background noise in a computer vision based algorithm can significantly impact accuracy and is tricky to deal with. We ended up having to modify our approach to background detection by introducing the red foam to increase accuracy, which is not ideal. Thirdly, manufacturing a table can take longer than we thought initially. Finally, we learnt that we should have left more time to integrate with our hardware, especially the Raspberry Pi's that we used. Things that we assumed would be simple, such as boot up scripts, and installing modules on the Raspberry Pi, turned out to be significantly more challenging than we had anticipated - leaving a lot of integration to be done in the last week or so.

REFERENCES

[1] Amelia, Amelia. Convolution Neural Network to Solve Letter Recognition Problem. Convolution Neural Network to Solve Letter Recognition Problem, users.cecs.anu.edu.au/ Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_190.pdf.

[2] Erkmen, Burcu, and Tulay Yildirim. Statistical Neural Network Based Classifiers for Letter Recognition. SpringerLink, Springer, 1 Jan. 1970, link.springer.com/chapter/10.1007/978-3-540-37258-5_140.

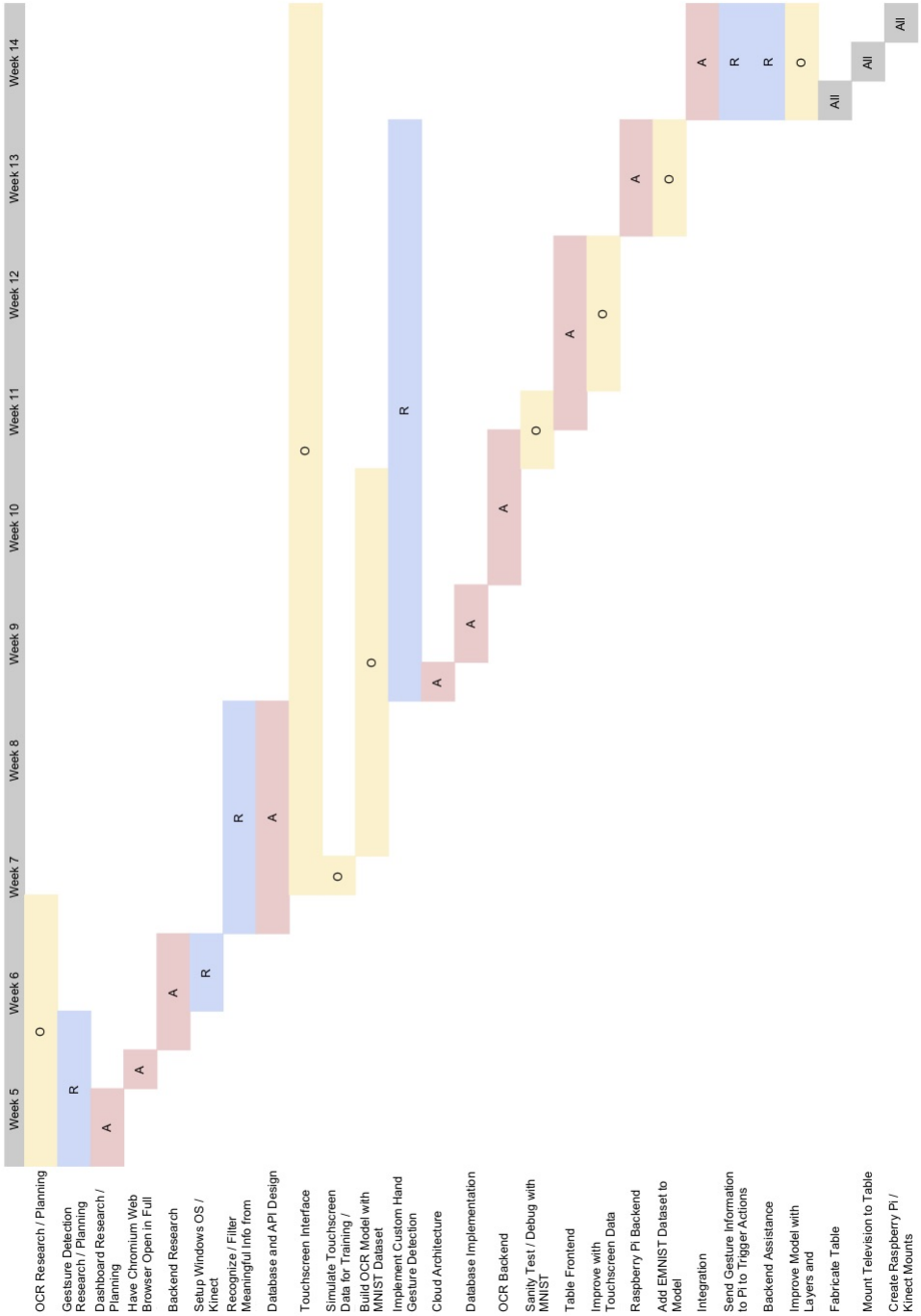
[3] Geitgey, Adam, and Adam Geitgey. Machine Learning Is Fun! Part 3: Deep Learning and Convolutional Neural Networks. Medium.com, Medium, 13 June 2016, medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721.

[4] Google, Google, gsuite.google.com/products/jamboard/.

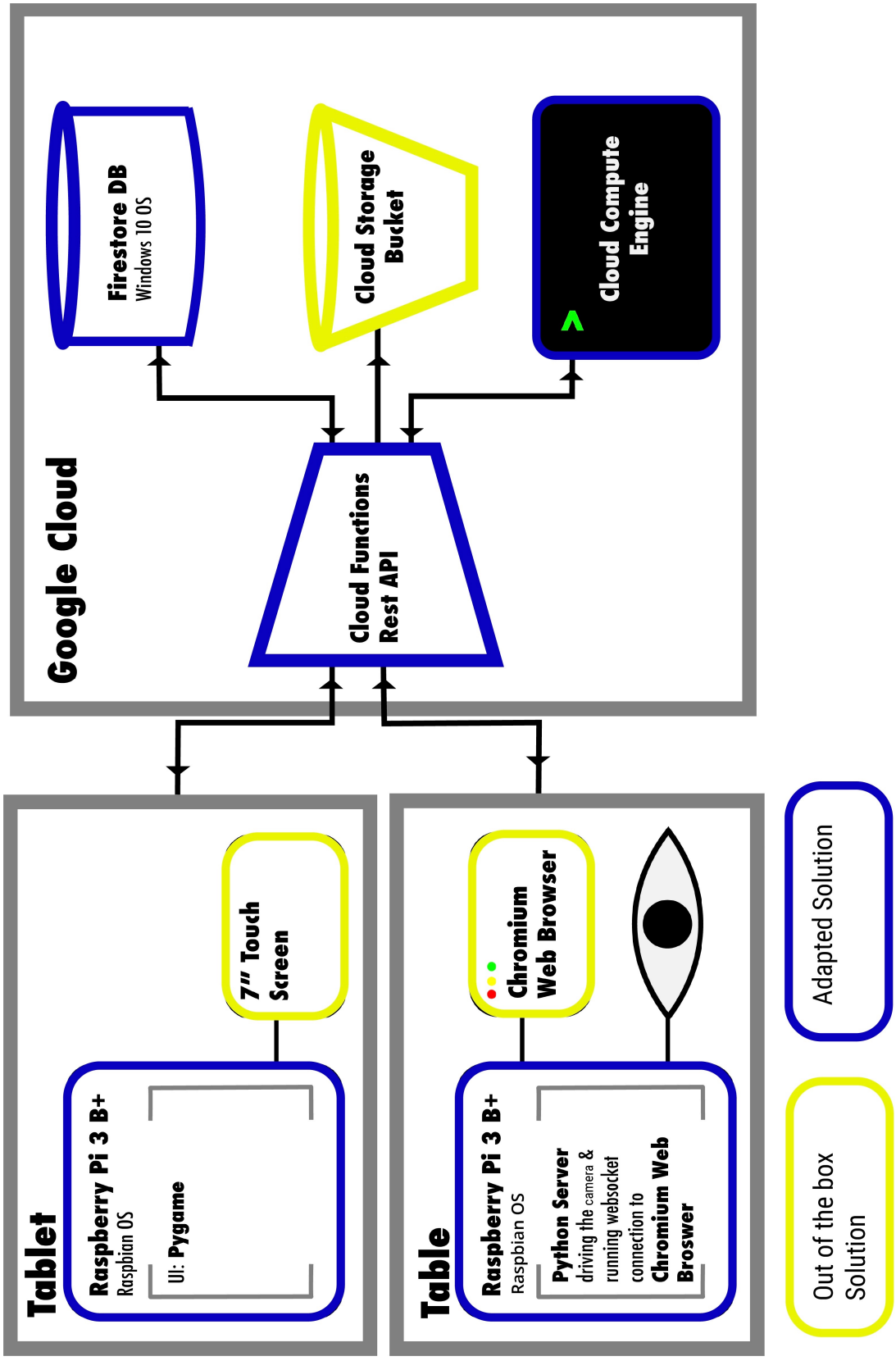
[5] Kandaswamy, Chetak, et al. Improving Classication Accuracy of Deep Neural Networksby Transferring Features from a Different Distribution. 2014, Improving Classication Accuracy of Deep Neural Networksby Transferring Features from a Different Distribution, www.researchgate.net/publication/269392865_Improving_Classification_Accuracy_of_Deep_Neural_Networks_by_Transferring_Features_from_a_Different_Distribution.

[6] Microsoft Surface Hub. Software Asset Management Microsoft SAM, www.microsoft.com/en-us/surface/business/surface-hub.

APPENDIX A
SCHEDULE



APPENDIX B
SYSTEM ARCHITECTURE - BLOCK DIAGRAM



APPENDIX C
BUDGET & PARTS

Part	**Cost per unit**	**Quantity**	**Total cost**
Raspberry Pi 3 B+	$38.10	2	$76.20
Raspberry Pi 7” Touch Screen	$74.00	1	$74.00
Logitech HD Web Camera	$19.90	1	$19.90
Asus 42” TV Screen	$200.00	1	$200.00
Fabrication cost	$50.00	1	$50.00
Total	-	-	$420.10