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 directly write on the table, converting their handwriting into text and digitally embedding it into native Google Drive apps 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 will facilitate collaborative meetings by having an interactive dashboard capable of displaying common Google tools, such as Google Slides and Google Docs. The table will also have a notes widget which will record and display notes that a user takes. The user will be able to use hand gestures to move and resize these widgets, and to take a screenshot of the current table display. Using these tools, meeting collaborators could display and interact with each others work, making comments and keeping track of their progress along the way.

Finally, there will be a touchscreen component of the smart table, which will serve multiple purposes. The touchscreen will act as the interface to the notes widget displayed on the smart table dashboard. The user will be able to write their notes on the touchscreen and, using OCR, our table will translate the handwriting into text and send this text to the notes widget. That way, users can take notes as they usually would on a notepad, but the notes can then be manipulated and saved. The touchscreen will also take 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 will 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 aim to implement, and then discuss metrics that we will use to test and validate our achievements with respect to the requirements. We also detail the requirements from the hardware and software we will use.

A. Gesture Detection

- The average time taken to detect and correctly classify a hand gesture should be 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 will manually test and measure the *gesture detection latency* over all the supported hand gestures, with a required 9 out of 10 successful trials.
- 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. In general, the targeted accuracy is a little lower due to the significant challenge presented in filtering out the image background. Please refer to the following table for more specific information.

Gesture Type	Target Accuracy	Function
Clap	85%	Screenshot
Clench	90%	Pickup & Drop widget
Swipe	90%	Move widget
Two-finger tap	85%	Select widget

Table 1: Different gestures and their targeted accuracy

B. OCR Segmentation and Classification

- Because we are performing OCR on characters written on a touchscreen, our letter segmentation should be 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 can create a bounding box for each letter based on when the user places and picks up the pen. There exists a small amount of segmentation error, for reasons such as overlapping letters, but otherwise our segmentation is ideal. Therefore, we expect to be able to segment letters with 97% accuracy.
- 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. Although the touchscreen should provide reliable information about this spacing, it is possible a user's

handwriting could have words spaced too closely together. Because of this, we will account have two separate requirements, dependent on the user's handwriting:

1) Words are spaced farther apart than letters. In this case, we require 100% accuracy. This case should be ideal.
2) Words are spaced closer together than letters, equally close, or have variable spacing. In this case, we will use the Bing Spell Check API to separate words. The success of this method will be shown with testing, but we will require any word separation method to have an accuracy higher than 90%.

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

Gesture Type	Target Accuracy
Letter Segmentation	97%
Word Segmentation	100%
Classification	92%

Table 2: Different OCR components and their targeted 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 will allow a user to access all their Google Slides, Google Docs and Notes through the Google Cloud API with 100% accuracy.
- Users should be able to also present their slide decks or notes as in a traditional meeting setting. To achieve this, we will support a presenter mode, allowing a user to directly plug in their laptop to the screen that is a part of the table.
- 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 will conduct a *user study* inviting different people such as students and professors 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 will be using a TV screen, which will be mounted onto a table. This TV screen will be used to display the web dashboard in a Chromium browser. The table will be controlled using a Raspberry Pi 3 B+, which will run the web dashboard. The Raspberry Pi 3 B+ will also be connected to a camera mounted on top of the table, through which it will receive a continuous video stream of the table surface. A lightweight Tornado based web server, written in Python, will receive 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 will interact with the table. Specifically, the user will be able to control what is displayed on the table through the touch screen and also be able to write notes onto the touchscreen, which will then trigger 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 to integrate easily with Google Drive tools, store user slide decks and docs and parallelize the training and testing of our neural networks.

D. Web dashboard

This web dashboard will be the main display on the table. It will run in a Chromium browser and host all widgets that the user interacts with such as the Notes widget, Google Docs and Google Slides widget.

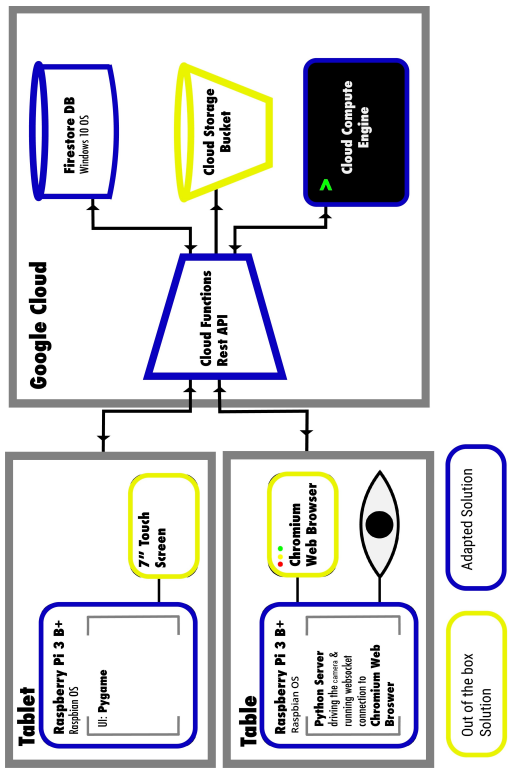


Fig. 1. Mesa - Overall System Architecture

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 screenshot 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 an Intel Depth Camera, which serves both as a camera and as a depth sensor. This allows 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. This information will be combined with the depth sensor data to accurately track the finger tips and use that as the basis for hand gesture recognition.

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 an Intel Realsense Depth Camera since it provides both a video stream and depth information - providing a perfect replacement for the Kinect.

To interface with the Raspberry Pi Touch screen we decided to implement a UI in Pygame in Python, as we found an intuitive device driver written in Python to directly interface with the touch screen. This allows us 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 will write notes on a Raspberry Pi 7 inch touchscreen display. These handwritten notes will be shown in Pygame, which will run on the touchscreen Raspberry Pi. The user will be able to 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 will send an image of the Pygame screen, along with additional information, to our Cloud Functions endpoint and redirected to our Cloud ML Engine Instance. This instance will be running a neural network in Tensorflow to perform OCR on the handwritten text. The results of this OCR will be saved in the Firestore DB, which will be 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 will be relatively simple. Since the user is writing on a touchscreen, segmentation information can be saved 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 will keep track of the minimum and maximum X and Y coordinates seen so far. Once the user stops touching the screen, this information will becoming the bounding box of that letter, and these bounding boxes will be sent along with the writing itself to the Cloud Compute Engine for classification. The letter order will be maintained using the X coordinates of the bounding boxes. Ideally, word separation will also be maintained by averaging the distances between each letters bounding box and assuming spaces significantly larger than that indicate a word end.

Text classification will be the challenging aspect of the handwriting OCR. However, since the letters will be so perfectly segmented, the classification can be 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 will always be at the center of their bounding box, so a regular neural network will accomplish the same task. Our planned neural network will have three layers: an input layer, a hidden layer, and an output layer. The input layer will have 28 x 28 = 784 neurons. The hidden layer will be the most dependent on experimentation, both in size and in activation function. The activation function will either be Sigmoid or ReLU, depending on performance. Finally, there will be a Softmax output layer will have 62 neurons (10 digits + 26 lowercase letters + 26 uppercase letters).

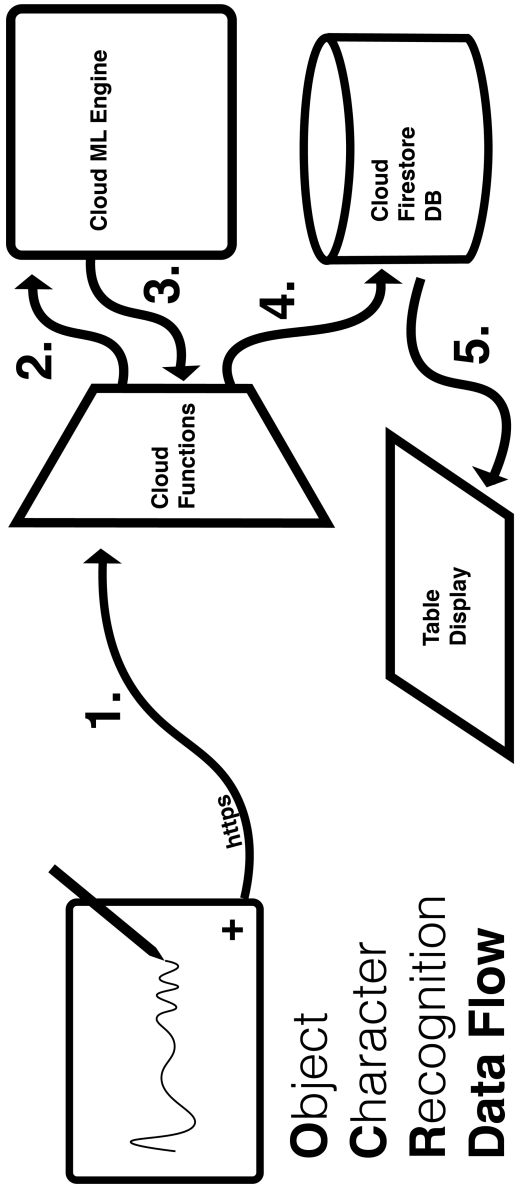


Fig. 2. Mesa - Touchscreen Data Flow

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 create 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 will be made to the Cloud Functions

REST API (see arrow 1). The Cloud Functions API will redirect this Request Body to the Cloud ML Engine Instance (see arrow 2) which will run the OCR and send a response to the Cloud Functions API (see arrow 3). This response will finally be 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 will poll every 10 seconds (see arrow 5) to refresh the display with any updated Notes objects. The granular user input flow is also fairly similar. A user will input information (for example, they will select a Google Slides Deck to display). This information will be bundled in an HTTP Request (see arrow 1), and stored in the Firestore Database (see arrow 4). The table will be polling for updates to this information every 10 seconds (see arrow 5) and will display the updated data as it is changed.

B. Interactive Table Implementation

This subsystem consists of the table display itself, along with the depth sensing 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 hand gesture detection, we have decided to use a purely computer vision based approach. We will have a hand gesture detection script, which is run as part of the a local Tornado based Python web server. This hand gesture script will be implemented in Python and use OpenCV for primitive image manipulation. The algorithm to detect hand gestures begins by first extracting the current frame and blurring it using a Gaussian Kernel. The blurred 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 plan on implementing. Here we describe the general approach we will follow to detect a few gestures:

A clench can be detected easily 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 can be detected by checking the number of convex hull vertices that are present along with a change in the average depth of the hand. Finally, a drag or swipe motion can be detected by comparing the change in (x,y) co-ordinates of the hand over successive image frames.

A challenge that we will face in isolating the contour around the hand would be ”noise” from the display of the screen itself. We aim to tackle this by taking a screenshot of the table display (just the background) and subtracting the pixel values from the image taken by the camera to easily isolate just the hands in the photo.

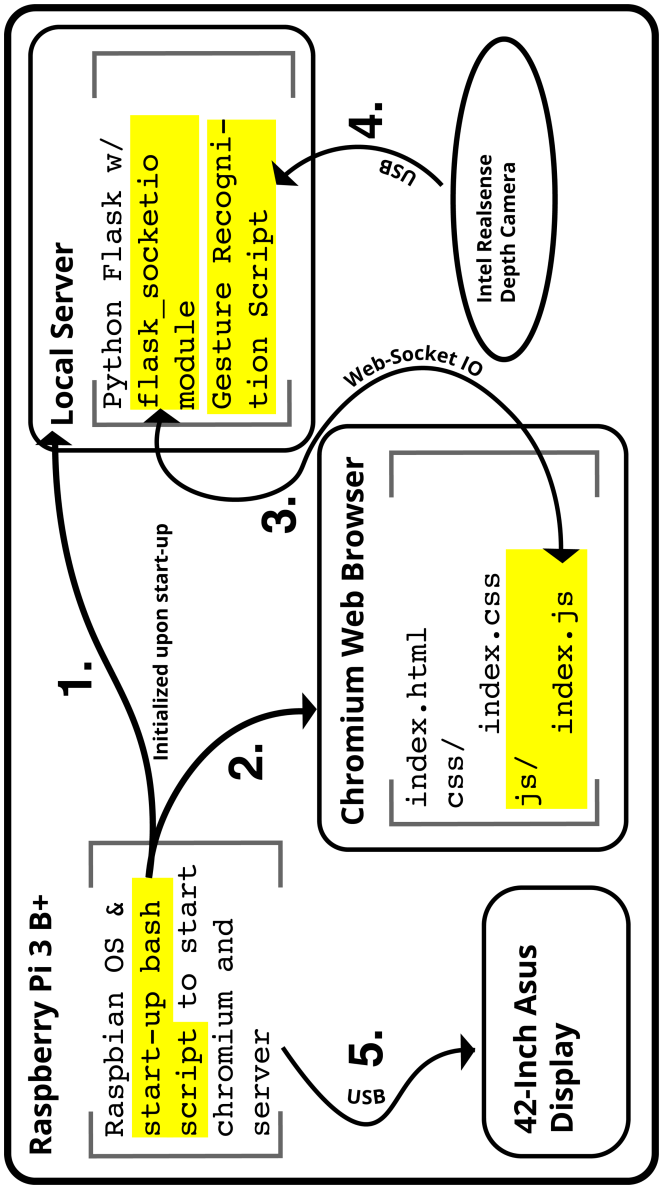


Fig. 3. Mesa - Gesture Detection Data Flow

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 will then establish 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 will interface with the Intel Realsense Depth 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 will then call the appropriate Google Cloud API endpoint function to indicate that a certain gesture was detected to trigger the appropriate action. This change will also be communicated to the Chromium browser

running in the display to reflect the change on the table screen itself. The Chromium web browser will be displayed on the screen itself through a direct wired USB connection with the Raspberry Pi. (refer to arrow 5)

C. Cloud System Implementation

Every device will be communicating with the back-end through a REST API. This API will be hosted on Cloud Functions, running a Python 3 run-time on a Flask Web Server. This Cloud Functions will host a single endpoint from which we will manually dispatch to the other desired endpoints within the same server. We will also be using a Cloud Firestore database to store non-file persistent data and a Cloud Storage Bucket to store files. Lastly, we will be using the Cloud ML Engine to run our OCR Model.

Using the Python Client Libraries that Google Cloud offers, we will communicate with our Firestore database, our Cloud ML Engine Instance, and our Cloud Storage Bucket all from our Cloud Functions server. This will allow us to have the one single endpoint to communicate with our entire back-end. Also, since they're all in cloud, authentication is made far simpler.

D. Web Dashboard Implementation

The meeting table will be built using a Raspberry Pi 3 B+ running a Raspbian OS. It will use a television – laying horizontally – as the main display and will be powered by a wall outlet. Upon start-up, this Raspberry Pi will instantiate a chromium web-browser and a local Python web-server. The chromium web-browser will be running Kiosk Mode, allowing it to go full screen. Upon starting up, it will immediately request static files from our CDN, allowing it to render our table User Interface. The local web-server will be built using the Python Flask web framework. The server will be polling data from a webcam, and running a gesture-recognition script to draw conclusions about the user's interactions with the table. When the server decides what gestures have been performed, it will push the changes that need to be made to the UI using a websocket connection to our Chromium web browser.

The Chromium web-browser will also be in HTTP communication with our Cloud Functions server in Google Cloud. It will poll every 10 seconds to query the database for any new Notes that could have been made on our tablet. It will also use the Cloud functions server to communicate with our database when displaying GSuite widgets.

VI. Project Management

A. Schedule

For our project, we have a lot of overlapping parts. To ensure efficiency and productivity, though, we wanted to segment our work (see following section for description of our responsibilities). We did so fairly successfully, so we w er able to schedule lots of our work in parallel, without blocking each-other. Having said that, there are some aspects of the project that block each-other. For example, if Olivia were to successfully train her model locally, she'd then be ready to deploy it to ML Engine and integrate it with the tablet UI. If Arman hadn't created the REST API, though, this would be a huge blocker. As such, we put the development of the REST API as an initial task for Arman and put the OCR model deployment later on in Olivia's schedule.

See Appendix A for a full schedule.

B. Team Member Responsibilities

To break up tasks, weve divided our work into three overarching areas, each of which is assigned to a team-member. Olivia is primarily responsible for OCR, Raunak is primarily responsible for gesture recognition, and Arman is responsible for Integration.

As a part of OCR, Olivia will be working on training a Neural Network to do OCR, but will also be responsible for implementing the simple touch-screen UI to collect the users hand-writing. She will also be using the tablet to put hand-written characters in a bounding-box.

For gesture recognition, Raunak will be primarily responsible for writing software that detects hand-gestures using an RGB and IR camera. Raunak will also be responsible for feeding the results of his gesture recognition software to the Chromium Web-browser using web-sockets on Python Flask.

For Integration, Arman will be 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 will be designing and implementing 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've designed a of risk-management plan. This plan includes a *bare-bone*s MVP, as well as minimum deliverables for each of our respective responsibilities.

1) MVP: For our MVP, we've defined it as a table that displays Sticky Notes. Users can put new sticky notes on the table by writing with their fingers on a tablet and pressing send. Users can only write with upper-case english characters. Our back-end would then run OCR on the hand-writing and create 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 OCR, we can add complex features to increase accuracy (such as the previous character). We can also recognize upper-case and lower-case letters and all numbers. In our most simple MVP, though, we'll flatten a 28x28 image as the input to our model, and we'll be training a neural net with 26 outputs: the capital letters in the English alphabet.

3) Gesture Detection: For Gesture Detection, we can 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'll have two gestures (open/close palm and translate hand) on a simple white background.

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'll not have any of that. Instead, we'll set 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, allow 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 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 in's agnostic to professional field or culture. As for the interactions, we decided to go with hand-gestures 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'll be working with, so this decision will lead to a 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.

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.

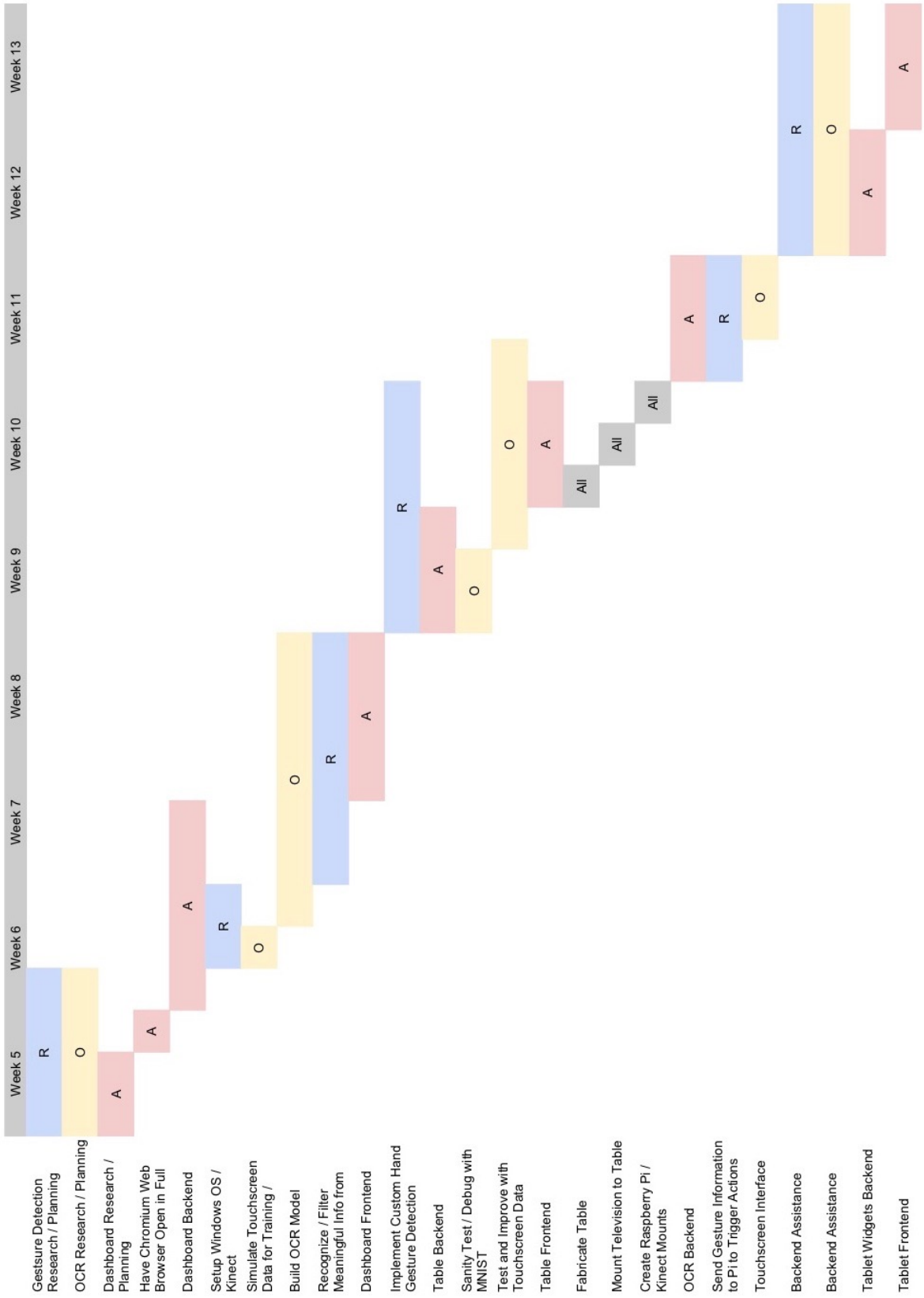
[3] Geitgey, Adam, and Adam Geitgey. Machine Learning Is Fun! Part 3: Deep Learning and Convolutional Neural Networks. Medium.com, Medium, 13 June 2016, medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721.

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

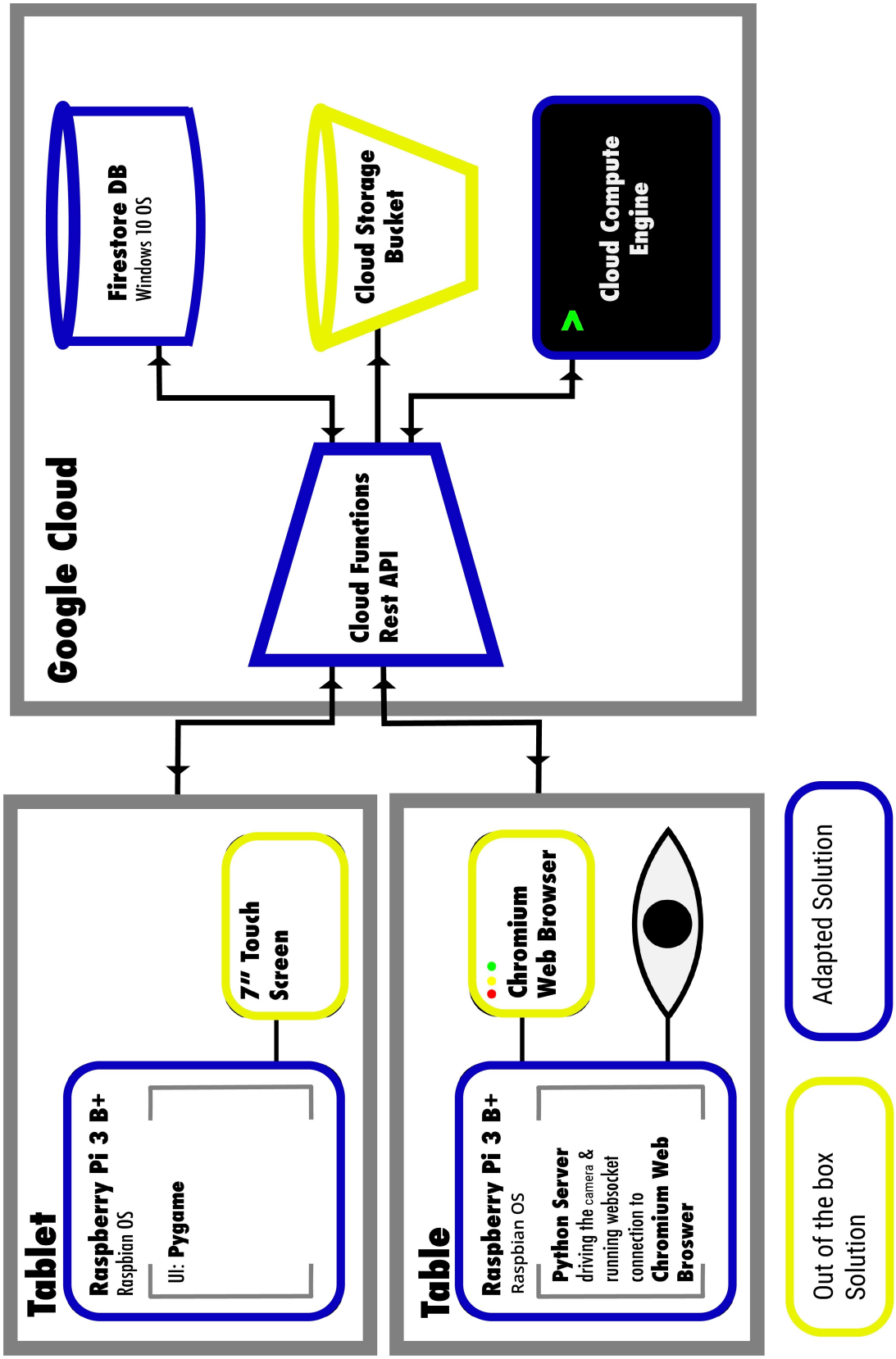
[5] Kandaswamy, Chetak, et al. Improving Classication Accuracy of Deep Neural Networksby Transferring Features from a Different Distribution. 2014, Improving Classication Accuracy of Deep Neural Networksby Transferring Features from a Different Distribution, www.researchgate.net/publication/269392865_Improving_Classification_Accuracy_of_Deep_Neural_Networks_by_Transferring_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
Intel Realsense Depth Camera	$199.00	1	$199.00
Asus 42” TV Screen	$200.00	1	$200.00
Fabrication cost	$50.00	1	$50.00
Total	-	-	$599.20