E7: PokerCam

Saisiddarth Domala, Jeremy Klotz, Ethan Rich
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A compact, standalone card shoe that images cards as they are dealt and provides a web interface to visualize the state of any card game.**

***Keywords*—Playing card image classification, card shoe, computer vision, machine learning**

1 INTRODUCTION

PokerCam automates the live analysis of card games by exactly determining the rank and suit of cards as they are dealt. To track card games, many casinos place multiple cameras around the table, and operators manually inspect the footage to identify each card player's hand. Our solution automates this process by imaging cards as they are dealt, providing real-time information about each player's hand. This product is targeted both at casinos tracking card hands to ensure game fidelity and casual competitions whose live, online spectators wish to follow along. This approach is cheaper than manually reviewing video cameras, both in equipment and labor costs. Furthermore, the card shoe will be approximately the same size as any other card shoe without adding any cables to the table.

PokerCam is completely self-contained. It is a modified card shoe with an Nvidia Jetson Nano and a printed circuit board holding the camera, lighting, and sensors that control the imaging system. This system must classify cards within 2 seconds with at least a 98% accuracy and no false classification triggers to accurately track the game state.

2 DESIGN REQUIREMENTS

2.1 Classification Accuracy

To properly track the state of a card game, the audience must have accurate information. In a single card deck of 52 cards, this system allows for exactly one misclassification. A single misclassification is easy to detect as the software can identify the two card labels classified as the same and the single card label that was never predicted. This allows the audience to localize the misclassification to a pair of cards having two labels.

We believe 98% classification accuracy is reasonable given recent work in image classification on playing card images. In 2011, cards were classified by rank and suit with 94.4% accuracy using lower-resolution images with varying warps and illumination [1]. Our system of imaging cards in a card shoe enables a more consistent pose and illumination with higher resolution images. This design will enable us to achieve a higher classification accuracy.

2.2 Classification Latency

To compete with systems that include live footage with human reviewers, we require the system to update the web display at most 2 seconds after dealing a card. This balances processing time on a low-power SBC with user demands for a responsive, real-time system. Furthermore, we require that the system classify an entire deck of 52 cards in less than 104 seconds (52×2). If the system implements a buffer, it must not sacrifice latency in classifying a continuously-dealt deck.

2.3 Card Retrieval Speed

The dealer must retrieve the card over a $\frac{1}{2}$ second period to ensure adequately sharp images given the camera framerate limitations. A Bicycle Standard card is 8.9×5.7 cm. (see Figure 3). Given that the camera will image only the top left corner of the card (see section 5.1), the suit and rank are approximately 0.6 cm. wide, and assuming constant card velocity v and shutter time T, we calculate $R_{overlap}$, the fraction of the card that overlaps due to motion-blur in a single capture:

$$v = \frac{5.7}{0.5} = 11.4\text{cm/ sec} \tag{1}$$

$$D = vT = \frac{11.4}{180} = 0.0633\text{cm} \tag{2}$$

$$R_{overlap} = \frac{D}{0.6} = 0.106 \tag{3}$$

With a $\frac{1}{2}$ second card retrieval time, the image capture is equivalent to convolving a sharp image of the region of interest (ROI) with a 1-dimensional box function whose width is 11% of the ROI's width. This upper-limit on the card retrieval speed balances the consumer need to quickly retrieve cards with the prohibitively expensive cost of high-framerate cameras.

2.4 Battery Life

A game of poker (or any other card game) lasts approximately two hours. We require a portable, rechargable power system to power *PokerCam* through a typical card game of at least two hours. A portable battery will sit inside of the card shoe to power the Jetson Nano for the required time without any external cables.

2.5 No False Triggers

Because there is no user-input to track which player receives a dealt card, it is critical that the system have

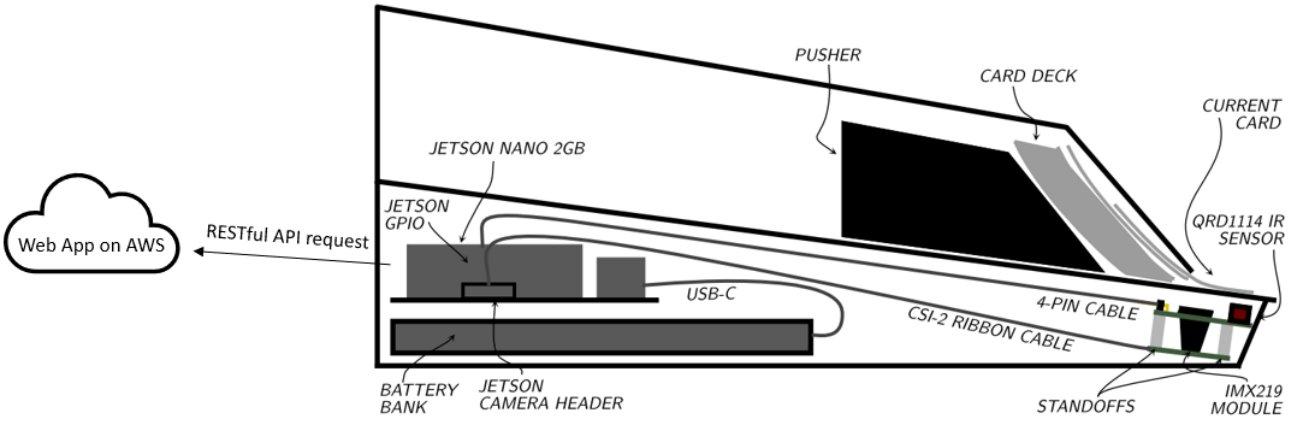

Figure 1: PokerCam: Augmented Card Shoe Diagram

no false triggers. Any false trigger would prematurely advance the system's game state, creating inaccurate reports of cards dealt afterward.

3 ARCHITECTURE OVERVIEW

3.1 Imaging System

Beneath the card shoe, the imaging stage consists of an off-the-shelf camera module and LED's mounted on a custom PCB enclosed in diffuse black acrylic to keep light out. The diffuse black acrylic walls will enable controlled and consistent lighting while avoiding specular reflections. The camera views the playing card from a small angle to reduce height requirements for the card shoe while maintaining the minimum object distance from the lens to remain in focus. The imaging system is triggered by an infrared reflection detector to trigger the imaging pipeline.

Once the card shoe detects a card moving over the imaging stage, the Nvidia Jetson Nano triggers continuous high-framerate captures from the camera. The camera sends the RAW captures over the camera serial interface (CSI) to the Jetson Nano. The Jetson Nano then stores the captures in memory and performs image demosaicing and preprocessing to identify the best capture and ROI for classification (see section 5.1).

3.2 Hardware System

The core of *PokerCam*'s hardware is an Nvidia Jetson Nano which sits underneath the rear side of the card shoe (see figure 1). It is powered by a rechargeable USB-C powerbank. The Jetson Nano controls the hardware and performs image classification. It includes a USB wireless internet interface to allow the Jetson Nano to perform web updates without a physical connection. This hardware system is completely portable as it relies on an internal battery and wireless interface.

An IR reflection sensor will be placed at the base of plate where the card exits the shoe. This sensor will determine when a card has left the shoe and will trigger the imaging and classification systems. The sensor is both an emitter and receiver for IR signals in one package.

The Jetson Nano is passively cooled by its heatsink. We do not anticipate adding active cooling since we will only draw intermittent full-loads during image preprocessing and classification. The excess heat caused by this process can easily be dissipated by the stock heatsink.

3.3 Software

The Jetson Nano stores a model trained offline to perform image classification. This model will be manually trained using existing python libraries (Scikit-learn and Pytorch). When an image arrives for classification from the imaging pipeline, the Jetson Nano executes the forward pass and uploads that classification result to the web application. This result is sent to the web application via a RESTful API request, and the web application updates a centralized MongoDB database containing the cards of each player. Visitors to the web application will then see the updated cards of each player. The web application and centralized database are stored on the cloud via AWS hosting services.

Python will be used to train the model, classify images, and constitute the back-end of our web application through the Flask framework. HTML and JavaScript will be utilized to create the front-end of our web application. The PyMongo module will be used by our Python code to make queries (getting, deleting, updating, and inserting data) to our MongoDB database. All Python, HTML, and JavaScript code will be developed, and cloud hosting services will be purchased to host our application.

4 DESIGN TRADE STUDIES

4.1 Imaging System

4.1.1 Lens Geometry

We chose a low-distortion, 75° field-of-view lens with an IR cut filter for the camera sensor. While we considered fish-eye lens that provide a significantly wider field-of-view, we chose to avoid that non-linear perspective since it would add complexity to training a classification model with non-linear projections. We also explored lens distortion correction algorithms, but the added complexity offline was not worth the benefits of having more images containing the ROI for classification. Furthermore, using a wider-angle lens reduces the resolution of the ROI.

The IR cut filter is critical to avoid any interference from infrared wavelengths transmitted from the card presence sensor.

4.1.2 Sensor Framerate and Resolution

Sensor framerate and resolution are conflicting parameters that both increase *PokerCam*'s price. To balance the need for high framerates to avoid motion-blur and high resolution to enable accurate classifications, we examined the trade-offs in the overlap ratio of a card image and pixel size. We selected a relatively cheap sensor (Sony IMX219) that provides 720p resolution at 180 fps. As we show in equation 5, 720p provides excellent resolution of the region of interest for classification. Therefore, we selected the highest possible framerate to enable fast card motions without significantly increasing *PokerCam*'s price by relying on specialty sensors.

The SBC's memory requirements increase linearly with the camera framerate in our current capture algorithm that saves every capture while the card is moving. Future improvements to *PokerCam* include a lower-power embedded processor to reduce the power and cost. To enable future memory-limited hardware, we limit our framerate to 180 fps to balance reasonable memory requirements with rapid user card motions.

4.1.3 Illumination

While multispectral illumination may enable improved discrimination between red and black suits based on reflectance values under different illuminants (hearts and diamonds v. clubs and spades), that configuration would require narrow-bandwidth laser diodes for each illuminant. We do not believe that is worth the added hardware to discriminate the card's rank from four options to two. Currently, we can accomplish a similar effect by examining red sensor values for black and red suits from the RGB camera. Furthuremore, multispectral illumination would require alternating images between two different illuminants. If the classifier requires images under both illuminations, this would halve the effective framerate, reducing the number of possible images to classify. Finally, any reliance on multispectral illumination would require future imaging systems to include a multispectral sensor. Since *PokerCam* aims to replace current surveillance systems with a cheaper alternative, we hope future iterations may include a monospectral camera that can classify images independent of the illumination choice. A monospectral camera would nullify any gains from multi-spectral illumination.

We will install multiple LED's around the PCB that holds the camera to illuminate the card. Those LED's will be appropriately spaced and angled from the sensor to avoid specular highlights. While we cannot comment on the spacing without finishing the prototype, we will include details in the final report of how the lighting geometry creates or avoids specular highlights.

4.2 Image Classification

For our machine learning classification algorithm, we will experiment with several Scikit-learn and PyTorch models to implement SVMs and neural networks. PyTorch utilizes CUDA-accelerated routines, and Scikit-learn utilizes vectorized NumPy calculations. These prebuilt optimized libraries allow us to achieve our latency requirement during classification without the added complication of building a machine learning system from scratch.

We will dedicate several weeks to experimenting with training these models. SVMs are relatively fast to train, but they may not provide the best classification accuracies. Fully connected neural networks have more weights, but they also have the potential to increase accuracy. Convolutional neural networks may achieve similar accuracies with fewer learned parameters. They require less computation when using small kernels and historically outperform SVMs on traditional computer-vision tasks.

4.3 Web Application

Our web application will be developed in Python, JavaScript, and HTML, and it will utilize the Flask web framework. The main other framework we were considering was Django. Both frameworks have high performance, so our decision does not bear influence over our latency user requirement. The main tradeoff we were analyzing was that Flask is minimalistic while Django has built-in packages. The development time with complex applications on Django might be faster because there might already be a package that handles the logic we need. However, we wanted a flexible framework to give us complete control over our web app's components. Having flexibility and complete control over our code would allow us to optimize our web application logic to most effectively reduce latency. This was the main reason we chose Flask as our web framework.

Another reason we chose Flask was that it allows us to have more control over which database to interface with. While it has built-in database software packages, Django has a limited number of SQL databases to choose from. We decided to use MongoDB as our database program due

to its unstructured schema, so it allows for higher flexibility with regard to our data format. Our data format will consist of which players have which cards, and this information's structure depends on which card game the users are playing: poker, euchre, blackjack, etc. Each game might have a different number of players and a different number of cards in each hand. If we were to use SQL, there would need to be a fixed, relational table containing this information. However, our information will not follow a structured schema and will not have a predetermined number of players and cards in each hand. This is why MongoDB will better suit our user needs of being able to play multiple card games.

MongoDB also consists of shared clusters, so data is replicated across nodes for high availability. This will help speed up our queries and reduce our overall latency. In addition, data replication provides fault tolerance, so our web app will remain functional for our users despite any failure within a MongoDB node. Since we want to make the web app's display a convenient, effortless experience for audience members, it is imperative that we not only limit latency but also prioritize fault tolerance. The final reason we chose MongoDB was because SQL databases are generally only scalable vertically. This can be expensive in terms of computation and memory. MongoDB utilizes low-cost commodity hardware to support horizontal scaling. Hence, MongoDB represents a more cost-effective decision as our centralized database.

5 SYSTEM DESCRIPTION

5.1 Imaging System

The imaging system will use an Arducam B0183 camera module with a Sony IMX219 sensor and a low-distortion M12 lens for capturing photos. We chose the IMX219 since the Nvidia Jetson Nano's kernel include prebuilt drivers and it provides excellent framerate at an adequate resolution (180fps @ 1280x720). The M12 lens also includes an IR cut filter to avoid any interference from adjacent IR sensor. The camera will connect to the Jetson Nano's MIPI CSI-2 connector.

This camera requires a 3cm minimum object distance. We will place the camera 3cm from the card, aimed at the top left corner of the card. Note that the position of the card's rank and suit is invariant to rotation when the dealer loads the cards into the shwo since the cards are symmetric when rotated 180°.

This camera module provides a $FOV = 75°$ horizontal field of view and $W = 1280$px horizontal resolution. Assuming the object is $D_{obj} = 30$mm away from the lens, we estimate the imaged target size D_{target} and pixel size P for a planar object parallel to the imaging plane:

$$D_{target} = 2D_{obj} \tan\left(\frac{FOV}{2}\right) = 46\,\text{mm} \quad (4)$$

$$P = \frac{D_{target}}{1280} = 0.0359\,\text{mm/px} \quad (5)$$

Given the ROI containing the card's suit and rank is 0.6×1.8 cm (see Figure 3), that corresponds to 167×500px ROI on the image. We believe this resolution is more than sufficient to enable the 98% classification accuracy requirement, especially with little motion blur due to the high framerate.

When the dealer retrieves a card from the shoe, it trips the card presence sensor. That triggers the camera to begin the 180 fps captures. Once the sensor is no longer triggered, the camera captures hault. At that point, the camera has captured approximately 90 frames over the 0.5 second period. This requires approximately $1280 * 720 * 90 * 4\text{bytes/px} = 316\text{MB}$ to store the uncompressed captures in system memory. The pipeline then determines the correct image on which to perform the classification. Given that the rank and suit appear in the first 0.6cm of the 5.7cm width and assuming the card moves at a constant velocity, we estimate that the images containing the rank and suit will appear in the first $\frac{0.6}{5.7} = 10.5\%$. We then select an image from that initial subset of captures to preprocess and classify. Based on the latency of our trained model, we will determine if there is headroom to perform classification on all 10 images and save the label with the highest certainty or randomly select a single image for classification. We will report this finding in the final report.

To preprocess the image before classification, we will identify the corners that surround the card suit and rank. The program will crop that ROI, resample it to a constant image size through bilinear interpolation, and pass it through the classifier.

The Nvidia Jetson Nano will compute the forward pass of our machine learning model to classify cards. As opposed to other popular SBC's, we chose the Jetson Nano because it includes more memory and CUDA cores for GPU-accelerated networks.

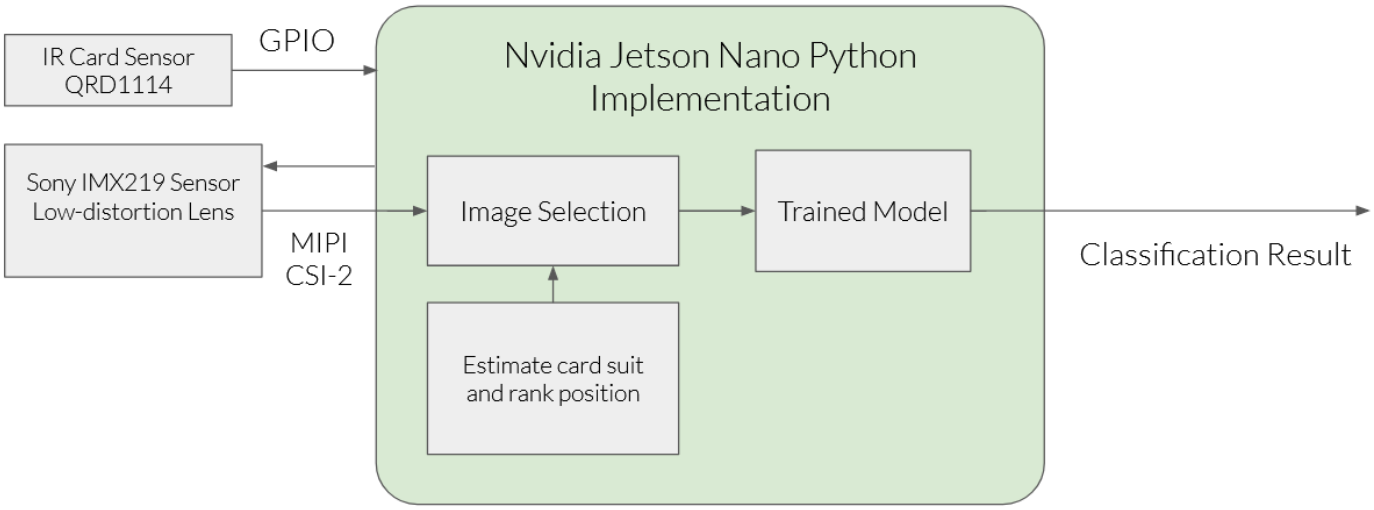

Figure 2: Imaging subsystem block diagram

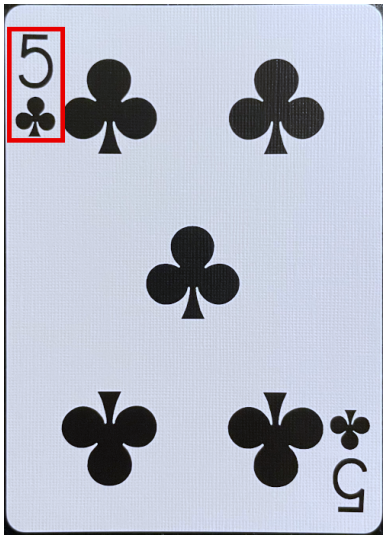

Figure 3: ROI of Bicycle Standard playing card contains suit and rank highlighted in the red box. The card size is approximately 5.7×8.9cm. The ROI size is approximately 0.6×1.8 cm. The card will be imaged as it moves from right to left.

5.2 Hardware System

Our hardware system (see figure 4) will consist of an Nvidia Jetson Nano with 2GB of memory with a USB Wifi module for over-the-air data transmission, an Arducam IMX219 camera module with a low-distortion M12 lens, a 10000mAh USB-C battery bank, a custom PCB with LED illumination, and an IR reflection sensor to detect card motion.

The custom PCB will include the lighting system and IR sensor. This PCB will be stacked on top of the camera module board using standoffs (modeled in Figure 4).

We will use the QRD1114 Reflective Object Sensor from Fairchild Semiconductor. It comes in a 4-pin through-hole package with pins 1/2 connected to a phototransistor and pins 3/4 connected to an IR LED.

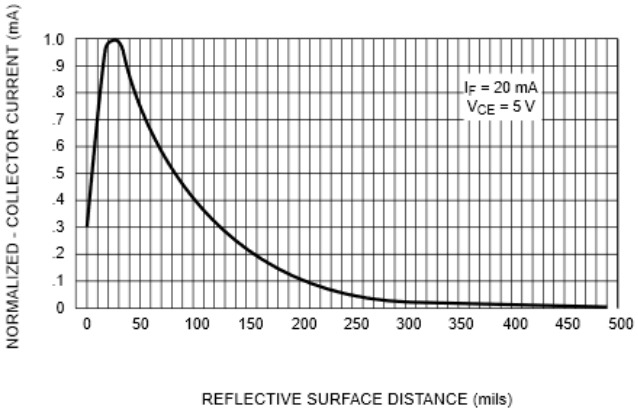

Figure 5: QRD1114 Normalized Collector Current vs. Distance [3]

Using this output current in conjunction with a resistor network (as shown below), we will convert this current into a voltage readable by the Jetson Nano's GPIO pins.

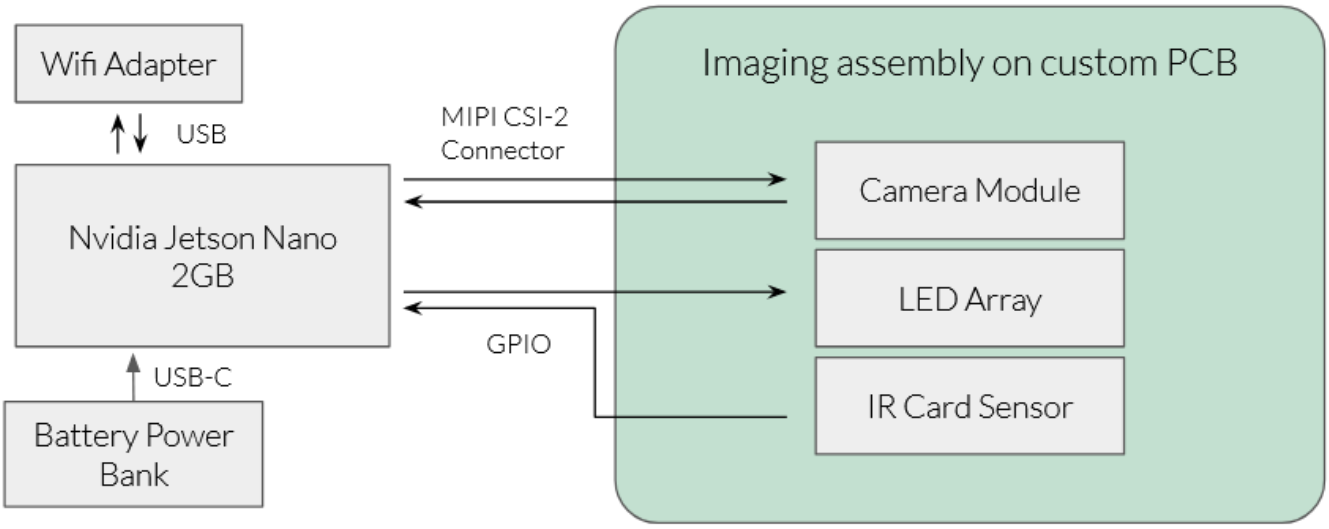

Figure 4: Hardware subsystem block diagram

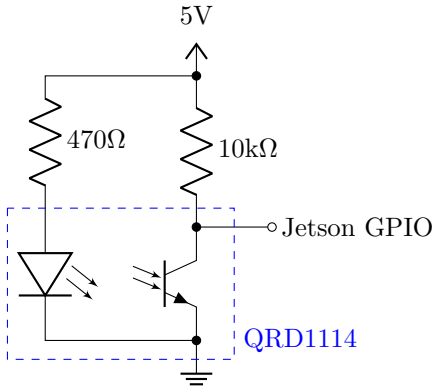

Figure 6: QRD1114 System Circuit

The battery bank we plan on using is a 10000mAh USB-C battery bank. It supplies 5.0V with a 2.1A max current limit. We estimate the battery life assuming the Jetson Nano draws the maximum current and completely drains the battery's listed capacity:

$$E_{batt} = VI \cdot \Delta t \tag{6}$$

$$= 5\text{V} \cdot 10000\text{mA h} \cdot \left(\frac{3600\text{ s}}{1\text{ hour}}\right) \tag{7}$$

$$= 180,000\text{J} \tag{8}$$

$$E_{jetson} = 5\text{V} \cdot 2.1\text{A} \cdot \Delta t \tag{9}$$

$$180,000\text{J} = 10.5 \cdot \Delta t \tag{10}$$

$$\Delta t = 17142.857\text{s} = \boxed{4.761\text{ hours}} \tag{11}$$

This is a conservative estimate. We only anticipate using full power during the image preprocessing and classification step. We aim to provide battery life for twice the length of a typical card game, having our device last approximately 4 hours. If we find the battery's banks usable capacity significantly differs from the listed capacity, we will transition to a 15000mAh battery bank to meet the technical requirements for battery life.

The LEDs we will be using are the Seoul Semiconductor S1W0-2835408003 [2]. They are a SMD-mounted 4000K white LED. The LEDs will be powered by the Jetson Nano's builtin 5V power supply rail and controlled by a GPIO pin through a NPN Bipolar Junction Transistor (BJT)

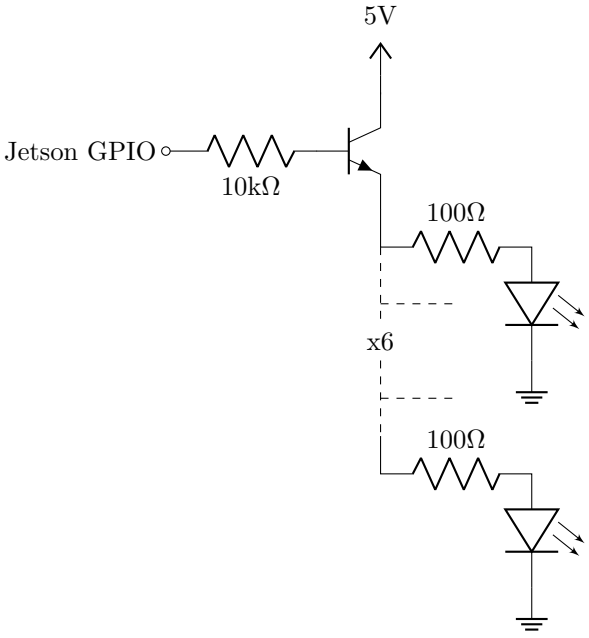

Figure 7: LED Array Circuit

The 100Ω resistors are current limiting resistors restricting the current to $I = \frac{V}{R} = \frac{5V}{100\Omega} = 50\text{mA}$. The max current ratings for the LEDs is 65mA. The maximum supply

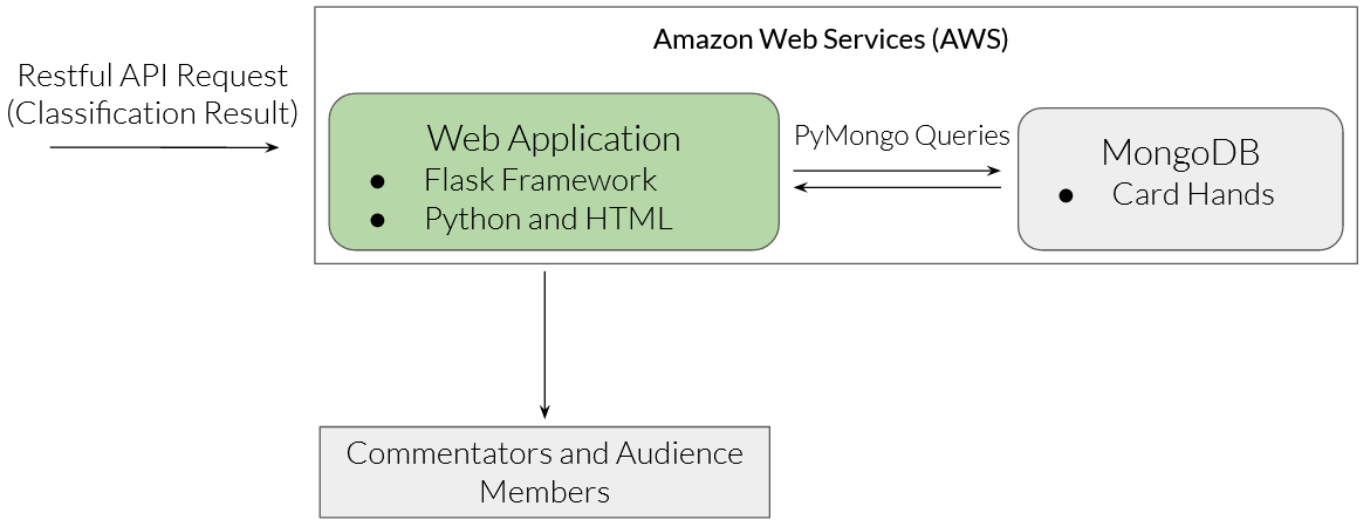

Figure 8: Software subsystem block diagram

current for the Jetson Nano's 5V rail is 500mA, thus our 6 LEDs consuming their maximum power will be using only 78% of this rating.

5.3 Software System

Our image classification algorithms will utilize a machine learning model implemented in an existing Python library. We will obtain training, validation, and testing data by manually imaging 4 decks of Bicycle Standard playing cards and noting their ground-truth labels. We will preprocess those images with the same pipeline that identifies and crops the ROI.

After obtaining the training data, we will train the following models on the preprocessed images: support vector machines, neural networks, and convolutional neural networks. We will use scikit-learn to train the SVMs and PyTorch to train GPU-accelerated neural networks offline on a PC. If necessary, each ROI will be resampled to a consistent size for training. SVMs with a radial basis function (RBF) kernel are faster to train and evaluate, but neural networks may learn a more complex decision-boundary that produces more accurate classifications. Using the model with the best validation accuracies, we will report the training, validation, and test accuracies in the final report and implement that model on the Jetson Nano.

Each model will predict 2 labels: 1 label for the rank and 1 label for the suit. We will train an ensemble of SVMs to predict these labels. The neural networks will output two vectors corresponding to label probabilities for the rank and suit. The overall loss function will be the sum of the binary cross-entropy losses for the rank and suit labels.

The chosen machine learning model will output its classification result in a RESTful API request to the web application (see figure 8). This POST request will contain information about the suit and rank of each card. Since the card dealing order is fixed in these card games, the web app will automatically determine which player the card belongs to based on the dealing order. It will then update its display of the cards of each player. The user will select the card game at the beginning of the session. This initial HTTP request with the game type will allow the web app to implement the correct state machine in determining dealer order.

The web page's back-end logic will be written in Python. It will work with the Flask framework to handle GET and POST requests. When handling these requests, the Python code will interface with a MongoDB database hosted on AWS. This will be accomplished through PyMongo, a Python distribution tool that allows Python code to make queries to a MongoDB database. When the Nvidia Jetson submits a POST request to the web page, the MongoDB database will be updated with which players have which cards. The web page, which automatically refreshes every 0.5 seconds (to meet our 2 second latency requirement), will then display the updated content. This updated content will be obtained by making a query to the MongoDB database to see the current state of the game. We need a centralized database to ensure all visitors to our web application view the same, consistent card data. Without a centralized database that is shared by all users, different visitors would see different data.

The web page's front-end logic will be written in HTML with inline JavaScript. The HTML web page will showcase a table consisting of players as rows and cards as columns. The inline JavaScript is necessary to make a new HTTP GET request to the web page every 0.5 seconds. This ensures the web page on the user's browser is displaying the most up-to-date information about the game's state without requiring the user to refresh the page.

6 VALIDATION PLAN

6.1 Subsystem Testing

6.1.1 Image Classification

Using the hardware prototype, we will image four unopened decks of Bicycle Standard playing cards as they are retrieved from the card shoe. Those images will constitute the training, validation, and testing sets. The test images will correspond to a single card deck exclusively for testing. We will train SVM and neural network models offline on a local PC. We will save the best performing model and upload it to the Jetson Nano. We will validate the model by examining classification accuracies on the validation set.

6.1.2 Hardware

The card presence IR sensor will need to be thoroughly tested to ensure there are no false/failed triggers. We will test this system by drawing cards from the shoe, making note of any false/failed triggers and adjusting the sensitivity/thresholds accordingly. We will also test the device in several common environments: under incandescent lighting, fluorescent lighting and outdoors in sunny conditions.

6.1.3 Web Application

The main testing for the web application is ensuring that users from various geographic locations can visit the web page. Since our web page will be ultimately configured on the cloud, we will have access to inbound security rules for our web application. We need to ensure that all IP addresses can successfully send GET requests to the web page. In addition, we will test the latency of these HTTP requests. Users of various geographic locations should be able to visit the web page and see updates within 2 seconds of the card being retrieved. To test the web app during development, we will send POST requests with player and card data to the web app to update the current state. This should hold true for all users regardless of their geographic location. If we observe delays in the web app for certain users, we will look into any AWS services providing content delivery networks. Since these services would help configure our web app on geographically distributed servers, latency could be greatly reduced for users. However, this can also incur higher costs, so it should only be purchased if necessary.

6.2 Full System Testing

To verify the product meets the design requirements, we will host a card game where the dealer exclusively deals a new, unopened deck of Bicycle Standard cards from *PokerCam* for at least two hours (the length of a typical card game). One of the team members will be present during the game to note ground-truth labels of cards as they are dealt. The team member will inform the dealer that *PokerCam* requires the user to retrieve a card over a 0.5 second period to enable accurate image classification. The team member will watch the web display to note any false triggers during the game. The system will be configured to automatically measure latency between the card trigger and classification result.

If the battery dies in less than two hours of gameplay, the system fails to meet the power requirements. After the game, we will review the classification results with the ground-truth labels to verify it achieves the accuracy requirement without any false triggers. We will also review latency data to verify the two-second latency is achieved for each classification.

7 PROJECT MANAGEMENT

7.1 Schedule

Our schedule includes an aggressive timeline to design the PCB to allow time for a second revision. Secondly, the schedule requires that Jeremy configure the imaging prototype quickly after the hardware arrives to enable experimentation with the lighting system and time to collect the training and testing set. This will require collaboration with Ethan, who is in charge of designing and configuring the hardware setup with the card shoe. Once Jeremy finishes the imaging prototype on the card shoe, he will hand it off to Sid to collect training, validation, and testing images from different unopened decks of playing cards for training. Sid will then work on training various ML models. After training and validation, he will choose the most optimal for our use case requirements to be stored on the Nvidia Jetson.

The schedule includes two weeks at the end for slack time and integration.

7.2 Team Member Responsibilities

Jeremy is primarily responsible for designing the imaging system. He will prototype the camera and lighting setup to provide the best images without specular highlights, motion blur, or defocus. He will implement the image preprocessing and selection algorithm. He will work with Sid on the machine learning image classification.

Sid is primarily responsible for the web application and machine learning. He will design and implement the software for the web app and work with Jeremy to collect training, validation, and testing photos to train appropriate models for classification.

Ethan is primarily responsible for the hardware setup. He chose sensors, will design the PCB, and will lead the physical design of the card shoe. He will assist with the imaging setup to include black acrylic walls to control the lighting.

7.3 Budget

Figure 10 contains our proposed budget. To work on the project remotely, we budget for three Nvidia Jetson Nanos

and appropriate accessories. We also purchased two camera modules to allow us to build two separate prototypes remotely. To recreate the project, one must only purchase a single Jetson Nano, one microSD card, one battery bank, one card shoe, and one PCB with the associate BOM.

All parts are purchased using our capstone budget, and none are borrowed from the course inventory.

7.4 Risk Management

Our most critical risk is achieving the 98% accuracy requirement while obtaining less than 2 seconds in latency. To reduce motion blur, we are working with a high-framerate camera. It is very possible that the high-framerate captures will overwhelm the Nvidia Jetson Nano's system memory if there is more capture overhead we have not accounted for. If that occurs, we will either drop frames or purchase a Jetson Nano with 4GB of system memory instead of 2GB. If we find that we can only achieve the classification accuracy with complex classification models or expensive preprocessing, we may sacrifice latency to achieve the desired accuracy.

Secondly, the image selection process prior to classification is risky at this stage since we cannot determine the number of images sufficient for classification before building the prototype. Similar to the memory overflow scenario, we may have to sacrifice latency to implement more complex algorithms to identify the image that classifies the card with the highest certainty.

The long shipping and turnaround time for the hardware is risky. We plan to mitigate this risk by manufacturing the first revision of the PCB quickly to allow for a second revision. In addition, we will build two prototypes to limit the number of hardware exchanges between teammates and dependencies on a single part.

7.5 AWS Credit Usage

Our AWS credits were used to configure an Ubuntu virtual machine through the EC2 web service. Located in northeastern US, our server currently hosts our web application. As of now, we have not yet utilized any of the AWS credits, so our bill is currently $0. We anticipate to spend $3 a month for as long as the web application and server are running. Thank you for providing us with AWS credits.

References

[1] Paulo Martins, Luís Paulo Reis, and Luís Teófilo. "Poker Vision: Playing Cards and Chips Identification Base on Image Processing". In: *Pattern Recognition and Image Analysis* (June 2011), pp. 436 –443.

[2] *Mid-Power LED: 3528 Series*. Datasheet. Rev 1.2. Seoul Semiconductor, 2020. URL: https://www.digikey.com/en/products/detail/seoul-semiconductor-inc/S1W0-2835408003-0000003S-0P002/9686996.

[3] *Reflective Object Sensor, QRD1113, QRD1114*. Datasheet. Rev. 3. Semiconductor Components Industries, LLC. 2019. URL: https://www.onsemi.com/support/design-resources/datasheets?part=QRD1114/D.

A Appendix

A.1 Project Timeline

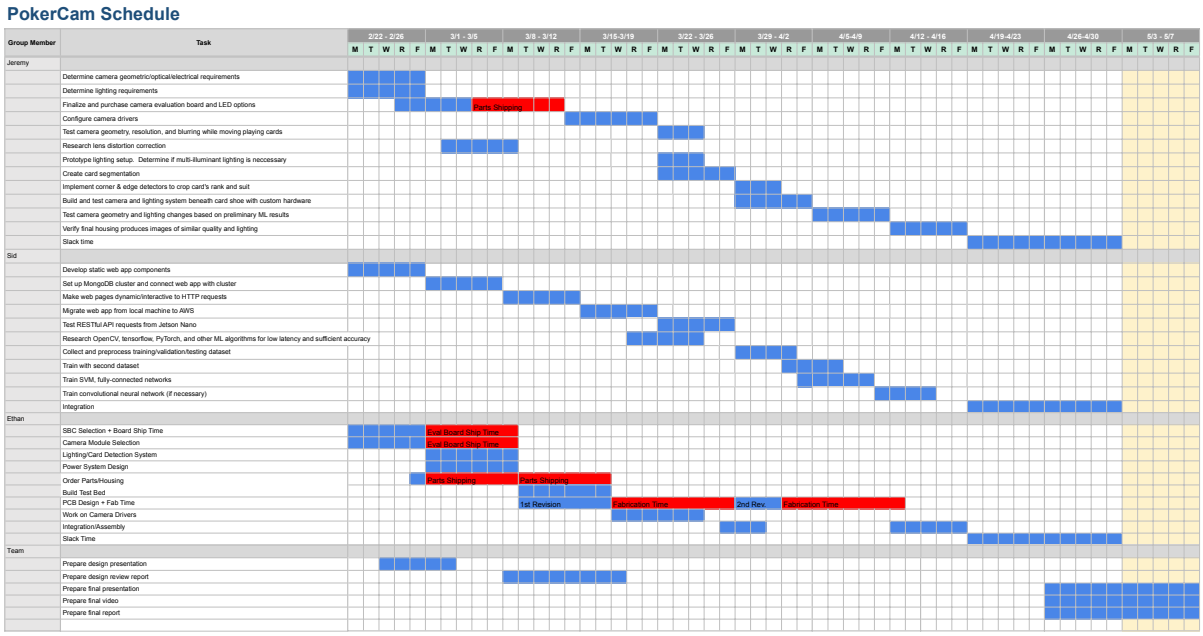

Figure 9: Gantt Chart detailing individual tasks and contributions

A.2 Proposed Budget

Item	Part No.	Vendor	Quantity	Unit Price	Price
Nvidia Jetson Nano 2GB	-	Amazon	3	$49.99	$149.97
32GB MicroSD card 3-pack	-	Amazon	1	$16.99	$16.99
5V 4A DC Power Supply*	-	Amazon	2	$14.95	$29.90
10000mAh USB-C batterybank	-	Amazon	2	$16.00	$32.00
Arducam OV9281 Camera Module*	B0165	Amazon	1	$42.00	$42.00
Arducam IMX219 Camera Module	B0183	Amazon	1	$35.99	$35.99
Brybelly Six Deck Blackjack Dealing Shoe	-	Amazon	3	$16.99	$50.97
Bicycle Standard card deck (8pk)*	-	Amazon	1	$24.00	$24.00
AWS credits	-	Tamal	10	$0.00	$0.00
PCB Rev. 1	-	OSHPark	1	$20.00	$20.00
PCB Rev. 2	-	OSHPark	1	$20.00	$20.00
IR Reflector Sensor	QRD1114	Sparkfun	2	$0.95	$1.90
SMD LED 3528 White, Neutral 4000K	S1W0-283540800	Digikey	12	$0.18	$2.16
SMD NPN BJT TO-236AB	PMBT2222A,235	Digikey	2	$0.50	$1.00
100R 1W SMD Chip Resistor	3520100RJT	Digikey	12	$0.33	$3.95
10K SMD Chip Resistor	352010KJT	Digikey	2	$0.46	$0.92
Shipping (Estimate)	-	-	-	$15	$15.00
				Total	$446.75

**Only used for prototyping*

Figure 10: Proposed Budget