6-11-2024

Hiv3: An Efficient Beehive Monitoring System

Jack Ursillo

Aneal Kuverji

Connor Merhab

Anshuman Sahu

SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 7, 2024

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Jack Ursillo
Aneal Kuverji
Connor Merhab
Anshuman Sahu

ENTITLED

Hiv3: An Efficient Beehive Monitoring System

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

DocuSigned by:
Behnam Dezfouli
6ED175D6B91A4FF...
Thesis Advisor

DocuSigned by:
Shiva Jahangiri
46FB5AED309540B...
Thesis Advisor

Department Chair

Hiv3: An Efficient Beehive Monitoring System

by

Jack Ursillo
Aneal Kuverji
Connor Merhab
Anshuman Sahu

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 11, 2024

Hiv3: An Efficient Beehive Monitoring System

Jack Ursillo
Aneal Kuverji
Connor Merhab
Anshuman Sahu

Department of Computer Science and Engineering
Santa Clara University
June 11, 2024

ABSTRACT

Beehive monitoring plays a major role in ensuring the health of beehives by checking for overpopulation or underpopulation within a hive. Beehive monitoring provides beekeepers with the opportunity to take action and save the hive before the problem becomes irreversible. Most solutions are too expensive for everyday beekeepers and lack elements of sustainability, making it impractical for small scale beekeepers. In this thesis, we propose a solution to this problem, demonstrating its sustainability and user-friendliness, which enables us to effectively reach a larger consumer market. We support these claims through the use of sustainable systems such as using a solar panel coupled with a rechargeable battery and incorporating deep-sleep capabilities into the system's low-power embedded system (ESP32) which is connected to a camera. The ESP32 sends images to a Raspberry Pi, which performs image processing using a machine learning model and transmits the processed images to the cloud. We present a system architecture diagram describing how these systems are integrated as well as how other measures, such as security and single sign-on, are implemented to ensure the integrity of the solution. The system tests conducted in the field show that the machine learning model yields a mean average precision (MAP) score of 52.2, compared to the benchmark score of 53.7, ensuring accurate, real-time monitoring utilizing a low-power system.

Table of Contents

List of Figures

Chapter 1

Introduction

Bees play a vital role in our ecosystem, since they are essential for pollinating many crops that we consume on a daily basis. In 2018, 1.8 million bee colonies were shipped to California to pollinate various crops; yet, during transit around 45% of the colonies were lost [1]. The majority of these losses can be attributed to the poor health condition of the colonies, as the state of a beehive is a large factor in how productive a colony will be. In order for beekeepers to maintain a colony's health, the beekeeper must keep track of the current population of each hive and frequently check bees for varroa mites. A hive's population can fluctuate rapidly in the spring and summer seasons, as numbers can increase from 20,000 to 60,000 bees [2]. If the hive's population is left unchecked, overpopulation can lead to swarming, where an estimated half of the hive, including the queen, leaves in search of a new one. This sudden abandonment leaves the current hive depleted of its resources and increases the risk of disease. Current methods to prevent swarming involve weighing the hive, counting the number of unhatched eggs, and analyzing the brood pattern. These solutions vary considerably in accuracy and will not always give the beekeeper a proper count of the hive. The proposed solution removes the variability present in prior solutions by keeping track of the number of bees entering and exiting the hive, providing the beekeeper with real-time updates which can be viewed on the web app. We aim to improve this project by further by updating the software, implementing new hardware, and introducing a solar power energy system. By incorporating these improvements, we believe that we can cut both latency and energy costs of the system leading to a better user experience.

1.1 Problem Statement

There is a wide array of solutions to assist in a beehive's overall health; however, many solutions are impractical due to the expense or lack of efficiency. We intend to remedy most of these issues, as we aim to be a small compact system placed near the entrance of the hive. With the camera being one of the main components the overall expense of adding more systems is much cheaper compared to other potential solutions, as a customer would only need to purchase another camera, solar panel, and battery. This camera would be able to count the number of bees entering

and exiting the hive using an energy-efficient and reliable communication system. The data collected from the camera will be sent over the cloud to the managing Raspberry Pi and the database. The data can be easily accessed through the web application. We will leverage data analytics to evaluate how many bees are present in the hive during the day and track the population of the hive over a period of time.

1.2 Background

As a continuation project, we would like to mention last year's members who worked on this project. While visiting the apiary, we noticed that the monitoring system had a power cord running from the garage outside, and we thought about the system's feasibility if it was in a practical field. We drew influence from this and decided that the system should be energy efficient and sustainable through the use of solar energy. After speaking to the client, we received feedback about designs and quality of life updates for the system. The objective is to create an energy-efficient external monitoring system. The first step we needed to understand was the power consumption and capabilities of the ESP32 camera with a sustainable power supply that consists of a solar panel and external battery. Then we would need to figure out how to relay data to the machine learning model and find a way to store it for later use. Subsequently, we needed to figure out how a user is able to visualize the data without impairing the quality of the live stream, while being able to modify the camera settings.

1.2.1 External Beehive Monitoring System

We want to acknowledge a current competitor like Eyesonhives [3], as we were able to create use cases with an external monitoring system. Eyesonhives is a solar powered external monitoring system that analyzes and makes an inference on the supervised traffic. This inference can be one of four patterns which has been built on over 7 Terabytes of data. The traffic data and inference are relayed over to the cloud, allowing an authorized user remote access to their information through a mobile application or a website.

1.2.2 Internal Beehive Monitoring System

Another competitor we would like to acknowledge is Broodminder [4]. Broodminder's internal beehive monitoring system measures the internal temperature and humidity with an accuracy of 1°F which is used to determine the health activity of the hive. Broodminder uses Bluetooth Low Energy to transmit data to the cloud free of charge and users can access this data on their mobile application.

1.3 Approach

While researching the ESP32 camera, we found an open-source camera web server which is capable of hosting a live stream, providing the user the ability to manipulate the camera specifications, and capturing images with minimal

interference to the live stream. This example is an over the air (OTA) camera web server [5], which allows the camera to connect locally to the internet by using a local internet protocol(IP) to access the server. We were suggested to use a Raspberry Pi to manage the camera, and once we established this connection, the next step was to decide on a way to store data. We discussed a possibility of setting up an edge device, which would eliminate the need for cloud services, with specifications of an NVIDIA 4070 GPU and WD-Black 2 TB SSD to allow for fast machine learning predictions. However, halfway through the process, we needed to transition to using a cloud database due to time constraints. The GPU and the SSD we purchased would be used in a server to train the machine learning model, and the model would be deployed and run on the images sent to the Raspberry Pi. This transition to cloud base services introduced MongoDB, which would be the primary database for holding the data transmitted from the Raspberry Pi. To visualize this data, we developed a React web application which we hosted on Heroku, a cloud platform that allows for public hosting. We then secured the connections from each step through a secure socket layer (SSL) tunnel by Cloudflare, and were successful in establishing a connection to the cloud using a Raspberry Pi [6].

1.4 Organization

In Chapter 2, we talk about the background of the proposed solution and mention competitors within the field. In Chapter 3, we walk through the proposed solution's design at a high level and the architecture of individual components. We also mention the requirements and logic that the team had when designing the proposed solution. In Chapter 4, we expand on each of the individual technological components used within the proposed solution. This includes a further explanation on why certain services were selected while keeping true to the Design Rationale. In Chapter 5, we talk about the internal and external testing of the components of the system. In Chapter 6, we talk about the timeline and some risks of the project. In Chapter 7, we mention the challenges encountered throughout the project and highlight incorporated industry standards. In Chapter 8, we talk about the impact of the project in respect to societal topics such as ethics and sustainability. Finally in Chapter 9, we discuss future work and end with some final closing remarks.

Chapter 2

Related Work

In this section, we talk about the client's requests as well as design inspiration for the project. We also mention the current competitors and analyze their costs from a stakeholder's perspective. We examined their strengths and incorporated most of them in the design of the project, such as temperature and wind speed measurements. We identify the target audience of the project and discuss design features aimed at enhancing the user experience. The goal is to ensure that a customer would be more inclined to choose this solution over the current competitors.

2.1 Methods

Early within the project, the client expressed interest in simultaneous photo capture and live stream. We took this as the foundation of the research and design process and became an essential part of the minimal viable product, as we wanted to satisfy their requests. During the research phase, we discovered that the client might have issues with the camera interface as there was over 15 modification options for the camera, which inspired us to make user-friendly camera controls. These controls would only modify the relevant features such as dynamic resolution or brightness and are located on the web application.

2.2 Stakeholder Needs

The goal of the system is to provide more information to a beekeeper about a bee's behavior by supplying images and tracking bees that are entering and exiting a hive. The system prioritizes efficiency as the product incorporates power-efficient sleep cycles, which conserves solar energy. By utilizing affordable materials such as the ESP32 camera module, we are able to reach a larger consumer market, as the product can be used by bee farms of any size, whereas existing solutions cannot. We prioritize reliability and security, as authorized users are able to view the live feed whether they are connected locally or wirelessly. The user interface incorporates Google Single Sign On (SSO) [7], to ensure validity of an email address and assigns a unique user id. This user ID is then verified with MongoDB to see if it exists in the database. If the ID is not found in the database, then the user is granted the guest role and has limited

access to ensure that they are unable to tamper with the system.

To address current competitors, the proposed solution maintains its reliability with a significant reduction in cost. External solar energy beehive monitoring systems such as Eyesonhives [3], can cost upwards of $350 per unit. Internal monitoring systems such as BroodMinder [4], can cost around $42 a unit, but is powered by a lithium-ion battery which will need to be replaced on a yearly basis and can be difficult to access the sensor.

2.3 User Stories

As a new beekeeper, one would want constant supervision to ensure the hive is healthy and stable. Rather than hindering the beekeeper, the system would capture subsequent images and provide real-time updates on the total number of bees entering and exiting the hive with minimal downtime. Now that the beekeeper no longer needs to hover around the hive, they can do other things and can check on their bees remotely.

As a rural beekeeper who may not be tech-savvy, one would want a system that is easy to understand. The beekeeper would like to grasp how to use the product without the need for extensive training. The user would also prefer to view the product and be quickly able to grasp the basic information provided by the product. They would like a system that can operate on its own, without the need for extensive maintenance.

A beekeeper interested in beehive monitoring systems prefers a product that has the option to customize parts of the system. The beekeeper wants to use use different resolutions and frame rates in order to get their ideal live stream. Furthermore, the user wishes to take images captured from the stream and save them locally.

Chapter 3

Design and Rationale

In this section, we examine the project's design at both a high level and at an internal system architecture level. We describe the different methods applied to each component and compile together results within the framework of this project. While designing the project, the team followed their requirements to ensure that the components would function cohesively. This section dives further into the design rationale and analyzes why certain technologies were selected, as well as how they fulfilled their respective tasks.

3.1 Design

The team came up with multiple designs for the project and Figure 3.1 shows the high-level overview of the final design. The rationale is that the solar panel provides charge to the battery, which supplies power to the ESP32 camera. The ESP32 camera is mounted over the beehive entrance and takes continuous images, while simultaneously transmitting over a user's network to a LAMP server(Linux, Apache, MongoDB, and PHP), which is hosted on the Raspberry Pi. The Raspberry Pi will utilize Watchdog scripts to move the images to another folder and then run an externally trained machine learning model on the images as it makes predictions. After processing the last image, these predictions are updated in and out of variables, and the variables and images are submitted to MongoDB. These images and data variables can be pulled on the user interface with a Firebase security layer, allowing unique user identification. Once a user logs in, they are routed to the homepage, where they can see a live stream, the most recently added data, and a data table displaying analytics. Users can make remote modifications to the camera via the live view page by manipulating the resolution drop-down tab or the sliders to change settings such as frame rate and brightness. These updates are then sent over the cloud through the Raspberry Pi to the LAMP server. The camera will then receive these changes, allowing for almost simultaneous updates.

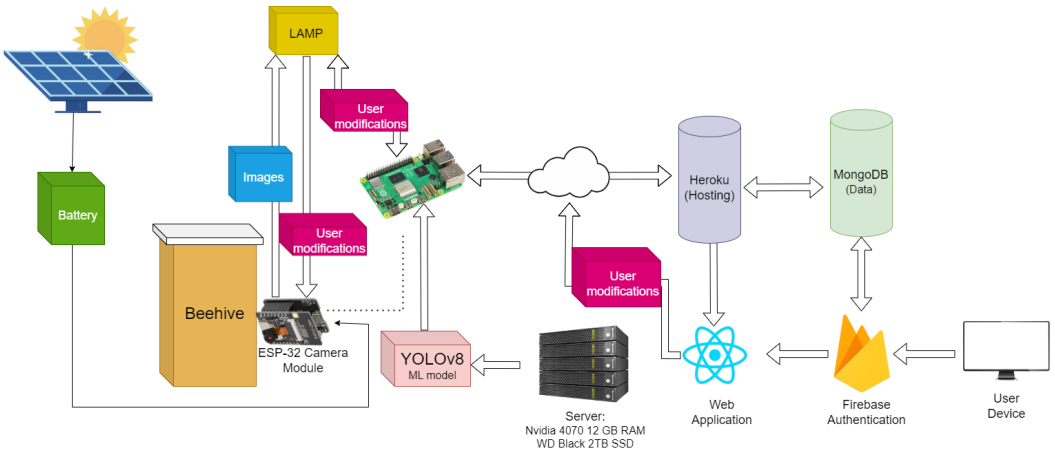


Figure 3.1: High Level Overview

3.1.1 ESP32 Architecture

As shown in Figure 3.2, the ESP32 first connects to the local network using the Arduino WiFi library and gets a list of known IPs from its IP table. If a WiFi network is known, then the ESP32 attempts to establish a connection to the known WiFi network. After connecting to the network, the ESP32 then establishes a connection to the LAMP server over WiFi and this is possible since the ESP32 and Raspberry Pi can verify each other's identity through the use of SSL certificates. Once verification is complete, the ESP32 can transmit images to the LAMP server, and the ESP32 can also receive updates from the Raspberry Pi.

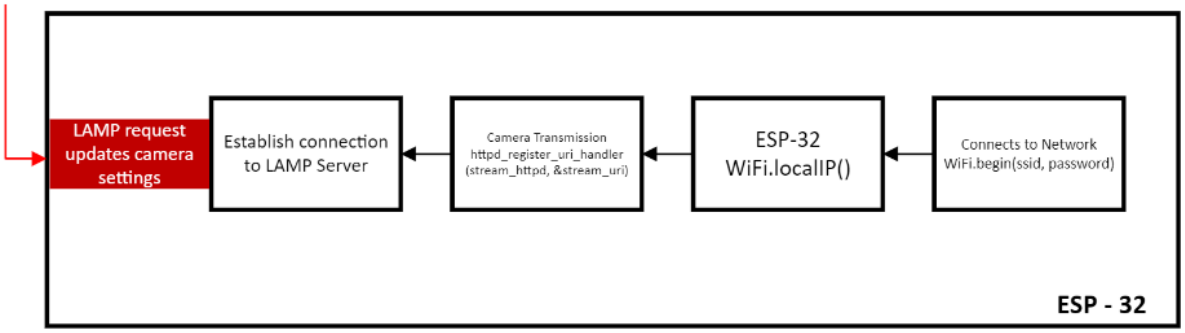


Figure 3.2: ESP32 Software Architecture

3.1.2 Raspberry Pi Architecture

The Raspberry Pi is responsible for most of the functionality of the system and it is a vital piece in the architecture since it allows for remote management, data processing, and uploading to the cloud. As shown in Figure 3.3, the Raspberry Pi receives incoming images from the ESP32 and stores these images in the LAMP server. The LAMP server then moves the received images to another folder where the machine learning model is run using a Python script. The processed photos are then sent to the database through an SSL tunnel, allowing for secure communication

between the Raspberry Pi and the database. The Raspberry Pi also allows for remote access through reverse SSH, which means that the Raspberry Pi only establishes outbound communication with known devices.

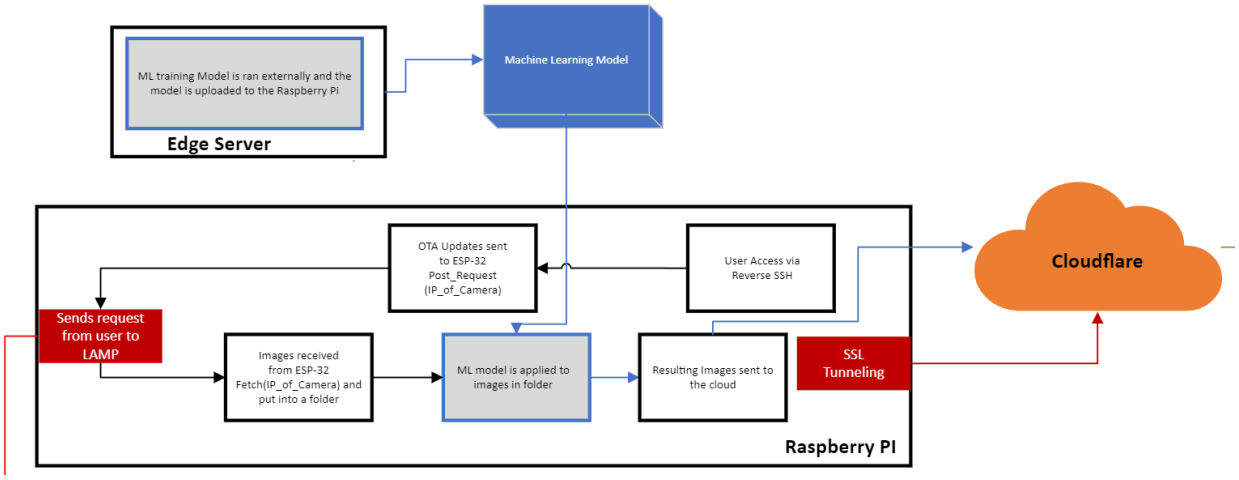


Figure 3.3: Raspberry Pi Software Architecture

3.1.3 Cloud/User Interface Architecture

Another crucial part of the system is the user interface, which displays quantitative and qualitative data in a user-friendly way. As seen in Figure 3.4, the data transmitted through Cloudflare's SSL tunnel will be stored on MongoDB. The data stored in the database will be used for the Heroku-hosted web application. When a user is attempting to access the project's web application, they will go through Firebase for authentication. Depending on whether the user has signed into the website before, the user will either be assigned a new or existing user id. Once a user is authenticated, they will undergo authorization. The web application accesses user roles and permissions from MongoDB through Heroku. If the user is not listed in user roles and permissions, they are automatically assigned to have default privileges. Depending on user privileges, users will have varying access levels to certain web application features.

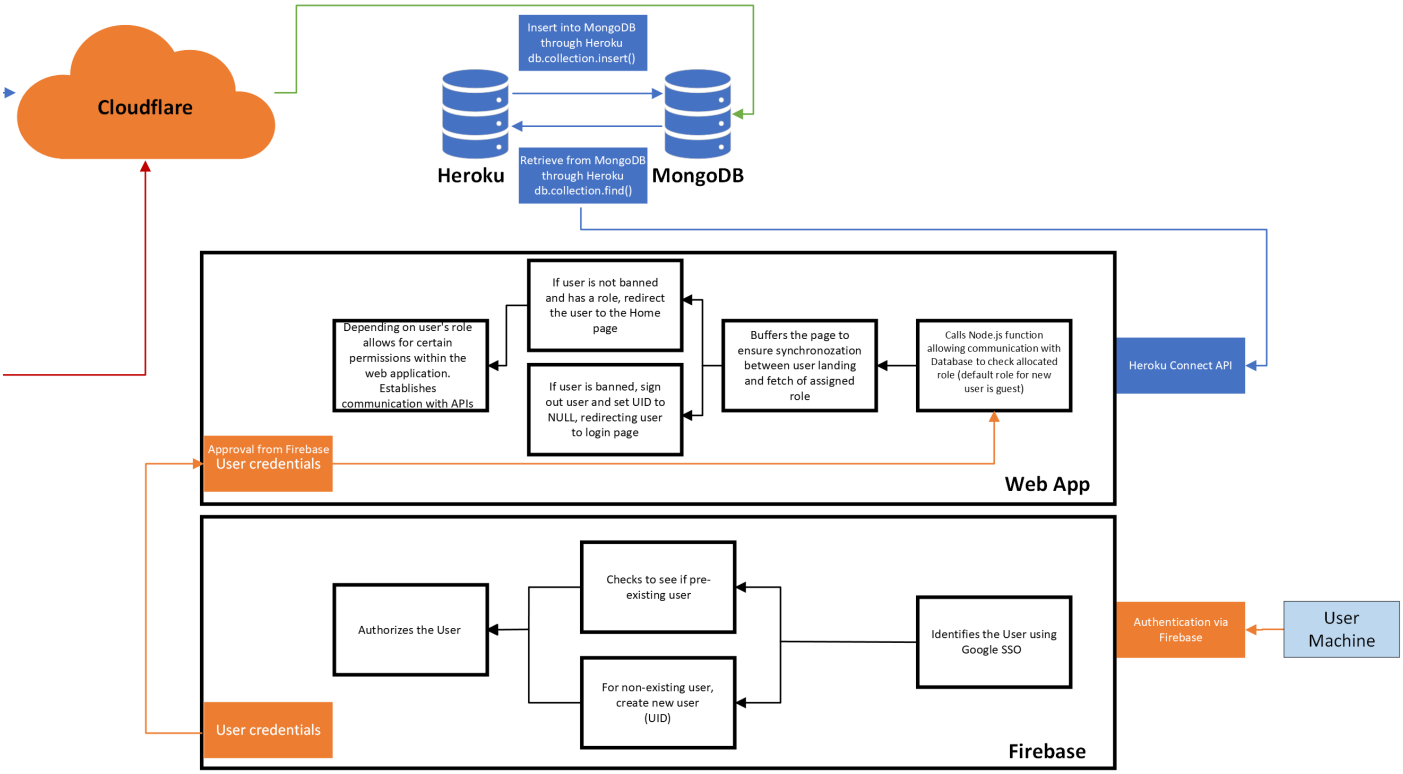


Figure 3.4: Database and User Interface Architecture

3.2 Functional Requirements

The minimum viable product must be able to track the number of worker and pollen bees going in and out of the hive. The system will display the results from the machine learning model and the frames used as input on the web app. Another thing that should be shown on the web app is the live stream showing the entrance of the beehive. The web app should have a simple, yet elegant, UI for the user to easily view the photo gallery, the live stream, and the data correlated with the beehive. The camera system should be implemented so that multiple cameras will be able to connect to a singular access point (AP), allowing the user to manage all the cameras simultaneously.

3.3 Non-functional Requirements

The goal is for the users to be able to navigate the system regardless of their technical background. For this reason, we need a system that is secure, reliable, easy to use, and easy to install. This also means we must incorporate fail safes and a system architecture that protects the system against any breaches or outside attacks.

3.4 Rationale

When originally designing the system, we had numerous ideas and technologies that we considered but ultimately did not implement. Initially, for the frontend, we wanted to set up a JSON server to manage the existing data. We considered using a JSON server because a programmer in React could not modify a JSON file; instead, they would need to use backend technologies such as a JSON server or a Node.JS server to make posts or patches to JSON files. A React JSON server allowed us to eliminate the need of having to create a backend server. Although it was a great tool for testing, two main issues deterred us from continuing. The two main problems were where the data was being stored and how the data was formatted. A React JSON server stores JSON data in a user's local storage. By storing JSON data in local storage, the website would not only hurt the user experience by taking up their own system's storage space but also impairing the overall web application's functionality. Any update to the user roles JSON file on their local storage would not translate to another person's user roles JSON file. Privilege escalation would also be an issue, as any user could modify their local user roles JSON file. Another issue we noticed during testing was how the JSON data was formatted. A benefit of creating a Node.JS server is that we can format the JSON data in whatever way we wish; however, with a React JSON server, the data is read the same way it is displayed in the JSON file. The formatting issue could be resolved using React.JS; however, we realized this would be too tedious with the amount of data and differing formats displayed on the web application. Having a database and a Node.JS server mitigated both these issues, so we decided to steer the project's direction to incorporate both of those technologies.

Chapter 4

Technologies

In this section, we examine the different technologies used as well as their function within the system. We start off by describing the essential frontend technologies used to develop the user interface. We follow up by providing information about the machine learning model and its related algorithms used to classify and track bees. We then examine the technologies present within the ESP32 camera module and the Raspberry Pi, two core components for performing image processing and communicating with the cloud.

4.1 React.JS

React.JS [8] is a frontend library used for developing web applications. We chose to use React.JS since it efficiently updates and renders components. React.JS has many features that we used to meet the project's objective. Component-based architecture is a core feature of React.JS that allows developers to build encapsulated components that manage different states. Each component is a particular piece that makes up the user interface. By splitting the user interface into individual sections, we were able to organize the web application better. React.JS also allows us to call upon these components anywhere within the web application, allowing for components to be continuously reused on other pages. Another feature that is a part of React.JS is Virtual Dom. We needed a tool that would allow us to perform quick updates and for those updates to be immediately reflected on the user interface. The actual Dom(Document Object Model) holds all the elements that create the web pages. If we were to update the actual DOM, it would be very slow and inefficient as we would have to rerender the user interface every time the web application is updated. The Virtual DOM is a lightweight copy of the real DOM that React keeps in memory. A better way to visualize this would be to compare it to a sketch or blueprint of the actual web page. When a developer makes changes to their React components, those changes are first applied to the Virtual DOM. The final determining factor that got the team to choose React.JS was that it uses JSX files. JSX is a syntax extension that allows users to write HTML directly within JavaScript. We found this feature useful because it provides a clear and concise way to create components by maintaining the HTML, CSS, and Javascript within one file. These core features of React.JS are the primary reason

we chose to utilize React for the web application and overall senior design project.

4.2 Node.JS

While implementing the web application using React.JS, an issue arose when we attempted to modify JSON files. We needed to be able to read a JSON file while continually updating and creating new JSON files. In order to solve this issue, we used Node.JS [9]. Node.JS is an open-source runtime environment that uses JavaScript. We chose Node.JS as the backend server for the web application because it can run thousands of concurrent connections with just a single server. Node.JS can achieve this by providing a set of asynchronous I/O primitives in its standard library. To describe the process further, during a typical one-thread-per-connection model, you would accept a connection and then hand the request off to a thread. While that thread handles the request, the thread is unable to perform any other actions, meaning a developer would have to implement multi-threading to handle multiple concurrent connections and requests. In order to resolve this issue, we can implement a set of asynchronous I/O primitives which would allow the CPU to immediately process the next action. Furthermore, Node.JS works seamlessly with React.JS in that both can access NPM libraries, which offer a wide array of new implementations and functionalities. Finally, another major benefit of Node.JS is that it is able to easily connect to the desired database using a private key, allowing access to all the information acquired from the system.

4.3 MongoDB Atlas

The team decided to use MongoDB Atlas [10] for the database. MongoDB Atlas is a cloud-based, fully managed database service provided by MongoDB. One of the main reasons the team chose to use MongoDB Atlas as opposed to other databases was because MongoDB Atlas is a NoSQL database. The main reason that we needed a database was to store all of the JSON files that were initially stored locally. By storing information in flexible JSON documents, MongoDB makes it easier for us to handle and query data. MongoDB is also highly scalable and it has reduced overhead since it utilizes a NoSQL database. Within each database, MongoDB also allows for additional collections. These individual collections can hold different types of data such as a sized-based or time-based collection. A sized-based collection is a collection that holds a certain amount of data, then eliminates old data and refills that spot with a new set of data, following the FIFO(first in first out) principle. Time-based collections will remove data depending on the time it was added and the amount of time that has passed since it was added. These collections were essential in increasing the overall efficiency of the project, as we were able to add photos into a size-based collection ensuring that the dataset would not surpass the predefined limit. We were also able to add the ML results to a time-based collection ensuring that old data would not take up much needed storage. MongoDB provides the Atlas cloud platform, a fully managed cloud database service that simplifies deployment, operation, and scaling of MongoDB databases. The team

can access the database through Atlas and review, update, or remove anything from any location with an internet connection. Finally, MongoDB offers a free service, allowing us to hold a database and not have the customer incur any cost to maintain it.

4.4 Heroku

As mentioned before, we have set up a database to hold the information that will be displayed on the web application. Hosting the web application on Firebase allowed for streamlines integration, as we had already deployed Firebase for Google Single Sign On [7]. An issue that arose from this seamless integration of Firebase Hosting is that it does not support databases like MongoDB. An example would be that the web application would function properly until it came to the previously mentioned user roles. Banned users would not register under the banned role, admin users would not be able to access the admin settings, and guests would have access to all pages as there would be no database from which to pull preexisting user information. To counteract this issue, we utilized an additional platform to host the MongoDB database. Heroku [11] is a PAAS or cloud-base platform that simplifies web development. Heroku offers a wide range of features that satisfy many of the criteria, such as simplicity, scalability, security, and support. Heroku is perfect for making web development simple. A key feature of Heroku is its ease of integration with databases. This simple integration allows Heroku to access and update the database in a timely manner. Another key factor that allows for updates in quick succession is Heroku's Dynos. Heroku Dynos are smart containers that provide a modern runtime environment for existing applications and databases. They isolate and scale web applications by processing one or more of a database's features. Heroku also provides the required security needed for industry standards. This also includes automated patching of the platform and any database associated with it. By using Heroku, we were able to eliminate the issues of being unable to access the database, while improving the performance and security of the web application in the process.

4.5 Roboflow

Using the machine learning algorithm, we tried to get data that allowed us to determine the current health of the hive. This was accomplished by tracking the number of pollen bees going in and out of the hive and the number of bees not carrying pollen going in and out. We needed to utilize computer vision to predict the number of bees going in and out of the beehive. The first step was to create a custom dataset using the Roboflow [12] annotator. This custom dataset contained pictures captured by the ESP32 camera, labeling each image with a ground-truth bounding box for the different kinds of bees. The team labeled the bees as either bees carrying pollen or bees not carrying pollen. Pictures were taken at the entrance of the beehive, replicating where the camera was meant to be placed. The machine learning model was then be trained using the custom dataset. With this, we can get the count of bees carrying pollen and how

many bees were currently in the hive.

4.6 YOLOv8+Bytetrack

Once we made the dataset, we plugged it into the machine-learning model. But what model did we want to use that allowed for fast prediction yet have a high enough accuracy to track the object properly? Because we had these requirements we needed to satisfy, we needed to have a proper ML model structure. Due to these restrictions, YOLOv8 [13] was the best model structure to use, as it allowed us to accurately detect and classify an object with a very fast prediction time. This object could be a bee, and the fast prediction time and accuracy allowed us to track each individual bee's movement. YOLOv8 worked by having the model look at the image once, then drawing prediction bounding boxes and labeling each box as either a box containing a pollen bee or a worker bee. We then used the Bytetrack algorithm to track each bee. The Bytetrack [14] algorithm compared previous and current predictions and associates the differing boxes together to see if the differing predictions come from the same bee. This ensured that each identifying bee would be properly tagged and tracked. Underneath are pictures showcasing how the bees are tracked going in and out of the beehive. The white rectangle indicates the region where the entrance was located. If the bee moved into the rectangle, then the model should track the bee as it's going in; if the bee goes out of the rectangle, then the model should track the bee as going out. In the pictures, when the bee went fully inside the rectangle, the model noticed that a bee was going into the hive and updated the counter.

Figure 4.1: Initial Capture

Figure 4.2: Bee crosses the border

4.6.1 Repulsion Loss

A loss function is used in a model in order to train the model and penalize the model when the detection is incorrect. The most used loss function in many detection models is Intersection Over Union (IOU). This method checks to see how much of an overlap there is between where the bee is and where the model predicts the bee. The IOU equation would give us a loss value, which the training portion will then use to change the model's weights. However, the IOU loss function is not good at detecting objects that are bundled up or close together. When objects are too close together, it might consider the two or more objects to be one singular object. To detect the bees that are closely together, we use repulsion loss. Repulsion loss [15] is a loss function that penalizes the model when the predictions overlap other objects and other predictions. This ensures that when there is a crowd of bees, it can detect all the bees properly. Repulsion loss works by considering three terms: attraction with one's own ground truth, repulsion from other ground truths, and repulsion from other predictions. By having an equation that takes care of all of these terms, we get a loss value that we can use in the model to backpropagate all of the weights.

Figure 4.3: Repulsion loss annotating tight clusters of bees

4.7 Open Secure Socket Layer (SSL)

One of the client's major requests was that his network was protected from outside attacks, and during a meeting he stressed the importance of finding a secure way to accomplish this. The solution we implemented used SSL certificates [16] to verify the user's identity before allowing them access to the network. The way this works is that the two users both have public keys and to confirm each other's identity, they communicate with the certificate authority, which acts as the authenticator and ensures that both users are actually who they claim to be. This secure method of communication also encrypts the data before it is sent, preventing unauthorized tampering. By using SSL certificates through SSL tunneling, a service provided by Cloudflare, we were able to secure the overall system and HTTP requests were promoted to HTTPS requests and now communicated with the internet via a secure port. Not only did this solution provide us with enhanced security protocols, but it also allowed us to do so without costing us additional funds.

4.8 Raspberry Pi

During the planning phase of the project, we talked about different devices that we could use as the remote manager for the overall system. One of the first options was the Raspberry Pi [17] as it was capable of performing multiple tasks such as a acting as a WiFi access point, allowing us remote management capabilities through reverse SSH, providing us the ability to run a LAMP server, and having the processing power to run the machine learning model. The best

part of all was that the Raspberry Pi fit well within the budget, and we felt that it was the best option to handle all of the tasks we assigned to it. The Raspberry Pi was an integral part of the overall system, being used heavily during the testing portion of the project and is still responsible for running the entire system. The other major benefit of the Raspberry Pi is that with the ability to log in remotely from whitelisted devices, developers have the ability to update the ESP32 using OTA (over the air) updates. Since the Raspberry Pi is deployed at the apiary and is on the same network as the ESP32, it can connect to the ESP32 and provide updates through this process, providing an efficient method of performing maintenance without the developers needing to be physically present.

4.9 Watchdog

We ran into an issue when developing the end-to-end model for the project, the Raspberry Pi received photos from the ESP32 camera module, but we had no way of getting those images to the machine learning model and if we ran the machine learning model we had no way to transmit the new data to the database. Most of the technology revolves around the Raspberry Pi and relies on detection. The project must detect when a photo is taken, what to do when it is detected, and when it is altered. In order to accomplish this, the team implemented a public API called Watchdog. The Watchdog API [18] is a tool designed to monitor processes and systems to ensure they operate correctly and efficiently. Watchdog continuously checks the status and performance of applications, servers, or other components and if it detects any issues such as downtime, errors, or performance bottlenecks, it can trigger alerts or take corrective actions to mitigate the problem. This helps to maintain system reliability and minimizes downtime. The API can be integrated into existing systems to provide real-time monitoring and automated issue resolution, enhancing overall operational stability and performance. By monitoring over certain directories and detecting changes, such as files being added or modified, we created a script that automatically sends the contents of a directory to the ML model once a photo is detected. The very last hurdle was to automate the script. To overcome this hurdle, we created a new cronjob. A cronjob is a scheduled task in Unix operating systems that runs at boot. By doing this, we achieve the desired outcome of sending photos to the ML model once they are transferred to the Raspberry Pi.

4.10 ESP32

When we were deciding what device to use to take photos for the project, we used five main criteria to determine whether it was the right choice for the job: security, quality, programmability, environmental impact, and cost. When we examined the ESP32 camera module [19], we found it was highly secure since it had a process known as secure boot. What this process does is when the ESP32 first starts up or receives new code, it checks that it comes from a verifiable source to prevent malware from being run on the device. This means the ESP32 will not run code from developers or devices it does not trust. In addition to this secure process, the ESP32 met the team's second criterion,

quality. The ESP32 is a high-quality device with an extremely powerful camera, which could provide us with the high-quality images that we need to develop and train the machine-learning model. This was an extremely important part of the decision process since, without high-quality images, the machine learning model would not be accurate enough to provide reliable data to the user and would ultimately prevent us from delivering a high-quality system. Luckily, the ESP32 passed this criterion, leading to the next one, which was programmability. The ESP32 was developed by a company known as Espressif Systems[19] and came with many example boards to demonstrate just how programmable the ESP32 really was. With such a vast selection of open-source libraries and its ability to run both low-level and high-level code, the ESP32 met the requirements of being highly programmable. When we examined the ESP32's environmental impact, we found that this largely depended on the tasks we assigned. To reduce the ESP32's carbon footprint, we decided to leverage the ESP32's programmability to implement a sleep cycle that would switch the ESP32 into deep sleep mode [20], reducing its carbon emissions. This meant that the ESP32 passed the team's fourth criterion; the last criterion it needed to pass was cost. After checking different websites, we found that the lowest cost we could purchase the ESP32 for was $13. This meant it fit well within the budget and was the best choice for the project. Overall, the ESP32 was an integral part of the entire system, and its ability to take high-resolution images, connect to WiFi, and have minimal environmental impact, all while being highly programmable and secure, made it the best option to complete the assigned tasks.

4.11 Solar Panel and Rechargeable Battery

When selecting a power source to power the entire system, we wanted a renewable power source that could sustain the system without requiring maintenance. After reviewing different power sources, we ultimately chose a solar panel since it could charge the system in an environmentally friendly way. The team felt that the solar panel was cost-efficient since it would only be a one-time purchase instead of having to plug the system into an outlet, which would be a recurring cost. We also knew this was a sustainable solution and would not require much maintenance after installation, reducing the number of on-site visits necessary.

Chapter 5

System Evaluation

While testing the system, we wanted to cover all edge cases to ensure that the system was fully secure and operating as intended. This included both internal component tests, such as checking if the database was successfully communicating with the user interface, and external tests such as capturing images at different resolutions. During testing, we had to make an important decision on choosing stable resolutions as failing to do so would sever the connection between the ESP32 camera and the Raspberry Pi. These tests were essential in ensuring the end-to-end system was fully functional and secure before deployment.

5.1 Internal Testing

Before deploying the system, the team needed a way to demonstrate to the client that all components were integrated successfully. To accomplish this task, the team designed tests for each component and ran these tests extensively to ensure that the components would perform successfully in the field. These tests allowed the team to catch errors and patch vulnerabilities, which provided the client with a fully functional and secure system.

5.1.1 Security Test

For the security component of the project, we tested the web app by attempting to access it using different levels of permissions. We tested this using different accounts where we assigned one to have proper permissions, one to have guest permissions, and one that had no permissions. We wanted to ensure that users who did not have proper permissions only had a specific level of access to prevent unauthorized tampering with the ESP32. After using different logins and providing different access levels to each one, we found that each user's permissions functioned as we expected. To ensure the overall integrity of the end-to-end system, we used Cloudflare to detect whether there were vulnerabilities present and conducted a scan of the SSL tunnel using the service. This helped us verify the service's security and gave us the confidence that the system would be secure when we deployed.

5.1.2 Integration Test

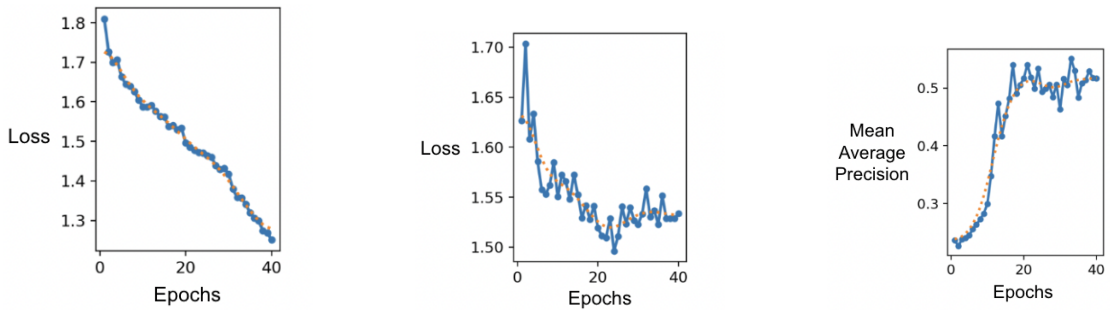
Another important test that we conducted was ensuring the reliability of the system when integrated with other devices. The most important integration we needed to test was the ESP32's capability to communicate with the Raspberry Pi's LAMP server over a wireless network. This test was crucial because upon deployment, the Raspberry Pi would be housed in a separate area, and the ESP32 would need to send data without a wired connection. The team tested this by leaving the ESP32 on and allowing it to send images to the LAMP server over the course of a couple of days. This allowed us to verify whether the ESP32 could communicate wirelessly with the Raspberry Pi for extended periods. After conducting this test, the team was confident that the ESP32 could reliably transmit images to the Raspberry Pi's LAMP server when we deployed it.

5.1.3 Database Test

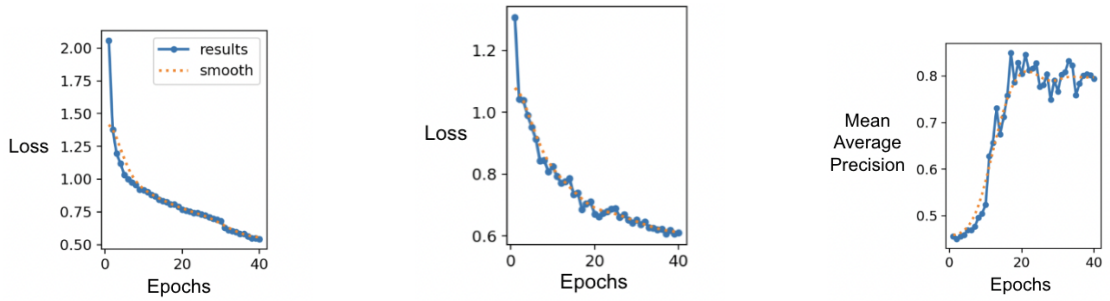
We needed to conduct various trials to test the database for handling data, connectivity, and collections. To test whether the database handled data properly, we tested each way to modify data: POST, PATCH, READ, and DELETE. Initially, the database only held the user roles and permissions collection, so we first tested to see if we could read certain users in the user roles and permissions collection. We ran a script that connected to the database using a MongoDB private key. Once we verified that data was being read properly, we developed similar tests for the other methods. After confirming that all methods of modifying data were functioning correctly, the next step involved testing MongoDB's connection to the website. The first test involved testing whether the web application could access MongoDB locally. To test this, we connected to MongoDB using the private key, assigned a localhost port to access MongoDB, and fetched data. The fetched data would be read and displayed on the frontend. After accomplishing this, we then tested MongoDB's ability to connect to Heroku. We performed the same test but fetched data from Heroku instead of localhost as we previously were doing. The team connected MongoDB by adding the private key to the Heroku configuration. We ran the same script on the web application as in the previous test; however, this time, we fetched data from the given Heroku URL. The final test involved testing different collections. MongoDB provides three additional collections beyond the standard: capped, time series, and clustered index. The team only needed standard, capped, and time series collections for the project. We used the standard collection in the previous tests which has no special features and is meant to hold data. The capped collection holds data up to a certain limit and performs FIFO operations with old and new data. The time series collection holds data for a specified amount of time. To test these collections, we created a capped collection with a very limited size to hold data and a time series collection that would remove data after thirty seconds. The purpose of these trials was to assess the functionality of these collections as quickly as possible. We posted data to these collections and observed whether they behaved as intended. Once we finished these trials, the team was confident enough to conclude that the database was functional.

5.1.4 Machine Learning

To evaluate the system's machine learning section, we split the dataset and looked at three values that dictate how optimal the machine learning model operates: training loss, validation loss, and mean average precision (mAP). The training loss and the validation loss are calculated after every training epoch, which tells us how well the model is learning. We split the dataset into three sections: train, validation, and test. The train data, Figure 5.1a and Figure 5.1d, is used to train the data. The validation data, Figure 5.1b and Figure 5.1e, is used after every epoch to show the loss when the model is inputted with images it doesn't train on. When the values decrease after every epoch, it tells us that the machine learning model is teaching itself properly by minimizing the loss as much as possible. If the values stay stagnant or increase, then that means that the model is not learning properly. The train and validation loss shows how well the model works with images it is familiar with and images not familiar with. Fortunately, when we looked at the graphs of the training and validation, we saw that as the model goes through each epoch, there is a decrease in both loss values. However, even though the model is properly learning through the training data, we still need to see how accurate the boxes are and how accurately they detect the different kinds of bees. To do this, we look at the mAP, which indicates how accurate the bounding boxes are. Two mAP values are important for determining the accuracy of the bounding boxes: mAP50-90 and mAP50. mAP50, shown in Figure 5.1f, uses an IoU threshold of 50%, calculated by averaging the precision scores from the precision-recall curves for each class at this fixed IoU threshold. mAP50-90, shown in Figure 5.1c, considers the averages of mAPs over multiple IoU thresholds ranging from 50% to 95% in increments of 5% and later averages the mAPs to get a single value. This will indicate the precision and accuracy of the bounding boxes. As seen in Figure 5.1, the mAP for validation is increasing as more epochs are being completed. The test data is finally used to see how well the images detect, classify, and track the bees.



(a) Box training loss (b) Box validation loss (c) Metrics of mAP 50-95



(d) Classification training loss (e) Classification validation loss (f) Metrics of mAP 50

Figure 5.1: Resolution Quality Comparison

5.2 External Testing

During deployments, we noticed that there was an inverse correlation between the quality of the images that were captured and the frame rate. The importance of this is that if we capture images at a lower frame rate, we can get good images for classification but a choppy feed, which results in lower tracking accuracy. However, if we increase the frame rate we would need to reduce the image quality and the images would have to be subsequent. This allows for better accuracy with the tracking model, but can result in a worse accuracy for the classification model due to the reduced quality.

5.2.1 Frame Rate vs Quality Capture

In this section, we will discuss the importance of image quality versus frame rates in more detail. As mentioned, we can take images while simultaneously providing a live stream where the images taken are stored on a local device. While testing with a laptop, we noticed a difference in size between the different qualities, as seen in Figure 5.2. We needed a way to quantify the size of images that we were transmitting, as a larger image/packet transmission would take longer was causing trouble for the camera. If the transmission for the images was longer than a second, we would lose the subsequent images as the new images would overwrite the old ones, resulting in choppy frames.

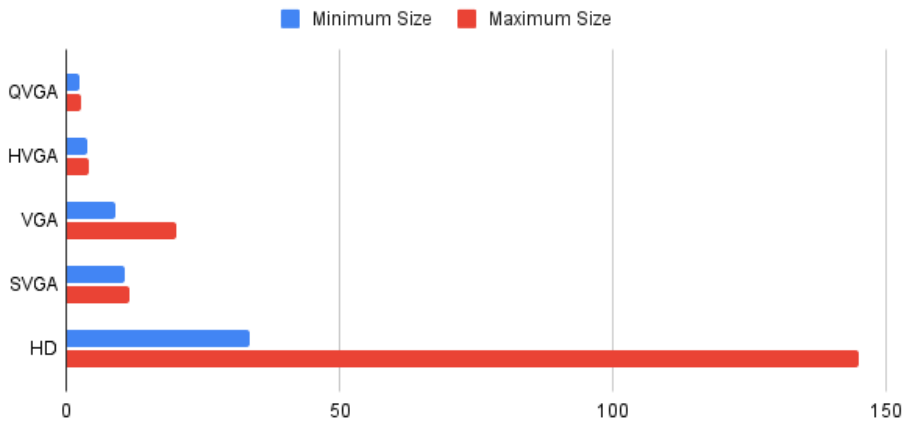


Figure 5.2: Image Quality Ranges in Kilobytes

Within Figure 5.2, we can see that the QVGA has the smallest file size in general, with an observed minimum file size of 2.47 kilobytes(Kb) and a maximum of 2.8 Kb at 19 frames per second (fps). HVGA is the next smallest, with an observed minimum file size of 3.9 Kb and a maximum of 4 Kb captured at 12 fps. VGA has an observed minimum file size of 9 Kb and a maximum of 20 Kb captured at 10 fps. SVGA has an observed minimum file size of 10.7 Kb and maximum of 11.5 Kb captured at 10 fps. HD has an observed minimum file size of 33.6 Kb and maximum of 145 Kb captured at 8 fps. One thing that we had to limit for the test was the maximum frame rate for each resolution, which is displayed in Figure 5.3.

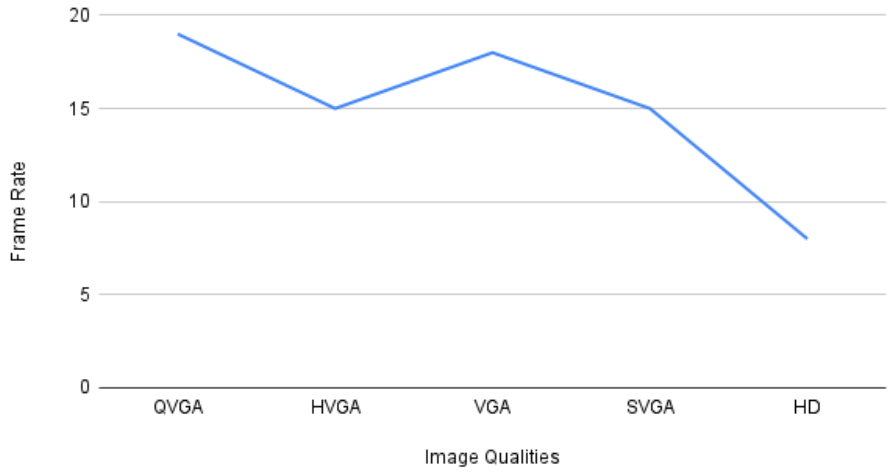


Figure 5.3: Maximum Frame Rate Per Quality

If the camera were to exceed the maximum frame rate, it would freeze, causing the web server to crash, and would need to reboot in order to reconnect to the local network. An example of a QVGA and an HD image can be seen within Figure 5.4.

(a) QVGA Resolution

(b) HD Resolution

Figure 5.4: Resolution Quality Comparison

Chapter 6

Implementation Plan

This section describes the time frame of the project and the team's software development process. Elements of their time frame include research, development, and deployment. A risk diagram was developed by the team to recognize issues of certain components and its impact on the project as a whole. The risk section can be beneficial to those who wish to innovate the team's design.

6.1 Timeline

During the fall quarter, we focused efforts on researching the ESP32 camera and tried to get sample images for the machine learning model. We split ourselves into small teams and focused on integration for the whole end-to-end system, with a team focused on the frontend/user interface, backend, and machine learning. With the frontend, we started drawing up the initial website design on Figma and talked about certain implementations such as data displayed, live stream features, etc. For the backend, we talked about how we could connect the Raspberry Pi to the ESP32 camera, using a GUI or SSH approach. For the machine learning, we talked about different models and how we could improve from last year's, after a couple of weeks of discussion we landed on utilizing YOLOv8. In the winter, we began working on the frontend and backend integration. This was done by initializing a connection between the web application to MongoDB and Google SSO. For the backend, we started implementing the sleep cycle for the ESP32 camera system and researching its deep sleep capabilities. As for the machine learning model, we did more research and started designing a model using repulsion loss. In the spring, we were able to finish final touch ups to all of the connections and deployed the project at the apiary.

6.2 Agile Software Development

At the start of development, we created user stories to capture the features we wanted in the project. We had a primary user story that focused on the basic functionality of the project, along with other user stories that focused on additional features. Breaking down the primary user story, we needed a system that would display a real-time bee count and a

live video feed that could be viewed remotely. Before assigning roles, we identified the basic technologies needed to achieve these goals. We needed a camera that could maintain a stable connection, a Raspberry Pi to process photos from the camera and run them through a machine learning model, a database to hold all necessary data, and a website to display all the required information. We split the team into smaller groups based on their own individual experience in specific areas. One group primarily worked on the ESP32 camera and the Raspberry Pi, another focused on the machine learning module, and the last group concentrated on developing the web application and the database. Once we finished the project's basic functionality, we moved on to the other user stories. These user stories focused on specific components of the system; for example, one user story aimed to make the project more user-friendly. The two main components needed to achieve this were a system that could run by itself and a web application that was user friendly. We continued to use the smaller groups to achieve this, mainly the group that focused on the ESP32 camera and the Raspberry Pi, and the group that focused on the website and database. Anyone who was not part of those teams was assigned to help either team implement these features. The last user story focused on making the project customizable. We needed a project that could change the resolution and frame rate of the camera from the web application. The project also had to be able to take captured photos from the camera and provide the user the option of storing these images on their local storage. We maintained the same groups to achieve these goals.

6.3 Project Risks

Throughout the development process we thought of different scenarios that are prevalent in the project. Since the system incorporates solar-power, it introduced a new set of potential issues that were not present in last year's project. Some of the risks that we thought of are present in Figure 6.1. While field testing the CameraWebServer Arduino code, we ran into an issue where a camera browned out using the FTDI programmer early on within the research phase. This error could have been caused by one of many events such as the wiring to the camera not supplying a constant 5V, the camera having an internal hardware issue, the camera not being in the right boot mode, and losing connection while uploading were some of the main issues we encountered. To remedy this, we tried to use a different form of connection connection, resulting in the usage of the ESP32 camera sister board and a USB-A to Micro-USB cable. This created a stable connection, but ultimately resulted in replacing the camera as the camera that was due to tamper issues from the breadboard connections.

Risk	Likelihood	Impact	Risk Rating	Response
Not enough sunlight	3	4	12	Battery is USB-C chargeable, solar panel can be repositioned
Camera “browns out”	2	5	10	Camera will need to be replaced
Unauthorized user access	1	3	3	Non-authorized users are guests with limited access
Cloudflare goes offline	1	5	5	Wait for reconnection

Likelihood values:
1 - Very unlikely
2 - Not likely
3 - Possible
4 - Probable
5 - Very likely

X

Impact values:
1 - Negligible
2 - Low
3 - Moderate
4 - Significant
5 - Catastrophic

=

Risk Rating
[1-6] - Low
[7-12] - Medium
[13-25] - High

Figure 6.1: Risk Analysis Chart

As seen in this table we have estimated the likelihood of other end-to-end issues that could arise such as unauthorized access to the user interface. As a result we formulated a risk rating to signify how detrimental those risks were to the project. The likelihood value represents the frequency at which a risk could occur, while the impact value represents the effect it has on the system.

Chapter 7

Constraints and Standards

This section analyzes the roadblocks the team faced during the development stages of the project as well as how they overcame these obstacles. The team describes how they needed to adapt to setbacks which required them to come up with creative solutions to continue moving forward with the development process. The team also discusses standards that were incorporated into the system and the importance these standards had on the system's functionality.

7.1 Constraints

Over the course of this project, the team ran into different roadblocks each presenting with their own challenge to overcome. While these roadblocks hindered the development of the project, we were able to overcome these obstacles through hard work and dedication. One of the toughest constraints that we needed to address was time. In the initial phases of the project we noticed that we would plan to accomplish certain goals by a set date but soon learned that we were extremely optimistic. Even if we managed to complete 90 percent of the goals for that week, that remaining 10 percent would be an additional burden that would carry on to the week following. To combat this issue, the team began to overestimate the amount of time needed for each task so that we could set aside enough time to ensure that each task was completed in an efficient and timely manner. This also allowed us to perform additional maintenance and catch errors that would normally be missed by the initial way of completing tasks.

Another issue that we came across was the ESP32's [19] inability to maintain a stable connection with the WiFi network while providing a consistent live feed. The reason for this issue arising was due to the fact that the ESP32 had multiple processes running on it and needed to perform different tasks simultaneously. Some of these tasks include providing a live stream which is able to be viewed through the web app, connecting to a WiFi access point, taking images of the bees and uploading these to the LAMP server [21], and utilizing the Cloudflare SSL tunnel [22] for secure communication with the web app. With all of these tasks being run on the ESP32 simultaneously, the ESP32 needed frequent restarts and this increased the downtime of the system tremendously. To mitigate the total downtime of the system, we focused efforts on optimizing the ESP32 in a way that it could maintain a stable connection with

the WiFi access point while performing all of its necessary tasks. The way that we remedied the problem was by incorporating a sleep function where the ESP32 took a few seconds to reset, and we timed this with one of the scripts that needed to pull images from the LAMP server, and these two processes needed to run independently of one another. By splitting up these two processes in this way, we were able to reduce the downtime of the system significantly and this meant that the analytics were closer to real-time while still being extremely accurate. In the end, the ESP32 was able to provide a live feed while maintaining a WiFi connection and performing its necessary tasks, all with minimal downtime of only 10 seconds to perform a system reboot.

An ethical constraint that we ran into while building the system was the matter of preserving the customer's internet safety by not exposing network ports to the open internet. During meetings with the client, he expressed that he did not want to open any ports on his network as this would expose his internet to attackers. As a result, he prohibited the use of port forwarding which we had been using during initial tests. At first, we experimented with SSL certificates, which would be self-signed and then shared between the devices that needed to communicate with one another. We later discovered that Cloudflare provided this service as well as SSL tunneling, which became part of the proposed solution. By using Cloudflare, we were able to provide internet security to the client while still having the functionality that we desired. Another way that we ensured the client's internet safety was by having the Raspberry Pi, which was left at the apiary, establish a connection with the database. By setting up the Raspberry Pi in this way, we ensured that only outbound communication was possible from the Raspberry Pi which further protected the client's internet privacy.

Another issue that we came across was attempting to use the previous year's system at the beginning of the development process. When we first accessed the code base and hardware, we found that much of the system was no longer functional and there was little documentation for us to go off of. As a result, we needed to rebuild the system using new materials and implement a completely new code base. This presented a major issue since we now needed to rebuild the system and add the new features that the client requested. While the previous team had a dataset consisting of images that could be used to train the machine learning model, we found that the resolution was not high enough, which would affect its accuracy, meaning we would have to collect an entirely new dataset. However, despite these setbacks, we worked to build a more efficient system with a brand new machine learning model along with documentation to assist future teams in setting up and accessing the system. With hard work and careful documentation, we were able to overcome this setback and provide the client with a system that had all of the features requested and more.

7.2 Standards

During the course of the project, we implemented many different systems that each had their own practices and testing methods. An example of a system that we chose to implement is Cloudflare (ISO 27001) [22] since it is a vital part

of securing the end-to-end system, and it is an industry-standard solution. Cloudflare is a standardized service that many companies use for handling SSL tunneling (ISO 27001) [23] and securing their systems. We chose to use it in the project to gain hands-on experience with the service and because it is trusted by many other industry leaders. By using Cloudflare, we were also able to maximize security while mitigating both risk and cost. As outlined in the risk analysis section, the likelihood of Cloudflare going offline is very low as it is a trusted industry standard solution and has a reputation for being extremely reliable. This made it an ideal choice for the system as it ensured reliable and secure communication between the devices and cost the team nothing since the service is free.

Another standard that we included throughout the project is C++ (ISO 14882) [24] which was used to develop the backend. We chose C++ for the backend because it provides the advantages of both high-level and low-level programming, offering flexibility during development. C++ is also an industry-standard language and is a popular choice among many industry leaders due to its versatility. We also decided that C++ was easier to document and paired well with the hardware since the ESP32 uses C/C++ libraries, which allowed us to implement more features on the ESP32.

As mentioned earlier, we also integrated an industry standard verification system Google SSO. By using Google SSO verification (ISO 27001) [25], we were able to increase the security around the user interface while reducing the potential for spam attacks. We used Google SSO to assign users a unique User ID, which is then verified with MongoDB to see if the user had visited before. If the user had visited before they were given their assigned roles, otherwise they were given the guest role. Similar to CloudFlare, this verification method provided the system with an additional layer of security and was completely free to use.

Another important standard that allows for direct communication between the ESP32 and Raspberry Pi is Wi-Fi (IEEE 802.11ac) [26]. Wi-Fi allows devices to communicate over a wireless local area network (WLAN) and this standard is crucial for maintaining remote connections. Without the incorporation of Wi-Fi, the system's devices would not have been able to communicate with one another without having a wired connection making it extremely difficult to deploy. The standard also encrypts communication between devices making it difficult for attackers to decrypt messages using brute-force methods. By incorporating this standard, we were able to ensure the integrity of the system and have devices communicate with one another remotely.

When fulfilling the client's needs, an important protocol that we needed to implement was HTTPS [27] to help ensure the client's internet security, and this would not have been possible without the use of the standard TLS (ISO 20648) [28]. TLS is a secure way of transmitting data between devices over the open internet and it uses cryptography to protect the transmitted data from being read by attackers. Without the standard TLS, the system would not be able to communicate across the open internet using HTTPS, leaving the client's internet vulnerable to outside attacks. By incorporating TLS, we were able to increase the level of security of the system and fulfill the client's needs.

Chapter 8

Societal Issues

When designing a project, it is important to understand its intended or unintended consequences. We evaluate different societal issues such as ethics and manufacturability to respond to some of the issues the team deemed important. The team also delves deeper into the sustainability and environmental impact of the project by including power measurements. In this section, we also include what we took away from developing the end-to-end system.

8.1 Ethical

Reviewing the whole end to end system, the proposed solution follows the IEEE Code of Ethics for Software Engineers [29]. The solution consists of open-source material and the code that is utilized will be accessible on a public GitHub repository [6] under the MIT license. Within the repository, we have included specific instructions on how to set up the camera system module, ensuring that all interested parties, no matter their level of expertise, will be able to use the proposed solution. We ensure security for client by utilizing secure communication and transmission throughout the system. All captured images are relayed to the database where the private camera IP address will be stored. The application incorporates a management system, that records and allows authorized users to make modifications while guests have limited restrictions. The web application will be as responsive as the domain, and so long as the domain is up and running, the application is accessible as well as other services within the domain.

8.2 Economic

The proposed solution strives to be a low-cost beehive monitoring system that utilizes solar energy. Aside from the raw materials, we ensure that the users are only charged for the recurring cost of hosting their web applications. This pay-as-you-go method allows the user to ensure that they are content with the system.

8.3 Manufacturability

The proposed solution utilizes accessible materials and open sourced code, allowing other interested parties to innovate the project in the coming years. We also have documentation [30] discussing the challenges as well as solutions to those subsequent challenges. The documentation also includes a process on how to set up and ensure a stable connection between the internal systems.

8.4 Sustainability

The proposed solution uses the ESP32 camera's deep sleep capabilities to conserve energy, resulting in an energy efficient, sustainable solution. These power cycles coupled with the solar panel allow for efficient power consumption, ensuring that the camera does not overheat or drain too much power from the battery. The solar panel has a 6 Watt power rating and a maximum charging rate of 1.2 Amps and 5 Volts. As for the battery, it's rated for 10Ah and can supply the ESP32 camera module with 5V from 160-260 mA with the LED module off to 310 mA with the LED module on. Utilizing the ESP32 camera's deep sleep capabilities, the camera only draws 10-25 μA during inactive hours allowing the product to be energy efficient and sustainable.

8.5 Environmental Impact

Bees play an important role in the ecosystem, as they make up a significant percentage of the global agriculture. The proposed solution utilizes solar energy to power an efficient external beehive monitor system that is capable of ensuring the health of a hive. The impact of this system allows beekeepers know about the health of the surrounding environment and ensure the health of their hive. As mentioned earlier, if more bees are entering rather than exiting, swarming may be occurring causing the bees to abandon their hives. On the other hand, if there are lots of bees exiting but not returning, an underlying environmental issue may be causing the death of those bees.

8.6 Usability

The user interface was designed with a simplistic nature to ensure six different pages that have different purposes. The landing page is intuitive, with a description of the project and a button that allows a user to sign in with their Google account. When a user logs in they are routed to the landing page which holds some of the other services, such as a live feed of the connected camera and data about recently tracked bees. The "Live View" page has a scroll bar that allows easier modifications of certain settings and a drop-down menu for different resolutions. These modifications are made in real-time and are visible on the live stream when modified. The "Manage Cameras" page prompts the user with two buttons: an add camera button and remove camera button which redirects users to their desired instruction. The "View

Data" page is split in two with the left side being an array of oldest existing images and the right side displaying a pie chart with the total number of bees in and out for the day as well as the most recently added values.

8.7 Lifelong Learning

Throughout this project, by incorporating industry standard solutions into the proposed solution, the team has gained a more complex understanding of various technologies. To construct the user interface, we had to learn the standards of web development. The development process required us to not only understand technologies such as React.JS and Heroku, but also proper security measures for web applications. Learning how to better secure websites from potential intrusions helped members with their own individual projects and tests for cybersecurity. By incorporating MongoDB into the project, we learned how to better manage and maintain a cloud database. Managing a cloud database also inspired us to incorporate similar technologies into other projects. Learning new concepts and practices from a new technology would be a trend for each component of the system. The team's knowledge grew not only from understanding specific components of the project but also from developing system architecture diagrams. Creating a system architecture diagram gave the team a stronger understanding of the functionalities of the end-to-end system, while also enhancing how each individual technology functions within the system.

Chapter 9

Conclusion

Over the course of the project, we successfully designed a new end-to-end system capable of tracking how many bees enter and leave the hive in a given period of time. The system created consists of a backend that processes requests sent by the frontend and allows user control of the ESP32 camera, a Raspberry Pi, with remote management capabilities, that processes the images through the machine learning model and then forwards the processed data to the database, a user interface that is user friendly and requires Google SSO to gain access, and a cloud security platform to protect all of these components from outside attacks. On top of constructing the overall system, we also implemented different metrics that can be viewed on the web app and provides beekeepers with important metrics such as temperature and bee count. These metrics can be used to gauge the health of the hive and can indicate whether there are problems within the hive. We also developed a machine learning model using a custom dataset which was gathered using the ESP32 while at the apiary. This custom dataset was then curated to filter out blurry images or images that did not have the best lighting and then each image was hand-boxed to outline each bee in the image. We then trained the model using the hand-boxed images and the result was an extremely accurate machine-learning model that could relay useful information to the frontend.

The project objective was to create an efficient and secure beehive monitoring system that could accurately track bees entering and leaving the hive and display the information to beekeepers in a user-friendly way. By the end of the project, we had fulfilled the task of creating an efficient and secure beehive monitoring system, and we were not only able to track the bees entering and leaving the hive, but we were also able to provide real-time access to the beehive through a live feed configured on the ESP32. We also fulfilled all of the requests given to us by the client and ensured that the system could not be accessed by outside users without the proper permissions. We also ensured that the system was fully sustainable and environmentally friendly, and this was done by using a rechargeable battery along with a solar panel, providing the system with a reusable power supply. On top of this, we also implemented a sleep cycle that would turn the ESP32 off during the night to save energy and turn it back on at dawn.

After completing this project, the team gained a better understanding of incorporating industry standard practices

into their work and got the opportunity to do so in a real-world setting. We learned how to construct system architecture diagrams and integrate different systems using the diagrams we constructed. We learned how to delegate different tasks amongst the team and pooled resources towards certain systems when necessary. We gained experience in technical writing when logging documentation and made sure to explain the current thought process at each stage of the project.

The beehive monitoring system provides beekeepers with an effective yet inexpensive way to monitor their beehives remotely while having a reduced carbon footprint and a much greater impact on maintaining the health of the beehive. An advantage that the system provides is the benefit of real-time data that can be used by beekeepers to gauge the level of activity in the hive and can tell them whether there are complications or not by analyzing the bee count over time. The system also has the advantage of being fully self-sustainable requiring little to no maintenance as the entire system can be controlled remotely. An area for growth in the system would be to incorporate additional ESP32 devices as this would allow multiple beehives to be monitored simultaneously and would also demonstrate the scalability of the system as a whole.

Future goals for this project would be to see the system functioning with additional ESP32 devices, as stated earlier, as we believe this would be more beneficial to beekeepers who need to monitor the health of multiple hives. Another hope for the project would be to find a more powerful, yet still inexpensive, camera as this would allow for higher resolution images along with the ability to handle more processes with reduced latency. While the ESP32 is a powerful device, we found that it can only run so many processes at a given time and for the system's purposes, we would need a device that is capable of doing so more consistently. Even though we found a way to make the ESP32 run consistently, we became concerned that additional processes that would be added in the future would be too much for the ESP32 to handle. It is for this reason that we believe finding a more powerful camera is imperative to developing the project further. Another future goal that we have for this project is improving the web app as we still believe there is much that can be added to provide a better user experience. While the web app has security protocols in place to prevent unauthorized user access and allow the user to control the ESP32 remotely, we believe that some more styling can be done to provide a better user experience to those using the app. We would also hope to transition to using the edge device to host the server as well as using an x86 machine to store data as this would eliminate recurring costs and would reduce the amount of processes running on the Raspberry Pi. We believe that this would allow for more flexibility as it would open up space on the Raspberry Pi to handle new tasks and would allow it to run more efficiently. As for the recurring costs, by transitioning to a hardware approach, we would no longer need to pay for a server as the edge device could host both the web app and process the images as they pass through the machine learning model. The benefit of this is that the machine learning model would be able to run more efficiently and more accurately providing better metrics about the health of the hive. The other benefit of hardware is it is more efficient since latency no longer becomes an issue. This allows for a faster overall system that is no longer reliant on a subscription model. Overall, the team believes that these changes would help with optimization and would provide a better user experience.

Chapter 10

Acknowledgments

We have worked vigorously on the Beehive Monitoring Project, learning new skills and adopting new practices to be applied elsewhere in the engineering field. However, the team was not able to achieve these accomplishments alone. We want to express the sincerest gratitude to all those who assisted us in these endeavors. Behnam Dezfouli, one of the project's advisors and an Associate Professor in Santa Clara's Department of Computer Science and Engineering, greatly contributed to the project, offering valuable insight that assisted the overall project. Furthermore, Professor Dezfouli was the originator of the idea for the Beehive Monitoring Project. Shiva Jahangiri, another advisor and a Professor in the Department of Computer Science and Engineering provided much-needed support in times of need for the team. Professor Jahangiri always believed in and supported the team. We offer the most sincere gratitude for her taking time out of her day to attend the team's senior design presentation. We also want to thank the family of Mr. and Mrs. Eschelbeck. The Eschelbeck family was kind enough to lend us their home and beehive farm to test the project. Finally, we thank the School of Engineering for providing us with the funds and education required to complete the Beehive Monitoring Project.

Chapter 11

References

1. A. LaSorda, ”In-Hive Sensors Could Help Ailing Bee Colonies,” Scientific American, 31-Aug-2021. [Online].
   Available: https://www.scientificamerican.com/article/in-hive-sensors-could-help-ailing-bee-colonies/.

2. F. Mortimer, ”Beekeeping Basics,” CALS. [Online].
   Available: https://cals.cornell.edu/pollinator-network/beekeeping/beekeeping-basics.

3. Eyesonhives, ”Shop - Eyesonhives,” 17-Jan-2019. [Online].
   Available: https://www.eyesonhives.com/shop/.

4. BroodMinder, ”Hive Sensors,” [Online].
   Available: https://broodminder.com/collections/sensors.

5. wjsanek, “wjsanek/wjsanek,” GitHub, Jan. 30, 2024. [Online].
   Available: https://github.com/wjsanek/wjsanek

6. “SIOTLAB/Hiv3-An-Efficient-Beehive-Monitoring-System,” GitHub, Jun. 06, 2024. [Online].
   Available https://github.com/SIOTLAB/Hiv3-An-Efficient-Beehive-Monitoring-System

7. “Integrating Google Sign-In into your web app,” Google Developers. [Online].
   Available: https://developers.google.com/identity/sign-in/web/sign-in

8. Meta Open Source, ”React,” React.dev, 2024. [Online].
   Available: https://react.dev/.

9. Node.js, ”Index — Node.js V20.2.0 Documentation,” [Online].
   Available: https://nodejs.org/docs/latest/api/.

10. MongoDB, Inc., ”Get Started with Atlas — MongoDB Atlas,” [Online].
    Available: https://www.mongodb.com/docs/atlas/getting-started/.

11. Heroku Dev Center, ”Documentation — Heroku Dev Center,” [Online].
Available: https://devcenter.heroku.com/categories/reference.

12. Roboflow, ”Overview - Roboflow,” 2022. [Online].
Available: https://docs.roboflow.com/.

13. Ultralytics, ”YOLOv8 Documentation,” 18-May-2020. [Online].
Available: https://docs.ultralytics.com/.

14. Ultralytics, ”Byte_tracker,” [Online].
Available: https://docs.ultralytics.com/reference/trackers/byte_tracker/.

15. X. Wang et al., ”Repulsion Loss: Detecting Pedestrians in a Crowd,” ArXiv.org, 26-Mar-2018. [Online].
Available: https://arxiv.org/abs/1711.07752.

16. OpenSSL, ”Open SSL Cryptography and SSL/TLS Toolkit Documentation,” [Online].
Available: https://www.openssl.org/docs/.

17. Raspberry Pi Foundation, ”Raspberry Pi Documentation,” [Online].
Available: https://www.raspberrypi.com/documentation/.

18. Python-Watchdog, ”Watchdog — Watchdog 2.1.5 Documentation,” [Online].
Available: https://python-watchdog.readthedocs.io/en/stable/.

19. Espressif Systems, ”ESP32 Technical Reference Manual,” 2022. [Online]. Available:
https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

20. Voltaic Systems, ”How to Put an ESP32 into Deep Sleep — Voltaic Systems Blog,” Blog.voltaicsystems.com, 30-Jan-2022. [Online].
Available: https://blog.voltaicsystems.com/how-to-put-an-esp32-into-deep-sleep/.

21. “What is LAMP Stack? - LAMP Stack - AWS,” Amazon Web Services, Inc. [Online].
Available: https://aws.amazon.com/what-is/lamp-stack/

22. “How does SSL work with HTTPS?,” ISO 27001 Guide, Dec. 05, 2020. [Online].
Available: https://iso27001guide.com/how-does-ssl-work-with-https-iso27001-guide-iso27001-guide.html

23. “Certifications and Compliance Resources,” www.cloudflare.com. [Online].
Available: https://www.cloudflare.com/trust-hub/compliance-resources/

24. “The Standard : Standard C++,” isocpp.org. [Online].
Available: https://isocpp.org/std/the-standard

25. “ISO/IEC 27001 - Compliance,” Google Cloud. [Online].
Available: https://cloud.google.com/security/compliance/iso-27001

26. “IEEE SA - IEEE 802.11ac-2013,” IEEE Standards Association. [Online].
Available: https://standards.ieee.org/ieee/802.11ac/4473/

27. Cloudflare, “What is HTTPS?,” Cloudflare, 2024. [Online].
Available: https://www.cloudflare.com/learning/ssl/what-is-https/

28. “ISO/IEC 20648:2016,” International Organization for Standardization, Mar. 2016. [Online].
Available: https://www.iso.org/standard/68622.html

29. “Code of Ethics — IEEE Computer Society,” Computer.org, 2017.
Available: https://www.computer.org/education/code-of-ethics

30. A. Kuverji, J. Ursillo, C. Merhab, and A. Sahu, “Senior Design Notes for Future members,” Google Docs. [Online].
Available: https://docs.google.com/document/d/1ObM5-dQ-IdqjVWn9XLgOmZXwsfn8XT0NiZ0EckTgeTs/

31. H. Sahota, ”The History of YOLO Object Detection Models from YOLOv1 to YOLOv8,” Deci, 5-Jun-2023. [Online].
Available: https://deci.ai/blog/history-yolo-object-detection-models-from-yolov1-yolov8/.

32. R. Santos and S. Santos, ”How to Program / Upload Code to ESP32-CAM AI-Thinker (Arduino IDE) — Random Nerd Tutorials,” 4-Feb-2020. [Online].
Available: https://randomnerdtutorials.com/program-upload-code-esp32-cam/.

33. J. Solawetz, ”The Train, Validation, Test Split and Why You Need It,” Roboflow Blog, 4-Sep-2020. [Online].
Available: https://blog.roboflow.com/train-test-split/.

Chapter 12

Appendices

This is a selection of key parts from the source code. The full source code is accessible at:

https://github.com/SIOTLAB/BeehiveMonitoring.

12.1 ESP32 Timer + Deep Sleep

```
void loop() {
  struct tm timeinfo;
  getLocalTime(&timeinfo);

  if (timeinfo.tm_hour >= 8 && timeinfo.tm_hour <= 19) {
    unsigned long currentMillis = millis();
    for (int i = 0; i < 60; ++i) {
      sendPhoto();
      delay(500);
    }
  } else {
    esp_deep_sleep(10000);
    delay(60000);
  }
}
```

12.2 Image Gallery.js

```
import React, { useEffect, useState } from 'react';
import ".././styles/ImageGallery.css"

function ImageGallery() {
    const [images, setImages] = useState([]);
    useEffect(() => {
        fetch('https://hiv3-app-1abe045e0a88.herokuapp.com/images')
            .then(response => {
                if (!response.ok) {
                    throw new Error('Network response was not ok');
                }
                return response.json();
            })
            .then(data => {
```

```
                console.log('Data:', data); // Check what exactly is being returned
                if (Array.isArray(data)) {
                    setImages(data);
                } else {
                    throw new Error('Data is not an array');
                }
            })
            .catch(error => {
                console.error('Error fetching images:', error);
                setImages([]); // Fallback to an empty array
            });
    }, []);

    return (
        <div>
            <h1>Image Gallery</h1>
            <div className = "containerrr">
    {           images?.map((img, index) => (
                    <img key={index} src={img.data} alt={img.image_name} className="galleryphoto"/>
                ))}
            </div>
        </div>
    );
}

export default ImageGallery;
```

12.3 Frontend API Calls

```
      const checkMembersInRole = (role, userEmail)  => {
       return fetch(`https://hiv3-app-1abe045e0a88.herokuapp.com/findmember?role=
       ${encodeURIComponent(role)}&current_user=${encodeURIComponent(userEmail)}`)
            .then(response => response.json())
            .then(data=> {
                console.log(data.message);
                return data.message.includes("Member exists in role");
            })
            .catch(error => {
                console.error('Error', error);
                return false;
            });
       }
```

12.4 Machine Learning BBox Loss Function

```
class BboxLoss(nn.Module):
    """Criterion class for computing training losses during training."""

    def __init__(self, reg_max, use_dfl=False):
        """Initialize the BboxLoss module with regularization maximum and DFL settings."""
        super().__init__()
```

```
        self.reg_max = reg_max
        self.use_dfl = use_dfl
        self.eps = 1e-7
        self.sigma = 0.5

    def forward(self, pred_dist, pred_bboxes, anchor_points, target_bboxes, target_scores,
    target_scores_sum, fg_mask):
        """IoU loss."""
        weight = target_scores.sum(-1)[fg_mask].unsqueeze(-1)
        iou = bbox_iou(pred_bboxes[fg_mask], target_bboxes[fg_mask], xywh=False, CIoU=True)
        loss_iou = ((1.0 - iou) * weight).sum() / target_scores_sum

        # DFL loss
        if self.use_dfl:
            target_ltrb = bbox2dist(anchor_points, target_bboxes, self.reg_max)
            loss_dfl = self._df_loss(pred_dist[fg_mask].view(-1, self.reg_max + 1),
            target_ltrb[fg_mask]) * weight
            loss_dfl = loss_dfl.sum() / target_scores_sum
        else:
            loss_dfl = torch.tensor(0.0).to(pred_dist.device)

        return loss_iou, loss_dfl

    def rep_loss(self, pred_box, ground_truth_boxes):
        # Assume pred_box and ground_truth_boxes are of shape [batch_size, number, xyxy(4)]
        batch_size = pred_box.shape[0]
        RepGT_losses = []
        RepBox_losses = []

        for j in range(batch_size):
            # Get relevant data for the current batch
            pred_boxes = pred_box[j]  # Predicted bounding boxes for this batch
            gt_boxes = ground_truth_boxes[j]  # Ground truth bounding boxes for this batch

            # Filter out invalid annotations (class label -1)
            #pred_boxes = pred_boxes[pred_boxes[:, 4] != -1]
            #gt_boxes = gt_boxes[gt_boxes[:, 4] != -1]

            if gt_boxes.shape[0] == 0:
                RepGT_losses.append(torch.tensor(0.0))  # No ground truth boxes, so no loss
                RepBox_losses.append(torch.tensor(0.0))  # No ground truth boxes, so no loss
                continue

            # Assume ignore_flags are randomly generated for each batch
            ignore_flags = torch.randint(0, 2, (gt_boxes.shape[0],)).bool()

            # Compute RepGT losses
            IoG_to_minimize = self.IoG(gt_boxes, pred_boxes)
            RepGT_loss = self.smooth_ln(IoG_to_minimize, 0.5).mean()
            RepGT_losses.append(RepGT_loss)

            # Sample predicted boxes for RepBox losses
            predict_boxes_sampled = []
```

```
            for gt_box in gt_boxes:
                # Randomly sample a predicted box
                index = random.choice(range(pred_boxes.shape[0]))
                predict_boxes_sampled.append(pred_boxes[index, :4])
            predict_boxes_sampled = torch.stack(predict_boxes_sampled)

            # Compute RepBox losses
            iou_repbox = self.calc_iou(predict_boxes_sampled, predict_boxes_sampled)
            mask = torch.lt(iou_repbox, 1.0).float()
            iou_repbox = iou_repbox * mask
            RepBox_loss = self.smooth_ln(iou_repbox, 0.5).sum()
            / max(torch.sum(torch.gt(iou_repbox, 0)).float(), 1.0)
            RepBox_losses.append(RepBox_loss)

        # Return the mean of RepGT and RepBox losses over all batches
        return torch.mean(torch.stack(RepGT_losses)), torch.mean(torch.stack(RepBox_losses))

    @staticmethod
    def _df_loss(pred_dist, target):
        """
        Return sum of left and right DFL losses.

        Distribution Focal Loss (DFL) proposed in Generalized Focal Loss
        https://ieeexplore.ieee.org/document/9792391
        """
        tl = target.long()  # target left
        tr = tl + 1  # target right
        wl = tr - target  # weight left
        wr = 1 - wl  # weight right
        return (
            F.cross_entropy(pred_dist, tl.view(-1), reduction="none").view(tl.shape) * wl
            + F.cross_entropy(pred_dist, tr.view(-1), reduction="none").view(tl.shape) * wr
        ).mean(-1, keepdim=True)

    def repulsion_loss_torch(self, pbox, gtbox, deta=0.5, pnms=0.1, gtnms=0.1, x1y1x2y2=True):
        repgt_loss = 0.0
        repbox_loss = 0.0

        print(pbox.shape)
        print(gtbox.shape)

        # Compute IoU between predicted and ground truth boxes
        pgiou = rep_bbox_iou(pbox, gtbox)
        ppiou = ret_bbox_iou(pbox, pbox)

        print(pgiou.shape)
        print(ppiou.shape)

        # Initialize repulsion losses
        repgt_loss = torch.zeros(pbox.shape[0]).cuda()
        repbox_loss = torch.zeros(pbox.shape[0]).cuda()

        # Iterate over each example
        for i in range(pbox.shape[0]):
```

```
            IOG = self.IoG(gtbox[i], pgiou[i])
            repgt_loss[i] = torch.mean(self.smooth_ln(IOG, deta))

            mask = ~torch.isnan(ppiou[i])  # Filter out NaN values
            repbox_loss[i] = torch.mean(self.smooth_ln(ppiou[i], deta))

        return repgt_loss.mean(), repbox_loss.mean()

    def calc_iou(a, b):
        area_b = (b[:, 2] - b[:, 0]) * (b[:, 3] - b[:, 1])

        iw = torch.min(torch.unsqueeze(a[:, 2], dim=2), b[:, 2]) - torch.max(torch
        .unsqueeze(a[:, 0], dim=2), b[:, 0])
        ih = torch.min(torch.unsqueeze(a[:, 3], dim=2), b[:, 3]) - torch.max(torch
        .unsqueeze(a[:, 1], dim=2), b[:, 1])

        iw = torch.clamp(iw, min=0)
        ih = torch.clamp(ih, min=0)

        ua = torch.unsqueeze((a[:, 2] - a[:, 0]) * (a[:, 3] - a[:, 1]), dim=2) + area_b - iw * ih

        ua = torch.clamp(ua, min=1e-8)

        intersection = iw * ih

        IoU = intersection / ua

        return IoU

    def IoG(self, gt_box, pre_box):
        inter_xmin = torch.max(gt_box[:, 0], pre_box[:, 0])
        inter_ymin = torch.max(gt_box[:, 1], pre_box[:, 1])
        inter_xmax = torch.min(gt_box[:, 2], pre_box[:, 2])
        inter_ymax = torch.min(gt_box[:, 3], pre_box[:, 3])
        Iw = torch.clamp(inter_xmax - inter_xmin, min=0)
        Ih = torch.clamp(inter_ymax - inter_ymin, min=0)
        I = Iw * Ih
        G = ((gt_box[:, 2] - gt_box[:, 0]) * (gt_box[:, 3] - gt_box[:, 1])).clamp(self.eps)
        return I / G

    def smooth_ln(self, x, deta=0.5):
        return torch.where(
            torch.le(x, deta),
            -torch.log(1 - x),
            ((x - deta) / (1 - deta)) - np.log(1 - deta))
```