University of Central Florida

Department of Electrical and Computer Engineering

Sit & Chow – Smart Training Dog Feeder

Group 25

Authors:

Maria Tran	Andrew Balmes	Andres Arzola	Natalie Nguyen
Computer Engineering	*Electrical Engineering*	*Computer Engineering*	*Electrical Engineering*

Review Committee:

Dr. Tong Wu	ECE	Assistant Professor
Dr. Shahana Ibrahim	ECE	Assistant Professor
Dr. Truong Nghiem	ECE	Associate Professor

Mentor:

Dr. Saleem Sahawneh	ECE	Lecturer

Table of Contents

Chapter 1 – Project Summary

Chapter 2 – Project Description

2.1 Motivation & background

Owning a pet comes with the responsibility of ensuring that pets are healthy, well fed, and properly trained. For many dog owners, balancing consistent feeding schedules and obedience training with the demands of daily life can be difficult. Busy work schedules, travel, and general responsibilities often interfere with maintaining reliable mealtimes or proper training. As a result, there is a growing need for smart pet care technologies that provide both convenience and effective behavior support.

One of the most fundamental commands in dog training is “sit” for basic discipline, safety, and other skills. However, consistency is essential to reinforcing this behavior, and not all owners are able to provide that reinforcement training every mealtime. Having human interaction whilst feeding their pets is extremely important in strengthening connections between pet and owner. Traditional automatic dog feeding systems offer convenience for feeding but do not allow the interaction between pet and owner, which can be a missed connection opportunity. Therefore, our system combines the ease of automatic dog feeding with the ability to communicate to your dog through the system.

Sit & Chow seeks to bridge this gap by combining automated feeding with training reinforcement. The system will have automated feeding at scheduled times that will only dispense after the dog is seen sitting. With this, the feeder provides both consistent food intake and daily behavioral training. Also, the system is connected to a mobile app in order to manage feeding schedules and portions, making it easier and more convenient for owners to balance their life’s schedule and have strong pet care support. In addition, the mobile app also allows the user to view the dog through the built-in camera at any point in the day. Thus, allowing a busy owner the ability to check in with their dog.

This project is motivated by the opportunity to go beyond simple automation and create a device that improves the daily routine of dogs and their owners, promoting healthier habits, and stronger training consistency.

2.2 Current Commercial Technologies and Existing Projects

2.2.1 YumShare Solo Automatic Feeder with Camera

The YumShare Solo Automatic Feeder with a Camera, developed by Petkit, integrates an AI-powered camera to monitor and manage automated feeding times [1] . The system offers scheduled feeding times, precise portion control, and real-time monitoring through a built-in camera, which is connected to a mobile application. The app provides a live video feed with owner-to-pet communication and voice recording whilst setting multiple feeding times with potentially customizable portions and servings.

The AI-powered camera has many advanced features for intelligent pet monitoring. It captures motion and then automatically classifies them into “feeding,” “eating,” and “pet visiting” with timestamps for each recorded event. The camera has a 940nm infrared LED fill light and high-sensitivity sensor, ensuring accurate detection even in low-light conditions.

The technology used in the Petkit YumShare is a reference point for the technologies that will be used for the Sit & Chow, such as scheduled feeding, portion control, and remote monitoring via a mobile app. While the YumShare leverages AI to categorize what pet activity is happening, Sit & Chow focuses specifically on behavior-based detection, using machine learning to determine if the dog is sitting before dispensing food.

2.2.2 Ultralytics YOLOv11

YOLO (You Only Look Once) by Ultralytics is a cutting-edge, real-time object detection framework that uses deep learning to identify objects in images or video streams. YOLOv11 offers fast and accurate detection, supports custom model training for specialized tasks, and provides a user friendly Python interface [2]. Therefore, making it easy to use with detailed documentation. It is pre-trained on thousands of various types of images, enabling robust general-purpose object detection.

A similar computer vision model approach will be developed for Sit & Chow but on a smaller scale. Rather than detect a broad range of objects, the system will specialize in identifying dog postures, specifically, when a dog is sitting. This model is still detecting in real-time while logging events through the camera system. By training a machine learning model on various images of different dog postures, Sit & Chow will be able to recognize the sitting posture to trigger the feeding system.

2.2.3 Academic Design Using YOLOv5

This project utilizes Ultralytics YOLOv5’s deep learning to manage smart nutrition management. It uses a weight sensor to control the portion size for each animal, a camera to identify each pet, and an ultrasonic sensor for proximity detection [3]. For each feeding session the system will recognize the specific animal that approaches the feeding system and the weight of that specific animal. Using this information, it will then dispense the specific amount of food that correlates to the animal's weight, allowing for healthy and proper control.

Sit & Chow will use a similar concept to portion feeding where the owner will be allowed to set the amount of food that will be fed to their dog. The camera will be powered by a machine learning model to identify when the owner’s dog is sitting down to receive its food. Thus, combining the aspects of proper portion sizing and training of good behavioral habits.

2.2.4 DIY Pet Feeder

The DIY Pet Feeder combines the concept of scheduled feeding for cats with the ability to entertain them as well [4]. This product uses a slotted wheel that dispenses roughly three grams of food per rotation of the wheel. During feeding time, this product uses an ultrasonic sensor in order to detect when the cat is relatively around the system, then it will dispense food through the slotted wheel.

However, the portion sizing in the DIY Pet Feeder offers minimal user control. Sit & Chow builds on this concept but adds several innovations. The owner will be able to customize the portion sizes through a mobile app, a machine learning powered camera to detect behavior, and logs feeding outcomes. This makes Sit & Chow not just a feeder, but a training and monitoring system specialized for dogs, integrating obedience training with automated feeding.

2.3 Goals and Objectives

2.3.1 Goals

Basic Goals

- Provide a reliable, automated feeding solution for dogs.
- Reinforce obedience training by requiring a dog to sit before eating.
- Ensure consistent portion control for healthy feeding habits.
- Allow owners to conveniently manage feeding times and portions through a mobile app.
- Allows owners to check in with their dog at any point of the day through the mobile app and camera.

Advance Goals

- Improve pet-owner communication with alerts and notifications.
- Track and analyze feeding compliance.
- Enhance reliability by monitoring food supply levels and system status.
- Owner can speak to pet through mobile app while viewing camera.

Stretch Goals

- Enable selective feeding for multiple dogs.
- Enable simultaneous obedience training and automated feeding for cats

2.3.2 Objectives

Basic Objectives

- Develop a food dispensing system that operates on a timer.
- Incorporate a camera to detect dog’s posture.
- Implement sound alerts to signal feeding time and when food dispenses.
- Design a mobile app for scheduling feeding times, adjusting portion sizes, and logging feeding history.

Advance Objectives

- Add app notifications for low food supply, failed training compliance, blockage, etc.
- Enable secure Wi-Fi connectivity for remote control and data synchronization.
- Allow scheduling multiple meals and portions in 1 day.
- Allow user to speak to dog through speaker.

Stretch Objectives

- Incorporate cloud storage for long-term feeding and training data.
- Enable camera to detect a cat's posture.

2.4 Specifications

2.4.1 System Specifications

Table 2.4.1: Specification of Sit & Chow

Food Storage Capacity	Maximum 5kg
Sound Output Level	65-70 dB
Dispensing Time	Under 30 seconds
Portion Dispensing Accuracy	±5 grams
Backup Battery Runtime	≥ 6 hours
Camera Streaming Resolution	720p – 1080p & depth stream
Camera Streaming FPS	10 – 30 fps
Depth Sensing Range	0.3 – 1 meter
Posture Detection Accuracy	≥ 90% accuracy
Feeding Schedule Accuracy	Within 1 minute
Wi-Fi Connectivity Range	≥ 10 meters indoors
Data Logging Retention	≥ 30 days
User interface	Mobile app, override button, camera display
App Scheduling Responsiveness	≤ 3 seconds delay
Mobile app features	Creating & editing feeding schedule, manual feed, feed history logs, live view camera, notifications, user authentication
Cost	~$150

2.4.2 Key Component Specifications

Table 2.4.2: Hardware Components

MCU (ESP32)	Wifi 2.4 GHz, BLE, 520 KB SRAM
Stepper Motor	~ 35 – 45Ncm Torque
Power Supply	12V, 2A + 5V buck, safe
Speaker	65 – 70 dB at 1m, 2 – 3 sec sound cue
Camera	720p – 1080p, 10 – 30 fps, 0.3 – 1 meter depth
LED Indicators	Red and Green, 2 – 3V
Food Storage Container	Maximum 5kg
Manual Override Button	≥150 gf operating force
Ultrasonic Sensor	≤ 0.5 meters
MicroSD Card	≤ 64 GB
Safety	Motor current limit
Backup Battery	Li-on battery pack 3.7-7.4V, 5000mAh

2.5 System Diagrams

2.5.1 Hardware Diagram

The diagram below illustrates the Sit and Chow's hardware architecture, with the ESP32 at the center connecting sensors, motor, camera, speaker, and mobile app. It highlights data, power, and wireless flows, as well as the regulated power chain from wall plug to components, ensuring reliable operation and user interaction.

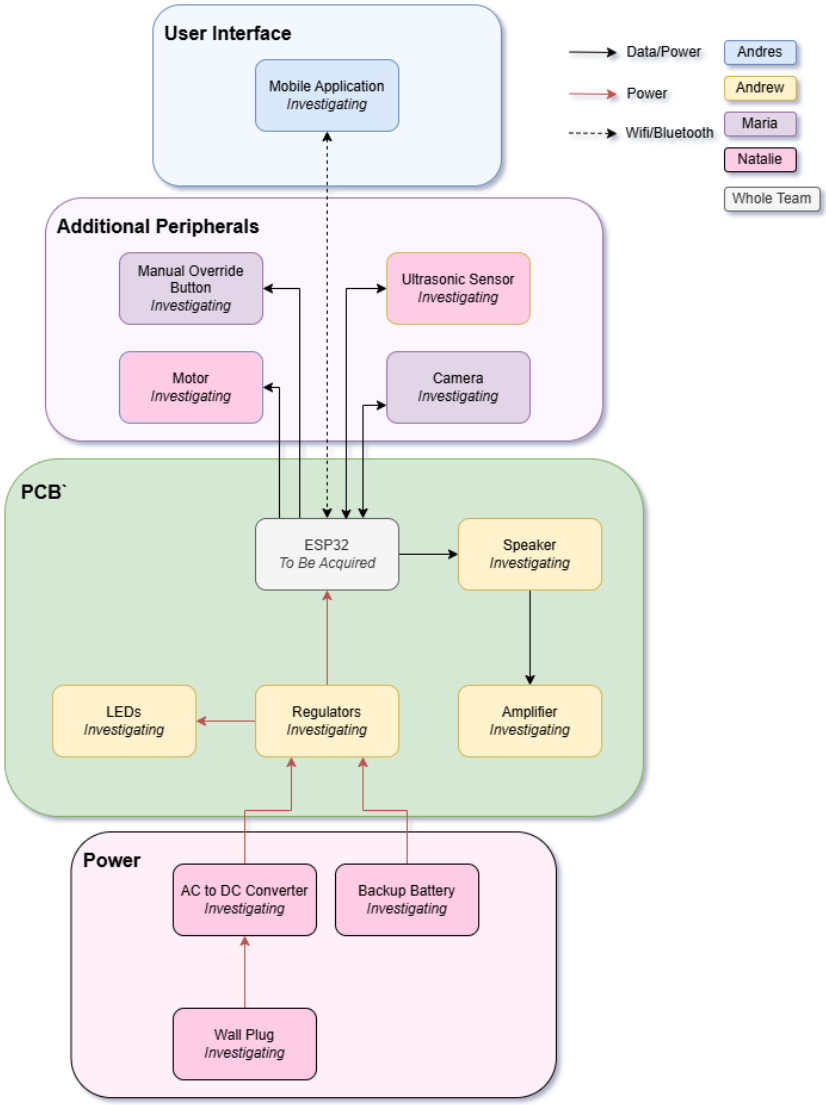


Figure 2.5.1: Hardware Diagram by Authors

2.5.2 Software Diagram

The software block diagram illustrates the Smart Dog Feeder's feeding logic and app integration. It shows how scheduled times trigger sound playback, camera-based sitting verification, and food dispensing via motor control. It also includes manual overrides, food-level detection, logging of feeding outcomes, mobile app communication, and user notifications for low food supply.

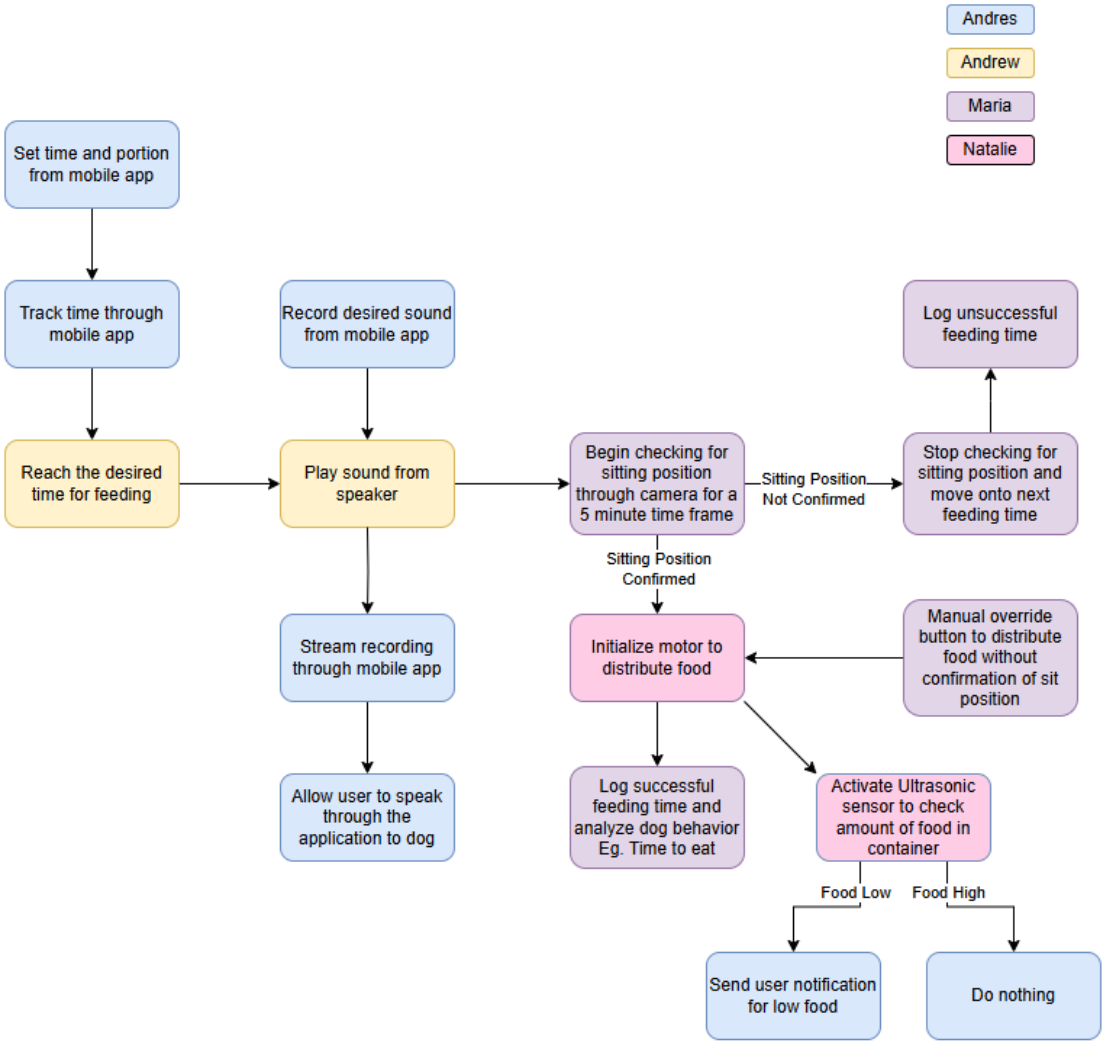


Figure 2.5.2: Software Diagram by Authors

2.6 House of Quality

House of Quality for Sit & Chow, illustrating the relationship between customer needs and technical specifications, and highlighting key design priorities for Sit & Chow.

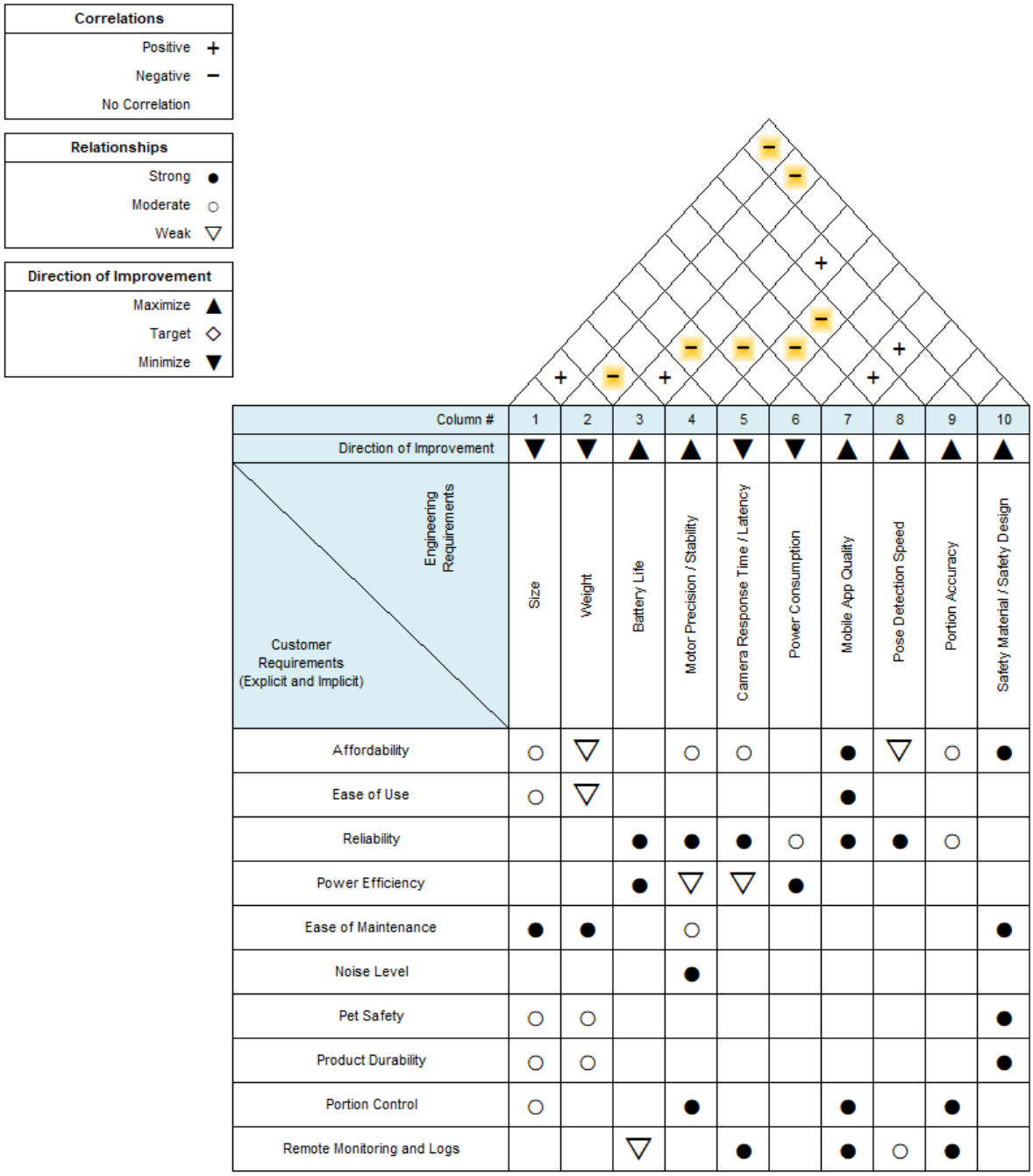


Figure 2.6.1: House of Quality by Authors

2.7 Prototype Illustration

A conceptual diagram of the Sit & Chow smart feeder, showing the integration of key components including the camera module, PCB, motor, and storage area.

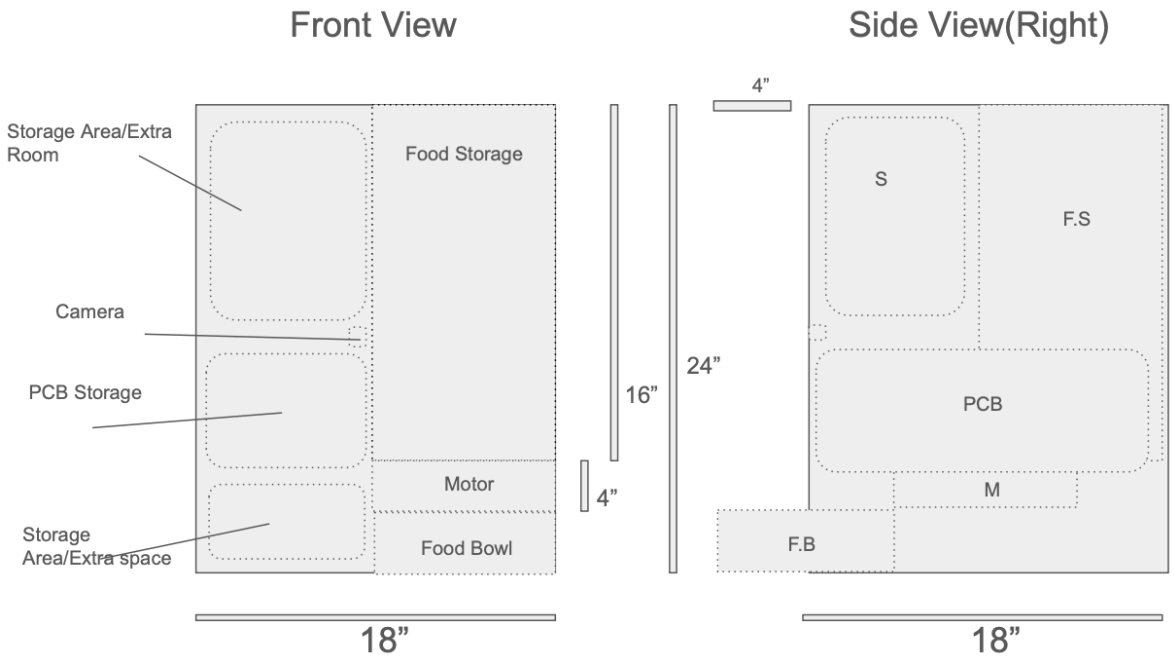


Figure 2.7.1: Illustration

Chapter 3 – Research and Investigation

3.1 Technology Comparison and Selection

3.1.1 FPGA vs MCU

We begin with the heart of this project, the computing system. There are several different kinds of computing systems we could choose, but we will be mostly considering two options, either an FPGA or an MCU. There are many benefits and downsides to using either one, as we will discuss. This choice is heavily influential in the process of constructing this project and determining the overall capabilities of the project. If we make the wrong choice, we could cripple our project by either making things too complicated and making the timeline to complete the project too long or by making things too simple such that there is not enough capability in the computing system to produce the desired outcome.

FPGA

We will first discuss using an FPGA. An FPGA, also known as a field programmable gate array, is a reconfigurable semiconductor device with a matrix of configurable logic blocks connected by programmable interconnects. FPGAs can perform many tasks in parallel, making them ideal for high-speed processes or real-time tasks. They have deterministic timing which ensures precise control with no OS-related delays in a repeatable manner. They have fully reconfigurable hardware that lets the developer tailor the system to handle specific tasks much more efficiently. Multiple independent modules can run concurrently without affecting the performance of one another, which is great for systems with many peripherals that must run simultaneously. They also support custom communication protocols or hardware optimizations. An FPGA is very strong in cases of advanced research or high-performance prototyping due to its extremely high power.

However, there are some downsides that come with using an FPGA. First of all, the biggest downside is the high design complexity that comes with programming and FPGA. There is a long development time due to synthesis and debugging cycles, and the learning curve to design an FPGA is quite steep. For cases where there is not a need for extremely high power, an FPGA could be overkill and bring in unnecessary complexity. Our project is a dog feeder; the most complex part of the dog feeder will be the machine learning algorithm which can be handled without an FPGA. This is the main reason why we will not be using an FPGA as compared to an MCU. However, that is not to say that there are no more reasons why we wouldn’t use an FPGA.

To add to the high design complexity, our project requires some sort of wireless communication such as Wi-Fi or Bluetooth. Neither one is built-in to the FPGA, which means that we would need to manually add these modules ourselves. Additionally, we would need to add communication protocols such as UART, I2C, or SPI to our design. It would also be difficult to update or modify the programming once deployed without having to fully reprogram the FPGA. An FPGA also draws a lot of power as compared to an MCU, making it less than ideal for an IoT project. The development boards for FPGAs are also quite expensive, ranging from around $50-$200+. We would have to be heavily invested in using an FPGA to buy it due to the steep pricing. Furthermore, FPGAs often require external microcontrollers or memory, increasing the total cost and design complexity of the project.

In conclusion, FPGAs are very powerful and capable of handling extremely complex tasks. However, if there is no need for such high performance, an FPGA is not ideal due to the high design complexity and costs. Since this project does not have such high-performance requirements, using an FPGA is not ideal as compared to using an MCU.

MCU

The second consideration for the computing system is an MCU. An MCU, also known as a microcontroller unit, is a small computer on a single integrated circuit. They are designed for embedded and IoT applications, making it a great consideration for our project. They are efficient for sequential control tasks like sensor reading, motor control, and schedule. They also bring real-time responses which are suitable for feeder timings and user interactions. MCUs are typically simple to program and usually have large communities with many libraries and examples for sensors, Wi-Fi, Bluetooth, and mobile app integrations. Some MCUs also have built-in Wi-Fi and Bluetooth modules, which is a major plus considering we want to have communication between the computing system and the mobile application. There are also native interfaces which bring communication protocols such as UART, I2C, SPI, etc. This makes it easy to integrate peripherals like cameras, motors, and sensors. Furthermore, there is excellent support for cloud services which will be necessary for our project. MCUs typically have low power draw and support sleep modes and efficient wake-up triggers. They are also very affordable, usually in the $5-$15 range and are widely available.

There are some downsides to consider. First of all, there is limited parallel processing as compared to an FPGA. MCUs do not have the performance capabilities FPGAs have. The fixed CPU architecture also limits hardware-level customization should there be a need. There may also be a need for careful timing management for concurrent tasks such that everything works in tandem properly. Lower end MCUs may also lack sufficient RAM/flash for more complex tasks such as machine learning algorithms or image processing. There is also limited scalability for high-end computing tasks. Overall, an MCU is unsuitable for applications which require fully customizable hardware timing and logic.

In conclusion, there are many benefits and downsides to both MCUs and FPGAs, but we believe that an MCU has a better alignment with our project's goals. An MCU might not bring the performance of an FPGA, but our system will not have such complex tasks that we need the performance of an FPGA. This, along with the increased simplicity and low cost of an MCU, makes it an ideal candidate for our computing system.

https://www.ibm.com/think/topics/fpga-vs-microcontroller

https://www.arrow.com/en/research-and-events/articles/fpga-vs-cpu-vs-gpu-vs-microcontroller

https://www.wevolver.com/article/fpga-vs-microcontroller

Table 3.1.1:FPGA vs Microcontroller: Aspect-Level Comparison

Aspect	MCU (e.g., ESP32-S3)	FPGA
Performance	Excellent for control and timing tasks. Suitable for moderate compute and edge operations.	Exceptional for high-speed, deterministic, and parallel data processing.
Development Complexity	Simple: uses C/C++, Arduino, or ESP-IDF. Quick to prototype and debug.	Complex: requires HDL (Verilog/VHDL), synthesis, and timing analysis. Steep learning curve.
Flexibility	Fixed CPU architecture with predefined peripherals.	Fully reconfigurable hardware—can design custom processors or pipelines.
Parallelism	Limited to software multitasking or RTOS scheduling.	True hardware-level parallelism; multiple processes can run simultaneously.
Wireless Connectivity	Built-in Wi-Fi and Bluetooth (ESP32-S3).	No native wireless; requires external transceiver and control logic.
Peripheral Integration	Ready-made interfaces: UART, SPI, I²C, PWM, ADC, GPIO, camera support.	Must manually design or instantiate IP cores for each interface.
Power Consumption	Very low power, supports deep sleep and battery operation.	High static and dynamic power usage; inefficient for 24/7 IoT operation.
Cost	Low cost ($5–$15 typical).	High cost ($50–$200+ for dev boards).
Debugging & Testing	Easy via serial monitor, JTAG, or logging.	Requires waveform simulation and timing verification.

Scalability & Reusability	Firmware easily updated and reused.	Hardware reconfiguration possible but requires re-synthesis and new bitstream.
Reliability	Stable, mature ecosystem; well-tested under IoT conditions.	Very reliable hardware once programmed, but prone to configuration errors during design.

3.1.2 MCU Considerations

The primary driver for this project will be the MCU. It is what will drive all the operations of the automatic dog feeder and is what will be communicating with the mobile app as well. As such, it is imperative that we choose the most optimal MCU for this project. We will be considering multiple aspects to determine what is most optimal. These will be power consumption, cost, processing power and speed, obtainability, complexity to program, and communication protocols.

For this project, we will ideally be capable of video streaming to a mobile app for extended periods of time. This means that we should not have an excessive amount of power consumed by the chip. We also would like to make this product cheap and accessible, and the price of the chip can play a large part in this aspect. Therefore, choosing a chip that is relatively cheap while being able to conduct all necessary jobs is important. The cost also plays a big part in the chips' processing power and speed; a cheaper chip will likely be less powerful than a more expensive chip, so we must weigh the pros and cons of each chip respectively.

We also do not want the chip to be hard to obtain. For example, there may be a chip that performs extremely well for a specific purpose and is relatively cheap but is not commercially available. This is undesirable as we would like the chips to be easy to obtain as proof of concept that this product is capable of being produced at a larger scale without difficulty in obtaining the materials. How complex it is to program the chip is also important as it will influence the difficulty in completing the project in general and will also influence how long it takes to complete it.

One last criterion we must consider are the communication protocols the chip is capable of sending/receiving. Our goal is to have a mobile app that is capable of communicating to the chip such that the user can control the dog feeder through the app. This means we must have a chip that is capable of wirelessly communicating to the mobile app. We will discuss in later topics the multiple kinds of protocols that can support this and the protocol we end up choosing. We will now go into detail about the different types of chips we considered and why we ended up choosing the one we did. We will also discuss the specific chip we will order and how we arrived at that decision.

STM32

We will start our discussion with the STM32. The STM32 is one of the chips under consideration due to its popularity in industry and academia. This popularity matters for our project because it shows the chip's proven reliability and availability across many applications, from low-power IoT to high-performance industrial control. It is one of the most used chips in many categories such as IoT devices, consumer electronics, automotive vehicles, medical devices, and in academia. It has many great features that make it a widely used chip.

For one, it has a very rich peripheral set. These chips commonly have support for all kinds of communication such as SPI, I2C, and UART. Many families also include CAN, USB, and Ethernet support, which makes them appealing in industrial and automotive contexts. They can also have a camera interface through DCMI, I2S for audio, SDMMC for SD chips, and advanced motor-control timers [5, 6]. On top of these peripherals, the STM32 runs in a real time operating system, allowing for complete and reliable control over its peripherals.

These chips are also incredibly stable in that they have long life cycles, high reliability, and have a large temperature range. They are so stable that they are used in many industries where stability is critical such as in medical devices and automotive vehicles. Their long-term availability from ST also means they are widely obtainable through major distributors, which helps ensure sourcing is not an issue. These chips support a wide range of performance scaling from some ultra-low power chips to very high-performance dual-core chips. For example, STM32L series devices can run in the microamp range for IoT sensors, while STM32H7 series parts include Cortex-M7 cores running up to 480 MHz for demanding applications [6]. They also have very accurate analog sensing for ADCs and DACs should we want to implement this. A chip like this is a very solid choice for an MCU as it is reliable, powerful, and stable in many cases.

Where this chip goes wrong is in its lack of support for Wi-Fi and Bluetooth. This chip does not come with a built in Wi-Fi/Bluetooth module, which means we would have to have a separate module that does have support. This not only increases cost but also power consumption, since the external module adds its own current draw. Therefore, the complexity to implement the chip would go up as we would have to not only support communication between the Wi-Fi/Bluetooth module to the mobile app, but we would also have to support communication between the separate Wi-Fi/Bluetooth module to the MCU itself. On top of increasing the complexity of implementation and programming, the cost would also increase as we would have to buy a separate module with the MCU itself.

Another downside for this chip is that it has little support for video streaming. The STM32 can capture from a camera interface via DCMI, but real-time Wi-Fi streaming is not realistic. The bandwidth, JPEG encoding, and networking stack are bottlenecks when it comes to streaming. To support this, we would likely have to have a separate chip that handles the streaming, once again increasing both the complexity and price of the project as a whole.

Additionally, the software development cycle is relatively complicated and difficult to learn. While ST provides CubeIDE and CubeMX to generate code, the HAL and LL libraries require a steeper learning curve compared to platforms like the ESP32, and there are fewer open-source hobby examples to draw from. There are also little open-source examples of the STM32, reducing the opportunity to learn from others. Considering all of the previous downsides of the STM32, we decided to move our search elsewhere, hoping for more compatibility in other chips.

In summary, while the STM32 remains highly popular, obtainable, and reliable for many embedded applications, its lack of built-in wireless, added cost for external modules, higher power draw in performance modes, and complexity in programming make it less practical for our feeder project compared to alternatives that integrate wireless and streaming more naturally.

https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

https://www.st.com/resource/en/product_presentation/microcontrollers-stm32-family-overview.pdf

Arduino

The next MCU we will discuss will be the Arduino family of MCUs. This is a similar case to the STM32 in that this family of chips is very popular and is used in a variety of applications. Some of these cases include academia, hobbyist projects, simple embedded systems, and some IoT applications. Due to its popularity, we would like to consider this family of chips in our project. Its popularity stems from its accessibility, affordability, and wide community adoption, rather than raw technical performance.

Let us now consider why we might use it. First of all, the Arduino family of chips is extremely easy to program. It provides its own Arduino IDE using C++ as the programming language and has many libraries and tutorials for beginners to learn from. The programming environment is designed for ease-of-use rather than complexity, which reduces the development barrier significantly. There is also a large community that uses the Arduino family and provides open-source software. This makes development easier and faster due to the countless code examples, forums, shields (a plug-in expansion board), and libraries. There is also a rich ecosystem of motor shields and sensor shields that makes these chips easy to plug and play for this project's hardware. The peripheral support of these chips makes them excellent for controlling motors, LEDs, buttons, and basic sensors.

Additionally, these chips are typically low cost and are available almost anywhere. Some forms of these chips such as the Arduino Uno and Nano can be less than $10, making them super affordable [7]. They can be sold worldwide from places such as Amazon or the Arduino company themselves, making them very easy to obtain. This high obtainability makes them a default teaching and hobbyist choice across the globe. Some of the newer boards can be ARM based. For example, the newer Arduino Portenta H7 and Nano 33 series use modern ARM Cortex-M MCUs with Wi-Fi, BLE, and more RAM. However, these higher-end boards come at a significantly higher cost compared to the classic Uno/Nano.

Where these chips fall off is in their processing power. These chips do not have the capacity to support computationally demanding tasks activities. Compared to other chips that will be considered, the Arduino family does not come close to the processing power the other chips have. For example, the ATmega328P on the Arduino Uno runs at only 16 MHz with 2 kB of RAM, far below the hundreds of MHz and megabytes of RAM available on STM32 or ESP32 chips [7, 8]. Due to the limited processing power of these chips, they are poor for demanding tasks. The limited speed and memory of these chips mean the tasks we need to support like video streaming, ML inferencing, and handling multiple peripherals simultaneously are not feasible on most Arduinos.

There is also no built in Wi-Fi/Bluetooth on most Arduinos giving us two options, adding a separate module to support this communication or moving to a newer Arduino board. Either option requires increasing the cost and complexity of the project. In terms of communication protocols, while Arduinos support the basics (UART, SPI, I²C), they lack advanced built-in options such as CAN, Ethernet, or high-speed USB found on STM32 and ESP32 devices [9]. Furthermore, older boards have inefficient power consumption and typically do not have extended support ultra-low power processing [9]. The newer ARM-based Arduino boards improve this somewhat, but they still lag behind STM32L and ESP32 in optimized low-power modes.

These chips also have a limited peripheral set compared to other chips. They have no camera interface, weaker ADC/DAC performance, and limited timers [9, 10]. Arduinos are not industrial grade either, they are typically chosen for hobby projects and are rarely used in an industrial capacity. They are best suited for educational, prototype, or small-scale DIY applications where reliability and scalability are less critical. All of these limitations place the Arduino family out of consideration as a viable MCU for our project. The most critical drawback is the lack of processing power, as we require a sufficiently capable chip to handle multiple demanding tasks simultaneously [7, 8]. In addition, the absence of built-in Wi-Fi or Bluetooth support makes Arduino an undesirable choice, since reliable communication with the mobile application is essential and other MCUs can provide this functionality natively. Finally, the limited peripheral set—most notably the lack of a camera interface—further reduces its suitability [10]. For these reasons, we decided not to select the Arduino family of MCUs for our design.

All of these limitations place the Arduino family out of consideration as a viable MCU for our project. The most critical drawback is the lack of processing power, as we require a sufficiently capable chip to handle multiple demanding tasks simultaneously [7, 8]. In addition, the absence of built-in Wi-Fi or Bluetooth support makes Arduino an undesirable choice, since reliable communication with the mobile application is essential and other MCUs can provide this functionality natively. Finally, the limited peripheral set—most notably the lack of a camera interface—further reduces its suitability [10]. For these reasons, we decided not to select the Arduino family of MCUs for our design.

https://www.arduino.cc/en/hardware

https://www.elprocus.com/different-types-of-arduino-boards/

Raspberry Pi

Now we will consider using the Raspberry Pi board. This board is very popular among developers due to its high versatility and high performance. Unlike the other MCUs, the raspberry pi is a full single-board computer rather than a microcontroller, changing how it fits within the project. Some use cases of this board are in education, hobby projects, IoT Gateways, Robotics, Computer Vision, Machine Learning, Networking, and other applications. We will now go over the specific advantages and disadvantages of this board.

One of the biggest advantages of using this board is its extremely high processing power. These boards typically run with 64-bit multi-core CPUs that run between 1-2 GHz, and also have 1-8 GB of RAM, far exceeding the performance of any MCU [11]. They are capable of holding full operating systems, handling multiple tasks at once, and handling complex computation. This kind of board is ideal for computer vision, machine learning inference, and image streaming. This already gives it excellent compatibility with this project. Each raspberry pi holds a full Linux OS which is capable of running Python, C/C++, ROS, OpenCV, TensorFlow Lite, and standard networking libraries on boot up. There is also further support for multiple languages and frameworks that aren't immediately included in the raspberry pi's library. This includes Python, Node.js, C/C++, Go, ROS 2, MQTT, and others which make this board useful for cloud or app integration. Most modern models have built in Wi-Fi and Bluetooth modules, not needing any additional hardware. They also have strong multimedia support, meaning they have native CSI/DSI camera/display ports, HDMI, and a GPU for hardware-accelerated video encoding/decoding [11]. This makes it perfect for high quality video streaming and machine learning algorithms, both of which are essential aspects of our project.

Additionally, there is a large community that uses the raspberry pi and has open-source projects to use as additional resources. There are many tutorials, libraries, and OS images for robotics, IoT, and AI. This makes adoption of the raspberry pi as a new board much easier as the learning curve is heavily reduced due to these resources. Furthermore, the chips on the raspberry pi have 40 GPIO pins and support SPI, I2C, and UART communication while also having libraries for them and PWM and GPIO [11]. This makes hardware integration fairly straightforward. The raspberry pi is also available worldwide and has additional, easy to use modular accessories such as official camera modules, HATs (an add-on board), and sensor kits.

However, this board has several disadvantages that make this board less suitable as the sole controller. First of all, there is not direct access to the board's processor. What I mean by this is that the board typically comes in with a full PCB and many peripherals that come with it. What we would have liked to do is simply use the full power of the processor and the Wi-Fi/Bluetooth module in our project. This would mean that we would have a chip powerful enough to support a machine learning algorithm, a mobile app, and video streaming at the same time while handling several other peripherals. However, after extensive research it does not seem as if there is any way to obtain the chip without also obtaining the board. Compute module versions of these boards exist but still come as full boards rather than discrete chips. There does not seem to be a way to separate the two and we would have to simply use the full board if we wanted to use the chip. This approach would simplify the design but reduce the engineering value of the project. This would remove a significant amount of PCB design, and we would also not have a dedicated MCU to handle the peripherals. This undermines the project requirements, and the total effort needed to complete the project. This is the biggest problem with using the raspberry pi by itself.

However, this does not imply that there are not any problems with the performance capabilities of the raspberry pi. This board also has very high power consumption. It typically requires 5-8 W during active use and needs a stable 5V/3A power supply and good thermal design. In comparison, MCUs like the ESP32 only require a few hundred milliwatts. This kind of board is not suitable for ultra-low-power always-on designs due to the possibility of overheating on this board and the lack of a low-power mode on this board. This board is also not a true MCU, which means that it lacks precise deterministic timing. The latency and jitter of the GPIO pins make fine motor control or sensor timing tasks less reliable without an additional MCU. If there was a need for precise I/O control, it would require an external MCU because of the board's unreliability for precise control.

This board also has a very long boot time and OS overhead. Unlike an MCU, it can take around 20-40 seconds to boot and is not real-time deterministic for critical control loops. The boot time is not much of an issue as we would likely have it on all of the time, but the fact that it is not real-time deterministic makes this board undesirable for handling peripherals. This board also requires full OS maintenance, meaning we would need to update drivers, handle SD-card corruption, and patch security problems. This makes the project upkeep much more difficult as compared to just using an MCU. Furthermore, there is also an additional programming complexity because of the OS itself and the large ecosystem of programming languages. There is also limited analog capability in that there is no onboard ADC and would need external chips in order to read analog sensors. Furthermore, these boards are much more expensive than if we were to use an MCU. Typically, these boards cost around $35-$80 as compared to some MCUs that cost $5-$20 [12]. Lastly, these boards are also fairly large and require thermal management, meaning that it is not suitable for compact embedded enclosures.

So, overall the raspberry pi is a very good board that excels at camera streaming, image processing, and AI tasks that demand significant CPU/GPU power. They also provide full Linux networking and package support to their projects. Where it falls short is in real-time control, power-efficient always-on use, and deterministic hardware timing. Due to these critical disadvantages, we will move on from just using the raspberry pi by itself.

https://www.raspberrypi.com/documentation/computers/raspberry-pi.html [a]

https://www.raspberrypi.com/news/raspberry-pi-product-series-explained/ [b]

MCU + Raspberry Pi

Now we will consider using a dedicated MCU in tandem with a Raspberry Pi. This is one of three that were under heavy scrutiny as a realistic model of what we would use in this project. This hybrid configuration was considered because it combines the high processing capability of the Raspberry Pi with the real-time control precision of a dedicated MCU, offering a complete theoretical solution. There are many advantages to using this combination that we will discuss. However, there are slight disadvantages that make other methods more desirable and suitable for this project.

The primary aspect of this combination is that we would cover all the grounds of this project. The Raspberry Pi would handle video streaming, machine learning inference, and communication to the mobile application while the MCU handles precise, deterministic motor and sensor control. This is everything we need to handle for this project to work, thus arriving at the first consideration that provides a true solution. On top of finally arriving at a solution, this combination provides high processing power with real-time reliability. The Raspberry Pi provides a 1-2 GHz multi-core processor for heavy computing, such as streaming the video and handling the machine learning algorithm. Raspberry Pi's come with 1–8 GB of RAM, giving ample headroom for multitasking compared to typical MCU memory in the kilobyte range. At the same time, the MCU provides low-latency PWM, ADC, and interrupt handling.

This combination also brings energy optimization in that the MCU can remain active with its low-power mode and can activate the Raspberry Pi only when needed. This allows the system as a whole to be more sustainable and have a longer lifespan. In tandem with this point, power domain management becomes simpler. The MCU can control the Raspberry Pi's power or reset lines for safe reboots of the system. Another great benefit of this combination is that the software architecture has more accessible scaling. Machine learning and video processing on the Raspberry Pi can evolve independently of low-level control firmware, making software maintenance easier. Also, there is enhanced communication flexibility. This is because the Raspberry Pi has built-in Wi-Fi/Bluetooth and Ethernet communication modules while the MCU adds extra UART/I2C/SPI lines, analog inputs, and timers to the system. Furthermore, there is improved fault tolerance in this design because if Linux crashes on the Raspberry Pi, the MCU can maintain basic safety states like stopping the motor. And similar to the previous examples, there is an excellent community backing up the Raspberry Pi and likely the MCU as well. Both Raspberry Pi and popular MCUs like the ESP32 and STM32 families are widely available through major distributors, ensuring no supply issues for the hybrid design. There will be a vast number of open-source examples for both the Raspberry Pi and the MCU, reducing the difficulty of implementing the system.

Now, we will discuss the primary reasons we will not be using this combination in our project. One large contributing factor is that the design complexity of the project is increased quite substantially. There will be two firmware environments, Linux and the IDE for the MCU. This increases the learning curve and debugging time, since two programming languages and development ecosystems must be managed in parallel. These two environments must be coordinated, adding a fair amount of integration overhead. Another contributing factor is that an additional communication layer will be required. The data between the Raspberry Pi and the MCU must be exchanged over UART, I2C, or SPI and requires careful protocol design and error handling. This communication typically needs buffering, timing synchronization, and sometimes handshaking to ensure reliability between devices. This again increases the design complexity of the project. The debugging process will also take a long time and will be complicated, requiring cross-platform debugging tools and synchronized logs between the Raspberry Pi and MCU. Furthermore, synchronization mechanisms such as handshakes or watchdogs must be implemented to ensure a consistent state between the two devices. The PCB design will also be more complicated as we must route interconnects such as power and communication while also including proper level shifting and protection. We must also account for not being able to have the Raspberry Pi on the main PCB, and we must have some sort of external communication for it. Additionally, having two codebases will mean parallel development and testing cycles, meaning that this on top of the increased complexity of the project will lead to a longer development cycle.

Adding to that, the entire project will also be more expensive as we will be buying both the Raspberry Pi, which costs around $35-$80, and the MCU, which costs $3-$10. Lastly, since there are two devices there is a slightly higher total power draw, although power will not be a concern in this project. Combined operation could draw 5–9 W total depending on the Pi model and MCU activity. Note that all of the issues listed are not major roadblocks that will prevent us from completing the project with this method. This method is simply not the most efficient or most optimal method of completing the project. We will likely not need the processing power of the Raspberry Pi in tandem with an MCU, which means that we would be increasing the price, complexity, and development cycle of the project without adding significant functional benefit to the project's requirements. That is why we will not be using this combination as the control unit of the project.

ESP32 + Small Edge Accelerator

We will now consider using the ESP32 as the primary MCU paired with a small edge accelerator. The ESP32 is a popular choice as an MCU in industry as it is a very powerful MCU that brings many additional peripherals. The primary reason we are using the ESP32 in this combo is that it comes with Wi-Fi/Bluetooth connectivity while also providing a significant amount of power as an MCU. The ESP32-S3, for instance, includes dual-core 240 MHz processors and up to 8 MB of PSRAM, giving it strong performance for real-time control and moderate ML tasks [13]. This is in contrast to the STM32, which also provides industry level performance but does not have Wi-Fi/Bluetooth connectivity. The ESP32 is also very cheap at around $10, making it an affordable option as an MCU. Hence, we will be using the ESP32 for this consideration.

Alongside the ESP32 is a small edge accelerator, a specialized low-power processor designed to perform machine learning and neural network inference locally on embedded systems, rather than relying on a cloud server or high-performance computer. These accelerators are made specifically to run deep learning operations efficiently in hardware, using minimal power and memory. Examples include the Maxim MAX78000, Kendryte K210, and Syntiant NDP120, all designed for embedded AI inference [14]. Most small edge accelerators operate on quantized (INT8) models, use dedicated CNN or neural engines, and can run models such as MobileNet, LeNet, or custom posture detection networks in real time while consuming only milliwatts of power [14].

With this combo we have many advantages that make this one of the best options for this project. This system brings a balanced amount of performance and efficiency. The ESP32 would handle peripheral control, networking, and streaming while the small edge accelerator handles the machine learning inference with dedicated neural hardware. For comparison, using the small edge accelerator for the machine learning aspect is around 10-50x faster than the ESP32 alone [14]. This makes real-time posture or motion detection extremely feasible. Unlike using the Raspberry Pi with another MCU, this combo brings low power consumption. Together, the pair draws only a few hundred milliwatts while still supporting vision and inference. This is an order of magnitude lower than the 5–8 W required by a Raspberry Pi setup, making it ideal for continuous operation. This also means that we would have continuous, local machine learning detection without depending on using the cloud, improving privacy and responsiveness. Also, this is a compact and PCB-friendly design as both chips can be mounted on the same PCB side by side, making it easy to meet the project requirements of significant PCB design. Furthermore, we would have modular software design, meaning the machine learning and control firmware can be updated independently, improving the project's sustainability. Once again, with the ESP32 and the small edge accelerator, we would have a good community with a large amount of open-source software to learn from.

There are some downsides to using this combo. Most of the downsides come from having to develop two different devices at the same time. The first downside is additional integration complexity, meaning we would have to write and tune an interface between the ESP32 and the accelerator, including data transfer and synchronization logic. There is also more complexity when debugging as, again, there are two devices to debug, which require multi-stage validation. There is also a longer development time as setting up machine learning toolchains and SPI protocols takes more time than a single-MCU implementation. The learning curve for each accelerator’s SDK and model conversion process adds further difficulty, especially for teams new to embedded ML.

On top of the typical downsides of having to work with two devices, there is also a limited small edge accelerator ecosystem. This means that each accelerator uses its own SDK and model-conversion tools, which can increase setup time and complexity. Furthermore, models must be quantized and compiled for each accelerator's architecture, limiting flexibility compared to desktop frameworks. Also, accelerators are not as accessible as say a typical MCU or a Raspberry Pi. Some accelerators have limited distributor availability or longer lead times than standard MCUs, meaning that this is not as replicable as other considerations. For example, the MAX78000 and NDP120 are primarily sold through select distributors, which can delay prototyping [14]. There is also a higher cost, simply because we would be buying another device on top of the MCU, meaning we would add roughly $5-$20 to the BOM depending on the accelerator. Lastly, there is limited RAM for large models, so these small edge accelerators are not equipped for handling complex architectures. However, this will likely not be a problem as our algorithm is not very complex, but it should still be noted. Once again, all these downsides are not capable of making this consideration a non-option for this project. However, these downsides are more significant than our final option, thus we will not be going forward with this design. While this configuration meets all functional requirements, its integration overhead and limited ecosystem make another solution more practical for our timeline.

https://www.edn.com/top-10-edge-ai-chips/ [c]

https://www.espressif.com/en/products/socs/esp32 [d]

ESP32

We now arrive at our final consideration and our choice for the MCU in this project. This will be using the ESP32 as the MCU with the machine learning algorithm hosted on the backend of the mobile application. This is what we have arrived at as the most practical and efficient solution for our project. We will be using the ESP32 as our MCU because of its many benefits, specifically its wireless communication along with its industry grade performance. The ESP32 will be in charge of managing all of the peripheral devices such as the motor, microphone, and camera while also running a video stream and sending it to the mobile app when necessary. The mobile application will be displaying the video stream from the camera while also hosting the machine learning algorithm that detects whether or not the dog is sitting. This reduces the load from the ESP32 and makes it capable of handling its tasks.

We will now discuss all the large benefits that come with this choice. The largest contributing factor to this choice is that it is the simplest overall design. We will only have one device that we need to program and integrate, reducing hardware and firmware complexity. This drastically shortens both integration and debugging time compared to multi-device solutions. All we need to program is the communication between the MCU and the mobile app, and the rest is what we would have already needed to program on the MCU, meaning there is a minimal amount of additional programming necessary. This is in comparison to the Raspberry Pi + MCU and the ESP32 + Small Edge Accelerator, which both need device-to-device communication and synchronization. Furthermore, the ESP32 has integrated wireless communication, making direct app connectivity simple, and eliminating extra modules.

This combination also comes with a fast development cycle as there are many examples and libraries for video streaming, MQTT/HTTP communication, and mobile integration. The ESP32 also has a very strong open-source ecosystem and documentation with many learning materials available freely. This reduces the learning curve of using the ESP32 as our MCU. Additionally, ESP32 modules are widely available from major distributors, making this a highly obtainable and replicable choice. Hosting the machine learning algorithm on the backend of the mobile application also means that the software is easily scalable. The firmware on the ESP32 is unlikely to be updated as it handles simple and straightforward operations while the mobile app will be handling the machine learning algorithm, which is the most likely to be updated. This separation of responsibilities improves long-term maintainability and update flexibility. This means that we can simply update the mobile app without having to reflash the ESP32 firmware. Another benefit is that using the ESP32 as the only device means that this design has low power consumption with a measly 200-400 mA drain while streaming [13].

There are some downsides to discuss for this choice, but we decided that they were the least consequential to our project of all the choices while still providing very strong benefits. Firstly, since the machine learning algorithm is hosted on the mobile app, we require constant network connectivity. However, this is not a large concern as we assume that the user will be connected to Wi-Fi regardless of whether they are using the mobile app or not. There will also be potential network latency as the response time depends on Wi-Fi speed and backend processing load. Once again, this is not a large concern as we are not attempting to construct a version of this project that is ready to be deployed as a product; we simply want to use this project as proof of the concept that this sort of product is feasible. Even with small latency, the application can still respond within seconds, which is acceptable for feeding operations. There are some privacy and bandwidth concerns as sending image frames for inference can raise data usage and privacy issues unless optimized. We can put this down once again as something that is not a large concern for us.

There is also some backend maintenance overhead as we need to maintain a reliable server or cloud function for machine learning processing and updates. However, we are likely not going to further develop this project outside of our senior design ecosystem, thus it is not a concern. Since the machine learning algorithm is hosted on the mobile app, if the user is offline, the feeder cannot perform posture detection, but we will have a manual override button for cases such as this. This manual override also enhances safety and ensures usability in all conditions. We also need to secure API endpoints and authentication between the ESP32 and backend to prevent data leaks, but once again this is not a large concern for us. The only concern that has some weight behind it is the back-end hosting cost. Cloud storage and app-server expenses are typically subscription-based services, meaning the cost will be confounding. However, these services typically do not cost that much at around $5-$10 per month, and considering that there are no other significant downsides, this choice makes the most sense for our project. The ESP32-only architecture therefore provides the best balance of simplicity, functionality, and cost-effectiveness, making it the optimal choice for our smart feeder's final design.

https://www.espressif.com/en/products/socs/esp32 [d]

Table 3.1.2: Types of Computing Systems

Category	STM32	Arduino	Raspberry Pi	Raspberry Pi + MCU	ESP32 + Small Edge Accelerator	ESP32 (Final Choice)
Power Consumption	Moderate to Low — efficient; some ultra-low-power models consume <100 µA in sleep	Low to Moderate — efficient for simple tasks but less optimized for power modes	High — typically 5–8 W active; requires 5V/3A supply and cooling	High — Raspberry Pi dominates power draw; MCU adds negligible load	Very Low — combined draw of a few hundred mW; ideal for always-on IoT	Low — 200–400 mA during streaming; excellent for continuous use
Cost	Moderate — $5–$15 depending on model and external wireless modules	Very Low — $5–$10 for Uno/Nano; <$30 for advanced boards	Moderate to High — $35–$80 for Pi 4/5 models	High — Pi + MCU ($40–$90 total) + integration cost	Moderate — ESP32 ($10) + accelerator ($5–$20) ≈ $15–$30 total	Very Low — ~$8–$12 for ESP32 module only

Processing Power & Speed	High — up to 480 MHz dual-core (STM32H7); excellent for real-time control	Low — typically 16 MHz, 8-bit (ATmega328P); poor for heavy tasks	Very High — 1–2 GHz quad-core; full Linux OS support	Extremely High — combines Pi's CPU and MCU's real-time control	High — 240 MHz dual-core MCU + dedicated ML inference hardware (10–50× faster ML)	Moderate to High — 240 MHz dual-core with strong multitasking for streaming and control
Obtainability	Excellent — available from major distributors (ST, Digi-Key, Mouser)	Excellent — globally available; sold by Arduino and third parties	Good — widely sold, though occasionally short supply during global demand spikes	Moderate — requires sourcing both Pi and MCU; availability depends on both	Moderate — ESP32 widely available, but accelerators (MAX78000, K210) have limited distributors	Excellent — ESP32 modules are abundant and well-supported worldwide
Complexity to Program	High — requires CubeIDE/CubeMX, HAL, RTOS; steep learning curve	Very Low — beginner-friendly Arduino IDE and huge community	Moderate — Linux environment allows Python/C++ but adds OS overhead	High — two platforms (Linux + MCU) require synchronization and complex debugging	High — requires dual toolchains (ESP-IDF + accelerator SDK); ML model conversion adds complexity	Low — single firmware with libraries for Wi-Fi, video, and app communication; fastest development cycle

Communication Protocols	UART, SPI, I²C, CAN, USB, Ethernet (no built-in Wi-Fi/BLE)	UART, SPI, I²C (no built-in Wi-Fi/BLE; requires shields)	Wi-Fi (Ethernet optional), Bluetooth, USB, GPIO	Wi-Fi/BLE via Pi + SPI/I²C/UART between Pi and MCU	Wi-Fi, BLE (from ESP32) + SPI/I²C/UART between MCU and accelerator	Wi-Fi, BLE (native) + HTTP, MQTT, WebSocket for backend communication; no external modules needed

ESP32-WROOM-32

Now that we have arrived at a proper MCU family, we should now choose which specific MCU we should pick. There are many options in the ESP32 family, but we will only be considering three. We will be choosing three considerations to represent different sides of the ESP32 family. The first consideration will be the ESP32-WROOM-32. This is a strong option for an MCU, bringing a Dual-core Tensilica LX6 CPU at 240 MHz along with 520 KB SRAM and 4 MB Flash [15]. This is an older model, but it still brings a CPU architecture that offers strong performance for real-time control and multitasking. The main reason we would like to consider this chip is because of its proven, mature platform. This is the original and most widely used ESP32 design, ensuring strong long-term stability and broad hardware compatibility. It is extensively integrated into commercial IoT boards and educational kits, ensuring easy prototyping and replacement. Furthermore, it is the cheapest ESP32 module pricing around $5-$8, making it one of the most affordable options in the family while still offering dual-core performance.

There are some downsides to this chip, however. The primary concern is the performance of the chip. This chip is relatively old and offers decent performance for its age, but we would like to have the most powerful chip we can get for our design to ensure that our MCU can handle all its tasks. We know that the ESP32 can handle tasks, but we simply do not have a good understanding of how well it can complete them. So, having the most powerful ESP32 would ensure that it can complete the tasks at the highest possible level. To be specific on how the WROOM-32 performs, we are concerned about the limited SRAM. We are concerned that the 520 KB of SRAM may constrain memory-intensive tasks like buffering images or handling multiple concurrent network operations. The WROOM-32 also lacks external PSRAM, which limits that headroom for applications that need large memory buffers. It also does not have a native camera interface, which is desired to reduce additional project complexity. It also does not have native USB support, requiring an external USB-to-UART converter for programming and debugging. So overall, due to the limited performance and peripherals, we will not be using the ESP32-WROOM-32 as our specific MCU.

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf [a]

ESP32-CAM

We will now be considering the ESP32-CAM as another option for our MCU. This MCU brings with it an ESP32-S module with 4 MB Flash and 8 MB PSRAM, along with an OV2460 camera module pre-integrated with it [16]. This is a strong choice for an MCU as it comes with a built-in camera module, bringing immediate image and video capture capabilities with no external wiring or configuration required. It also has an integrated microSD card slot, meaning if we wanted to store the video recordings, we would have a module on-hand capable of doing so. Furthermore, it is fairly cheap at a price of $7-$10. Furthermore, its 8 MB of PSRAM provides enough memory for JPEG image compression and low-resolution video streaming [16]. This chip brings very strong performance while also integrating a camera at the same time, making it a great choice for an MCU.

However, it does have some downsides that take it out of consideration. The primary concern with this chip is specifically the mechanical aspect of it. It would be fairly and unnecessarily difficult to position the camera in the desired position for the project. The ESP32-CAM is built such that the camera would be positioned right on top of the MCU, meaning that we would have to position it so that the board would be on the face where the camera would be resting. This makes it very awkward to design the PCB itself and furthermore position it. We would also be limited to the one camera designed for the ESP32-CAM, limiting our freedom of choice. There are also other downsides such as a lack of a USB interface, meaning we would have to program it through a UART adapter, making development less convenient. There is also a limited amount of GPIO pins as many of them are used by the camera and SD interfaces, leaving few for sensors and other peripherals. It also doesn't have a DVP/CSI camera interface, meaning that the performance of the camera is adequate, but it isn’t optimized for high-speed capture. In conclusion, we will not be using ESP32-CAM mostly due to the awkward nature of using the camera and the lack of freedom of choice for the camera itself.

https://components101.com/modules/esp32-cam-camera-module [b]

ESP32-S3

As our final consideration, we will be using the ESP32-S3 as the MCU for our project. The S3 has many benefits that make it extremely attractive as an MCU. For one, the D3 brings the newest high-performance variant of the ESP32. It has a dual-core Tensilica LX7 CPU running at 240 MHz with improved instruction efficiency and architecture over the LX6 [17]. A primary concern for the MCU in this project is its performance, so the S3 bringing such strong performance capabilities immediately places it as a top contender. Furthermore, if the performance is not enough, there is support for up to 8 MB of external PSRAM, providing sufficient memory for the tasks necessary [17]. There is also a built-in USB OTG interface, which allows for direct programming and debugging through USB-C, not needing an external UART adapter as compared to the previous two considerations. This makes programming with the S3 easier and less tedious. The S3 also brings a native DVP camera interface, providing official support for parallel 8-bit camera modules for high-speed image capture. Additionally, there are enhanced peripherals and an enhanced GPIO matrix, expanding PWM channels, improving SPI/I2C flexibility, and providing more pins than previous models [17]. The efficiency of the low-power mode is also improved, optimizing sleep currents and peripheral clock gating to reduce the overall power consumption as compared to previous chips.

There are some slight downsides to discuss, but they are very limited and have little impact on the project. Firstly, some modules have limited built-in flash memory and may require external flash or PSRAM chips for advanced applications. However, we can simply buy a version of the S3 with lots of memory to accommodate this. The S3 also has a slightly newer ecosystem as compared to other models, meaning there are fewer tutorials and examples available. Again, this has little impact as there are minimal programming differences between models and even so, there are still examples to draw from should there be a problem. Now as there is an increase in performance, there is also an increase in current draw when under load. However, once again this is not a big concern as we have continuous power flowing into the system and we will already consider the voltage and current necessary for each part of the system. Lastly, there is a slightly higher cost with an average price of $9-$12, but this is still quite cheap compared to other considerations.

To ensure that we have sufficient performance, we will be choosing the best performing S3 possible. So, we will be using specifically the ESP32-S3-WROOM-2-N32R8V. This chip has 32 MB of flash memory as well as 8 MB of PSRAM, along with a PCB antenna and costs around $10.67 on DigiKey. The ESP32-S3-WROOM-2-N32R16V brings 16 MB of PSRAM instead of 8 MB, but it is currently out of stock and will be restocked in the middle of December 2025. This is too long, so we will settle for the R8V instead. To conduct proper prototyping before assembling the PCB, we will be using the ESP32-S3-DevKitC-1. This lets us start programming without needing the PCB or other components.

https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf [d]

Table 3.1.3 MCU Comparison

Feature / Capability	ESP32-WROOM-32	ESP32-CAM	ESP32-S3
CPU Architecture	Dual-core Tensilica LX6 @ 240 MHz	Dual-core Tensilica LX6 @ 240 MHz	Dual-core Tensilica LX7 @ 240 MHz (improved performance and efficiency)
RAM / PSRAM	520 KB SRAM (no PSRAM on most modules)	520 KB SRAM + 8 MB PSRAM	512 KB SRAM + up to 8 MB external PSRAM
Flash Memory	Typically 4 MB	4 MB (integrated on board)	Up to 32 MB (module dependent, e.g., WROOM-2-N32R16V)
Camera Support	None (no DVP/CSI interface)	Built-in OV2640 2MP camera (via GPIO)	Native DVP camera interface (supports OV2640, OV5640)
MicroSD Slot	None	Built-in	None (external interface possible)
USB Interface	Requires external UART adapter	Requires FTDI programmer	Native USB OTG — direct flashing and debugging
Bluetooth Version	4.2 (Classic + BLE)	4.2 (Classic + BLE)	5.0 LE (improved speed, range, and stability)
Peripheral Interfaces	SPI, I²C, UART, I²S, PWM, ADC/DAC, GPIO	SPI, I²C, UART (limited GPIO due to camera/SD use)	SPI, I²C, UART, I²S, CAN, PWM, ADC/DAC, GPIO (expanded pin matrix)
GPIO Availability	30+ usable GPIOs	6–7 available after camera/SD use	30+ usable GPIOs (depending on module configuration)
Wireless Performance	Wi-Fi 2.4 GHz + BLE 4.2	Wi-Fi 2.4 GHz + BLE 4.2	Wi-Fi 2.4 GHz + BLE 5.0 (improved throughput)
Ease of Programming	Easy; widely supported by Arduino and ESP-IDF	Moderate; requires UART flashing and external power stability	Easy; native USB simplifies programming and debugging
Cost (approx.)	$5–$8	$7–$10	$9–$12

3.1.3 Mobile Communication Technologies

Choosing a mobile communication technology is of vital essence as it determines how the system is going to communicate with the mobile application. For this project to achieve the expected design, we must have a mobile application that controls the system. Therefore, we need to have a reliable way to wirelessly communicate between the two. As such, we will consider Bluetooth, Zigbee, LoRaWAN, and Wi-Fi as our possible solutions.

Bluetooth

Bluetooth is a short-range wireless communication technology that allows devices to connect and exchange data over short distances. This can be seen in a wide variety of devices such as phones, headphones, keyboards, smartwatches, and many more. Bluetooth uses radio waves as a way to send data between devices, acting in a similar fashion to Wi-Fi but for shorter distances and typically consuming less power. In order to connect, the two devices must pair with each other by establishing a secure connection, saved for future use and remembrance. Bluetooth is meant for short distance connections, having a typical connection range of 10 meters, although newer models can have longer distances [18]. Since the whole purpose of the project is to be able to view and control the dog feeder while not being at home, this short distance immediately eliminates Bluetooth as a consideration for our communication technology.

https://www.lenovo.com/us/en/glossary/what-is-bluetooth [a]

Zigbee

Zigbee is a specification for a suite of high-level communication protocols used to create personal area networks with IoT devices. It is a low-power, low-data-rate, and close proximity wireless network. This is essentially a simpler and less expensive version of Bluetooth. This is typically used for devices like light switches, home energy monitors, traffic management systems, and other equipment that needs short-range low-rate wireless data transfer [19]. Its wireless communication has a distance of 10-100 meters depending on environmental characteristics and line of sight. Zigbee is especially useful for a network of devices as certified Zigbee devices from different manufacturers can work together seamlessly [20]. It is also quite secure, using 128-bit AES encryption and can support thousands of devices simultaneously. Once again, the range of Zigbee devices immediately eliminates Zigbee as a possible communication technology.

https://en.wikipedia.org/wiki/Zigbee [b]

https://csa-iot.org/all-solutions/zigbee/#:~:text=Zigbee%20is%20the%20full-stack,will%20keep%20working%20for%20you [c]

LoRaWAN

LoRaWAN is a low-power, long-range wireless communication protocol for IoT devices across wide areas. LoRaWAN is similar to Zigbee in that it can be used to create networks with IoT devices, but the main difference is that it has a much longer range. Its range can extend up to several kilometers while also supporting a multi-year long battery life for its devices. This technology is typically used in smart cities, agriculture, and industrial applications [21]. Similar to Zigbee, it implements AES-128 encryption to secure its data. This is a very strong choice for devices that are in a fixed location but have a long distance between them. This choice has also been eliminated due to the range. Although the distance is significantly longer than the two previous considerations, we intend for the user to be able to access the dog feeder from anywhere in the world as long as they have internet access. So, while the range is quite large, it is not as broad as we need it to be.

https://www.cisco.com/c/en/us/solutions/internet-of-things/lorawan-solution.html [d]

Wi-Fi

Wi-Fi is a wireless communication technology that allows devices like computers and phones to connect to the internet through a router using radio waves. It operates under the IEEE 802.11 standard, defining the protocol that enables devices to communicate. This form of communication is used worldwide in many public spaces such as restaurants, hotels, libraries, airports, and more. The range of an access point is around 20-150 meters, depending on line of sight [22]. The difference between Wi-Fi and the other technologies is that while there is a small range, since it is used almost everywhere the user will be, the user will be able to access the internet. This makes using Wi-Fi as the communication technology the most reliable form of communication between the mobile app and the ESP32. As such we will be moving forward with Wi-Fi as the communication technology.

https://en.wikipedia.org/wiki/Wi-Fi [e]

Table 3.1.3 Mobile Communication Comparison

Feature	Bluetooth	Zigbee	LoRaWAN	Wi-Fi
Frequency Band	2.4 GHz ISM band	2.4 GHz ISM band (some sub-GHz variants)	Sub-GHz (typically 868 MHz EU, 915 MHz US)	2.4 GHz and 5 GHz ISM bands
Range	~10–100 m (depending on version and power class)	~10–100 m (up to 300 m line-of-sight)	Up to 10–15 km (rural), 2–5 km (urban)	~50–100 m indoors, 300 m outdoors

Data Rate	1–3 Mbps (Bluetooth Classic), up to 2 Mbps (BLE 5)	20–250 kbps	0.3–50 kbps	Up to several hundred Mbps (Wi-Fi 5/6)
Power Consumption	Very low (BLE optimized for battery)	Low (designed for mesh IoT)	Ultra-low (for long-term battery use)	High (requires constant power)
Range vs. Power Trade-off	Short range / low power	Moderate range / low power	Long range / ultra-low power	Short-medium range / high power
Latency	Low (a few ms)	Moderate (tens to hundreds of ms)	High (seconds typical)	Very low (a few ms)
Reliability	High for short links	High via mesh redundancy	Moderate (depends on network load)	High (robust TCP/IP)
Security	AES-128, frequency hopping, pairing modes	AES-128 with network and link keys	AES-128 end-to-end encryption	WPA2/WPA3 encryption
Best Use Cases	Wearables, short-range sensors, smartphone interfaces	Home automation, sensor networks, lighting control	Smart agriculture, metering, remote IoT	High-bandwidth apps, video streaming, data-intensive IoT
Advantages	Low cost, easy smartphone integration, energy efficient	Strong mesh, reliable low-power IoT	Long range, extremely low power, ideal for remote IoT	High data rate, global standard, easy internet connectivity
Disadvantages	Limited range and connections	Lower data rate, requires coordinator	Very low data rate, higher latency	High power usage, limited range for battery IoT

3.1.4 Camera

The camera we choose will play a pivotal role in the specific features that we aim to achieve for our automatic dog feeder. The automatic dog feeder will run through a mobile app that the owner will be allowed to use at any time. Once on the mobile app the owner will have the ability to join a live feed of the camera at any point in time. Providing the owner with multiple benefits that range from checking in on their pet when they have been away from home for extended periods of time, making sure that their pet has eaten when their scheduled feeding time occurs, and entertainment for times when they want to speak to their pet or watch what their pet is doing when away from home. To achieve these features, there are a couple of important qualities that we want our camera.

The first feature that the camera has to have is decent to good camera quality. More specifically, we want our camera to have relatively good resolution and frame rates) When the owner of the dog is away from home and wants to check in on their pet or they want to use the camera for entrainment purposes the camera needs to produce sufficient quality of the live feed from the mobile app in order for the owner to be able to effectively check in on their pet. If we were to buy a camera that didn't have the best quality the owner wouldn't be able to effectively check in with their animal and would make this feature relatively useless. It would also reduce the overall enjoyment and entertainment that this feature provides for the owner. Another feature in our product that requires relatively good camera quality is training and the detection of the dog. If we were to purchase a camera with lower quality, we risk the potential that the camera doesn't detect the dog walking over to the product or when the dog has sat down. Which overall, could cause the dog not to be fed from the camera not realizing that the dog has sat down thus not sending the command to start dispensing food for the dog.

Another important aspect that we want in our camera is the ability to implement it into our project smoothly. When looking at various cameras, the first feature that we wanted our camera to have is the ability to physically fit into our pet feeder, one way we aim to do this is by ordering a camera that is smaller. A smaller camera would be easier to fit into the designed spot that we are aiming to have the camera. It would also help to minimize the amount of space that the camera takes up, giving us more room for other equipment for our project. Another feature that we want the camera to benefit us by physically implementing it into our project is per drilled holes and a mount implemented into the camera. This would help save us time when it comes to physically implementing since we don't have to buy a mount or drill the holes ourselves and can just use the ones already implemented to do that.

The next major feature that we want our camera to have is the ability to easily integrate into our PCB design and our software. Theres a wide variety of features that a camera can contain that make it easier to integrate, but some of the important things our group are looking for consist of compatible logic levels, clear pinouts, and interface. For our logic levels, we want our camera to operate at the same level logic that our MCU operates at. The ESP32 – S3 operates at 3.3 V GPIO's and many cameras operate at a lower level which would force us to use level shifters so that our camera doesn't damage itself. Adding level shifters would force us to spend more money on the project and take up space so when researching what camera would fit into our project the best, we want a camera that can also operate at the same logic level. When researching which camera would fit our project the best, we also want a camera that has clear, labeled pinouts. Purchasing a camera with standardized pinouts would allow us to connect each peripheral to the matching one. If we were to purchase a camera that didn't have the same pinouts, we would have to cross reference each datasheet to connect the right pins, which would cost our group an extensive amount of time and could lead to errors. The next feature we looked over was the compatibility between our camera and our MCU. Purchasing a camera that uses an interface that the MCU directly allows us to communicate between the two without extra complications being required. Complications that could result in spending more money, some instability, a larger power consumption, and would increase the overall complexity.

OV2640

The OV2640 camera was one of the first cameras that we considered for our automatic dog feeder from its easy integration into most MCU's, mainly ESP32 MCU's. Most MCU's like the ESP32 already contain a prewritten driver for the OV2640. Which would allow us to integrate the camera onto the system in significantly less time than other cameras. Some of the prewritten features that the OV2640 contains consist of the ability to initialize the camera hardware, being able to set settings of our resolution, frame rate, and other parameters, and the conversion to JPEG. The OV2640 also operates at a 3.3V logic level, which was one of the important features that we wanted our camera to have. Stemming from the idea that most MCU's on our list and in general contain a 3.3V logic level as well [23]. If we were to have an MCU that operated at 3.3V logic level and a camera that operated on a different logic level it would force us to have to use a level shifter so our logic levels were matched. Since this camera already has a logic level the same as most MCU's, it would allow us to not have to use one. In return, saving our group time and money, in the overall idea would simplify the design of our PCB and increase the overall reliability that we have in the bridge between our MCU and our camera.

https://tang.sipeed.com/en/hardware-overview/ov2640-camera/

The OV2640 operates at roughly around 100-200mW during its active phase and requires a 3.3V input which is a relatively low power consumption when compared to other cameras, which can have its pros and cons in the qualities we aim to have in our camera [24]. One of the first benefits being that the camera is efficient when it comes to the energy it uses. The first thing this would impact is the overall cost of our product, since we are spending less on the overall energy that we are using to power the system. The next thing that would be directly impacted is the overall heat generation of our product, where other cameras with higher energy requirements would generate more heat. Although we don't expect heat to be a significant problem within our design, purchasing a camera that produces less heat is still a benefit. Having too much heat throughout the camera could also create a problem with the image consistency and the age of the camera.

https://www.uctronics.com/download/cam_module/OV2640DS.pdf

Although lower power consumption does increase the efficiency of our product, there are a range of negative effects that it also creates. The first impact that it has is a reduced power range and higher noise within the camera. Which could result in the camera being unable to recognize fine detail, the depth of color, and any low output. Since our product relies on the ability to recognize when the dog has approached the product and when the dog has sat down, this raises a significant problem with the reliability of this camera being implemented onto our design. If we are unable to detect the dog sitting or approaching the camera, then the rest of the operation towards feeding the dog will not begin, resulting in the dog not being fed and going hungry.

When designing and implementing our PCB for our camera, the OV2640 allows for a simpler and smaller design from its low energy consumption and higher energy efficiency. The OV2640 has a standardized pin interface which also helps to simplify integration with our microcontroller while also simplifying the regulator that we use from its power consumption and current draw [24]. However, the ability to have a simpler design has its drawbacks when designing the PCB board. Design challenges such as specific voltage regulation, image noise from voltage drops, and specific PCB line tracing to be able to keep our signal. To conclude that although it does help to simplify the design of our PCB, there are also specifics that we must make sure to design to be able to fit the requirements of the OV2640 camera.

https://www.uctronics.com/download/cam_module/OV2640DS.pdf

OV5640

The OV5640 was another camera that we considered using for our product from its ability to be easily integrated into our MCU, its high resolution, camera features, reliability, and its cost. At first glance, our group found the features of the OV5640 to fit exactly what we needed for our project, but we didn't want to make any decisions too quickly. But after doing the bulk of the research on this camera our group decided to finalize using the camera for its ability to work easily and implement easily into the MCU that we had picked out (ESP32 S3) and its high resolution and specifically its ability to identify faces and track gestures.

The first feature of the OV5640 that our group valued was its high resolution and camera ability. It has a image resolution of 2592 x 1944 pixels which helps to produce extremely detailed and correct color images, which is important when we need our camera to be able to detect the dog approaching the product and when the dog has sat down. The camera also has the ability to translate around 5 million pixels, which will allow the camera to be more reliable under strict light conditions, for example if the lights in the house were turned off or it was dark outside [25]. With our product being extremely dependent on the cameras ability to be able to pick minute details such as the dog approaching the product and the dog sitting purchasing a camera with the ability to pick up and operate under varying conditions is extremely important, and the OV5640 camera contains a significant amount of features that solve these problems that could occur when operating.

https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/OV5640_datasheet.pdf

The next feature that the OV5640 camera has is the ability to be easily implemented in the MCU that was using which is the ESP32-S3. The OV5640 and the ESP32-S3 have matching electrical compatibility, the same protocols, and the same software support, so transmission between the MCU are the camera will be smooth and reliable. They also both operate at the same logic level with that being 3.3V. As stated, when evaluating what features we were looking for when deciding which camera to use, having the same level logic is extremely important because we don't have to use a level shifter in our design, which would save us time and money when implementing. It would also create a more reliable and stable connection to the camera when in operating mode. The OV5640 contains an 8-bit data output that directly matches the ESP32-S3's DVP [25]. Because these match, it allows us to not have to use any type of software conversion which would result in the slowing down of performance. Since we want the camera to be able to operate in real time and in high resolution when detecting the dog approaching the camera and the dog sitting, its extremely beneficial to have a MCU and camera with the same interface. The interface of the OV5640 and the connection that it has with our MCU will also allow us to manually set up the camera for specific resolutions [26]. For example, we would be allowed to adjust the brightness or frame size. Allowing our product to be able to operate at different lighting conditions and if the dog is far away from the camera. The ESP framework also already has native support for the OV5640, making any type of low-level initialization and the setup of registers extremely easy. Which would allow us to focus more on developing the AI to identify the dog approaching and sitting instead of spending our time working on debugging or working on interface problems. The connection between the two also would allow for real-time image recognition without putting too much of a workload on the storage hardware from the internal JPEG encoder that the OV5640 contains.

https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/OV5640_datasheet.pdf

https://download.kamami.pl/p1180726-OV5640_Camera_Board_%28B%29_User_Manual_EN.pdf

Realtek AMB82

The next camera we considered using was the Realtek AMB82. The first thing we noticed about this camera was its AI capabilities to be able to detect motion, pet recognition, and when the pet has sat down. Where the camera is able to run machine learning directly on it which would benefit real time decision making when the camera needs to identify if the dog is in the view, when the dog begins to approach the camera, and when the dog has sat down in order to receive its food [27]. Although other cameras can also take out this task, the advantage toward using this camera is that it doesn't need any type of external server to run it. Having AI features within the AMB82 also allows for compatibility with varying computer vision and network interfaces. Which would benefit our product by being able to customize the AI to fit exactly what we wanted it to do, which would be identifying if the dog is present, approaching the camera, and sitting down for the food to be dispensed.

https://www.realtek.com/images/ar/2024_Annual_Report_.pdf

Along with the AMB82 camera being able to utilize AI to learn specific positioning of the dog the camera also has high quality imaging to also help with it. It contains 1080p image censoring to be able to pick up clear and specific visuals that will help to pick up the specific positions that the dog is in [28]. While other cameras with worse resolution might not be able to pick up the dog or the dog's position which would lead to the dog not being fed, this camera's high resolution and AI both help to make sure that doesn't happen. Along with that, this camera also has low light sensitivity to prevent false tasks from being carried out during times of low visibility like if the lights were off or it's too bright in the house from the sun.

https://robu.in/wp-content/uploads/2023/03/Realtek-AMB82-Mini-USER-MANUAL.pdf

Although this camera contained a couple of important features that we were looking for, ultimately there were a list of reasons we ended up not using it. For a project where everyone is relatively doing everything for the first time, it was important that we picked a camera that had a larger developer ecosystem. For example, with the Ov5640 there are significantly more tutorials and libraries that we can use as examples. Where the ecosystem of AMB82 is much smaller with much less tutorials. The AMB82 also doesn't contain the best flexibility if we need to change the resolution or the field of view of our camera. Which could introduce a problem when testing out our product for the first time, if the camera has a high failure rate because it wasn't picking up the dog or the resolution wasn't what it needed to be to pick the dog up at a high rate we wouldn't be allowed to change that leading in a significant loss of time and money. AMB82 also isn't as widely available to the other cameras that we considered using, so any type of faulty equipment of change in design would take significantly longer than if we were to use one of the other two cameras.

Table 3.1.4 Types of Camera

Features	OV2640	OV5640	Realtek AMB82
Resolution	2 MP (1600×1200)	5 MP (2592×1944)	2 MP (1920×1080)

Image Size	Up to UXGA (1600×1200)	QXGA (2592×1944)	Full HD 1920×1080
Frame Rate	15 - 30 fps	30 fps / 1080p	30 fps / 1080p
Field of View	55°–75°	60°–90°	110°–130°
Low-Light Sensitivity	Decent but needs good lighting	Good low-light and WDR performance	Good, can configure low light and WDR
Integration to MCU	DVP/SPI, need MCU (good with ESP)	DVP/MIPI, also need MCU (good with ESP)	Integrated SOC with wifi and bluetooth
Wi-Fi Capability	No Wi-Fi (Needs MCU)	No Wi-Fi (Needs MCU or board)	Wi-Fi (2.4 GHz / 5 GHz)
Cost	$5–$20	$10–$35	$20–$40
Power Consumption	~100–200 mW when active	~200–400 mW when active	~0.5–1 W when active
Power & Efficiency	Low power, good efficiency	Moderate	Higher
Size	≈25 × 24 mm	≈30 × 28 mm	≈40 × 30 mm
Ecosystem Support	Large ecosystem with support	Good Support, many tutorials	Less support compared to the other cameras

3.1.5 Motor

For the feeder's dispensing mechanism, the motor is obviously one of the most important parts. It is what physically moves the food out of container, so we need something that's strong, reliable, and precise.

The motor plays a vital role in converting electrical energy into mechanical rotational motion, which drives components such as an auger, wheel, or flap in the dog feeder. The motor's precision ensures that each portion of food dispensed is consistent and accurate. This accuracy typically depends on the repeatable steps or rotations of the motor and, in some cases, closed-loop feedback from a weight sensor that monitors the output.

Several motor types can be used in automatic feeding systems, and each with its own advantages

- A ***bipolar stepper motor*** (such as the NEMA-17) offers excellent precision, with a typical step angle of 1.8° - meaning 200 steps per revolution [29, 30, 31]. This fine control, combined with microstepping capability, provides accurate "dose by turns" operation. Despite its compact 42 mm frame, it can deliver a solid holding torque of around 3.7 kg·cm, making it ideal for metering mechanisms like augers [29].
- A ***DC geared motor***, on the other hand, is more cost-effective and provides high torque output [32]. However, to achieve precision in food dispensing, it usually requires additional components like an encoder, clutch, or extensive calibration [32]. When picking a DC motor for a pet feeder, we focus on getting the right balance of voltage, current, and torque [33]. Too much power can break the food or make it noisy, so we prefer micro DC gear motors for their quiet and efficient performance [33]. The motor should provide just enough force to move the dispenser smoothly. A DC motor with a planetary gearbox offers the best precision and reliability for consistent feeding.
- An ***RC servo*** motor offers simple PWM-based control, which makes it easy to use for basic movement, such as opening and closing a flap gate. However, it is limited in torque and lifespan, especially under continuous or heavy-duty rotation, unless paired with a winch or gearbox.

Table 3.1.5: Comparing types of Motor

Motor	Pros	Cons	Notes
Stepper (NEMA-17)	Open-loop precision, easy microstepping, stall detect with modern drivers	Higher current draw and heat when holding, more expensive	Best choice for auger metering
DC + gearbox	Low cost, efficient	Requires encoder or feedback loop for precision	Suitable for timed dispensing systems
RC Servo	Simple control set up	Limited torque and shorter lifespan	Ideal for flap gates only

The good choice for a dog feeder mechanism is the NEMA-17 stepper motor (42 × 38 mm, 1.7 A/phase). This motor provides 200 steps per revolution and a holding torque of approximately 3.7 kg·cm, offering a reliable balance between precision and power. It's compatible with a wide range of common motor drivers, making it easy to integrate into microcontroller-based systems. For linear dispensing applications, a NEMA-17 model with an integrated lead screw can also be used, providing accurate feed control along a linear path.

3.1.6 Motor Driver

Motor drivers act as the bridge between the microcontroller (MCU) and the motor itself. They translate low-power control signals - such as step and direction commands for steppers or PWM signals for DC motors - into precise, high -current outputs that drive the motor coils. These drivers also handle key performance aspects like microstepping control, current regulation, and built-in safety protections such as over-current and thermal shutdown. Without a reliable driver, the motor cannot operate smoothly or safely, especially under varying load conditions.

Two popular stepper motor driver options stand out for feeding applications:

- The DRV8825 supports up to 2.5 A and 45 V, with microstepping up to 1/32 steps [34]. It uses a chopper current regulation technique, which helps maintain consistent torque and motion precision. This driver is widely available on affordable carrier boards and is known for being robust and easy to integrate, making it a great choice for straightforward, reliable setups.
- The TMC2209 offers more advanced capabilities. It supports up to 2.8 A peak (about 2A RMS) and features extremely fine 256-microstep resolution [35]. What makes it special is the StealthChop2 technology, which allows for near-silent operation - an excellent feature for household or kitchen environments [36]. Additionally, its UART interface lets users fine-tune motor current, enables quiet modes, and even detects stalls without external sensors [36].
- For Sit & Chow project, the TMC2209 is the ideal choice for quiet performance and reliability. Its silent motion profile and built-in stall detection make it perfect for a “kitchen-friendly” device that can also sense jams automatically. However, if cost or simplicity is a priority, the DRV8825 remains a strong fallback option due to its straightforward setup and proven performance.

3.1.7 Speaker

Designing an automatic dog feeder requires both mechanical accuracy and user-friendly features like sound feedback. The speaker provides audible confirmation of feeding, so choosing the right one is crucial. An unsuitable speaker can cause poor sound or even irritate pets. In low-voltage systems (5 V or 3.3 V), it’s important to balance loudness, efficiency, and electrical compatibility. Therefore, we need identify the best speaker specifications to deliver clear, gentle sound within those limits.

A speaker converts electrical energy into sound, and its performance depends on three main factors: impedance, power rating, and type. Impedance (in ohms) measures resistance to the amplifier—lower values draw more current and sound louder but can overload the amp. Power rating (in watts) shows how much power the speaker can handle without distortion or damage. The type, such as full-range, defines its frequency coverage; full-range speakers are ideal for compact systems like feeders.

Researching the right speaker is crucial because mismatched specs can cause poor efficiency, distortion, or failure. The goal is to find a balance where the amplifier and speaker complement each other, producing clear, gentle sound. As noted by Dynaudio and NLFxPro, proper matching ensures stable performance and comfortable audio levels - important for reliable feed alerts that don’t startle pets [37, 38] .

For Sit & Chow project, there are three options were evaluated:

Table 3.1.7: Comparing types of Speakers

Feature	1W 8Ω Speaker	5W 8Ω Speaker	3W 4Ω Full-range Speaker
Speaker Type	A standard speaker, possibly optimized for a specific, narrower range of frequencies. The sound could be less balanced than the full-range option	A standard speaker, potentially with a narrower frequency range.	A single driver designed to produce a wide range of frequencies, from low to high. This is ideal for voice playback and alert tones
Impedance (Resistance)	Higher impedance (8Ω) draw less current, which may result in a quieter output compared to a lower impedance speaker on the same amplifier.	Higher impedance (8Ω) is less efficient for power transfer compared to 4Ω option when driven by the same amplifier	Lower impedance (4Ω) draws more current from the amplifier, enabling more efficient power delivery to the speaker
Wattage (Power Handling)	Lower wattage (1W) might limit maximum volume and clarity, potential leading to distortion at higher volumes if the amplifier output is overdriven.	While capable of handling more power (5W), it might not offer a significant volume increase in a low-power, embedded application unless driven by a higher-wattage amplifier.	The 3W power handling is sufficient for producing clear audio and alert sounds balance of power and efficiency for this application
Loudness (Sensitivity)	Less efficient and likely quieter than the 4Ω option for the same input voltage	Less efficient than the 4Ω option. Its potential for higher volume can only be realized if the amplifier is also more powerful	A 4 Ω speaker can be louder than 8 Ω speaker with the same input power because it is more efficient at converting power to sound. A full range design also helps project the sound more effectively

Pros	Low power rating means very modest current or drive required; 8Ω is a common safe impedance for small amplifiers (the microcontroller audio driver likely can handle it); Lower cost, smaller size	Higher power handling given more headroom for louder notifications, which may help in presence of ambient noise; Still 8Ω, which is compatible with many drivers	4 Ω impedance draw more current at a given voltage, meaning more available acoustic output for the same voltage – useful in low-voltage system; A full-range speaker covers the tone/frequency band of interest (the dispense chime) without needing separate tweeter/woofer; 3W rating is the modest but sufficient for a small feeder environment
Cons	Only 1W power capability may limit maximum volume or dynamic headroom (In a noisy environment like kitchen, it may not be sufficiently audible); With 8 Ω load, for a given voltage, we get less current and less available power than lower impedance (volume may suffer depending on efficiency)	Higher power rating may encourage driving at higher levels than necessary, increasing distortion or draining more battery/current; If the enclosure and acoustic coupling are poor, we may not realize the benefit of the extra wattage; Wasted cost/size	4 Ω load means the driver/amp must supply higher current; if the amplifier is not sized for it, it may be risked overheating or distortion; If the speaker efficiency is low, even 3W may not result in adequate SPL in some spaces; must check sensitivity; Selecting a full-range means we must ensure the acoustic enclosure supports the frequency band (not too low, not too high)

After doing a research for the technical parameters and comparing, we have created a table 3.1.7 [39, 40, 41, 42, 43, 44]. We decided 3W 4Ω Speaker is a best balance for what we need: moderate power (3W) suitable for embedded drive, lower impedance (4Ω) to use voltage efficiently, and full range for nice tone. The other options either under-power (1W) or over-spec (5W) potentially wasting cost and driving complexity.

The speaker model we use: Adafruit Speaker - 3" Diameter - 4 Ohm 3 Watt [45]

3.1.8 Amplifier

An electronic amplifier is a device that increases the power, current, or voltage of an input signal to produce a larger output signal [46]. In the context of a speaker, the amplifier takes a low-power electrical audio signal from a source, like a microcontroller, and boosts its amplitude sufficiently to drive the speaker cone and produce audible sound.

In an automatic dog feeder, the amplifier powers the speaker responsible for producing alerts, voice prompts, or short melodies. Although the system does not require high-fidelity sound, matching the amplifier to the speaker is essential for clear and reliable audio. An underpowered amplifier can cause distortion, clipping, or low volume, while an oversized one may create excess heat, waste energy, or complicate the design. Therefore, selecting the right amplifier is a key engineering decision that balances performance, cost, and efficiency. It directly affects the sound quality, loudness, and durability of the system. Too little power can distort and even damage the speaker, while too much power can exceed its handling capacity with similar risks. We need to be careful research to ensure the amplifier is properly matched to both the speaker's electrical specifications and the microcontroller's output, resulting in a dependable and efficient audio subsystem for the feeder.

There are some of amplifier classes we have to compare to see which is the best option for Sit & Chow.

Table 3.1.8.a: Types of Amplifiers

Features	Class A	Class AB	Class D
Definition	The output device(s) conduct for the full cycle of the waveform (360° conduction angle).	A hybrid between A and B: the output devices conduct more than half the cycle (>180° but <360°). In idle they operate somewhat like Class A, but under higher output the design behaves like Class B.	A "switching amplifier" where the output devices switch fully on/off rapidly (e.g., via pulse-width modulation) rather than operating in the linear region. The output is filtered to remove the high-frequency switching to yield the audio signal.
Efficiency	~25% (very low)	~50-70% (moderate)	>90% (very high)

Heat Dissipation	High, requires a large heat sink	Moderate	Low, minimal or no heat sink needed
Complexity	Simple design	Moderate, requires a "push-pull" arrangement	Higher complexity, uses Pulse-Width Modulation (PWM)
Fidelity	Very high, low distortion	High, with minimal crossover distortion	High, modern designs offer excellent audio quality
Advantage	Excellent linearity and low distortion (very little crossover distortion) because the device is always on and in its linear region; Good for high-fidelity audio (audiophile territory). Disadvantages:	Good compromise: lower distortion than Class B, better efficiency than Class A; Widely used in home audio stereo amplifiers.	Very high efficiency (often >90%) because the devices are either fully on or off (hence minimal wasted power as heat); Compact size, lower heat sink requirement, good for embedded systems (battery or low-power).
Disadvantage	Very inefficient (typical theoretical efficiency of ~25% or less) so lots of heat dissipation; Lower power output for a given size and premium cost. Not usually ideal for compact, low-cost embedded systems unless the budget and heat sinking permit.	Heat and size may still be non-trivial (especially for high power). Efficiency may average ~30-55% depending on design; Some crossover distortion may remain if biasing is not optimal.	Historically some designs introduced switching noise, higher EMI, or were less linear than Class AB (although modern designs mitigate much of this); Some speakers may present complex loads which may challenge the output filter and lead to unexpected behaviour; Some audiophiles feel Class D “sounds different” though this is less relevant for embedded alerts.
Best For	High-end audio where fidelity is the only concern.	Most home audio, balancing efficiency and fidelity.	Portable, battery-powered, or space-constrained projects; high power efficiency.

Based on our research, we made the table 3.1.8.a [47, 48, 49, 50, 51, 52, 53]. When we made the table, we did not compare class B, class G and class H. Since class B is generally considered unsuitable for audio applications, and its characteristics are largely superseded by the Class AB design, we do not use it. In addition, classes G and H were not included in the comparative table because they are specialized, more complex variations of Class AB amplifiers and are less common in hobbyist-level projects like a dog feeder. They are mainly designed for high-power, high-fidelity applications, such as professional audio (PA) systems or high-end home theater receivers, where maximizing efficiency without switching noise is critical.

For a dog feeder, a Class D amplifier is the optimal choice. Its exceptional power efficiency minimizes heat generation and conserves battery life. The small size of Class D amplifier modules makes it easy to integrate into a compact project enclosure. While early Class D amps had high-frequency noise issues, modern designs offer excellent audio fidelity suitable for project notifications.

However, there are 2 types of Class D amplifiers we need to concern ourselves with:

Table 3.1.8.b: Comparing MAX98357A and PAM8403

Feature	MAX98357A	PAM8403
Input Type	I2S digital audio	Analog audio
Channels	1 (mono)	2 (stereo)
Efficiency (Class)	>90% (Class D)	>80% (Class D)
Noise & Quality	Very low noise, high fidelity, digital signal stays clean	Can be susceptible to noise on analog input
Power Output (5V)	Up to 3.2W into 4Ω	Up to 3W into 4Ω
MCU Compatibility	Requires I2S support (e.g., ESP32, Raspberry Pi)	Any MCU with analog output (PWM + filter) or a DAC
Ease of Use	Simple hardware wiring; more complex software for I2S stream	Simple wiring and software for analog signals

In table 3.1.8.b, some research has been done to see which fits better for our project [54, 55]. The MAX98357A offers superior audio quality and immunity to noise by accepting a digital audio stream directly, which is a major advantage for a reliable, multi-featured project like a pet feeder. While an analog amplifier like the PAM8403 is also highly efficient, its audio quality depends on a clean analog signal from the MCU, which can be susceptible to interference on a breadboard or custom PCB.

In conclusion, we picked Adafruit I2S 3W Class D Amplifier Breakout - MAX98357A [56]

3.1.9 LEDs

In the dog feeder design, we use LEDs for three main purposes: to show system status, provide user prompts, and assist with diagnostics. The status indicators let users quickly see if the feeder has power, if the Wi-Fi connection is active, or if there's a feeding error. For user interaction, the LEDs display cues like pairing mode or low-food alerts, helping make the feeder intuitive without needing to open an app. Lastly, they serve a practical role during testing and maintenance. Most commercial feeders use single-color LEDs for basic pairing and error notifications, and we found this approach simple yet effective for our setup.

From a design standpoint, having visible LED indicators prevents unseen failures. For example, if the motor jams or the Wi-Fi disconnects, we want an instant visual signal rather than relying only on an app notification that might go unnoticed. Studying LED types early in the design process also helps avoid costly redesigns later (like choosing the right package size or optical type can affect the layout of the plastic housing and PCB). Plus, using low-voltage LEDs keeps the design within consumer safety standards, meaning all visible parts stay electrically safe to touch.

There are some advantages and disadvantage when using LEDs

- Advantages:
    - High Efficiency: LEDs convert a larger percentage of electrical energy into light and generate significantly less waste heat compared to incandescent bulbs [57].
    - Long Lifespan: A typical LED can last for tens of thousands of hours, far outlasting traditional bulbs and reducing maintenance [58].
    - Durability: LEDs are solid-state components with no fragile filaments or glass enclosures, making them highly resistant to vibration and impact [59].
    - Compact Size: The small size of SMD LEDs allows for high-density placement on a PCB, crucial for compact and modern designs.
- Disadvantages:
    - Initial Cost: While prices have dropped significantly, LEDs can have a higher initial component cost than other lighting options [59].
    - Voltage Sensitivity: LEDs are sensitive to voltage fluctuations and require a stable power supply or current-limiting resistors to prevent damage [59].
    - Directional Light: Standard LEDs emit light directionally, which may require diffusers to achieve a wider viewing angle.

For the project, the choice is primarily between Surface Mount (SMD) and Through-Hole (THT) LEDs:

Table 3.1.9: Types of LED

Feature	Surface Mount (SMD) LED	Through-Hole (THT) LED

Size	Very small and compact	Larger, with long leads for mounting.
Mounting	Soldered directly onto the surface of the PCB.	Leads inserted into drilled holes and soldered from the back.
Automation	Highly suitable for automated assembly, lowering mass production costs.	Mostly manual or specialized automated insertion; slower for mass production.
Durability	Good vibration resistance, as the solder joint is flush with the board.	Better mechanical stability due to leads passing through the board, ideal for components under stress.
Thermal Management	Heat is more easily dissipated through the PCB, aiding lifespan.	Less efficient heat dissipation due to limited contact area.
Cost	Generally lower unit cost, especially for high volumes	Lower cost for small-scale prototyping or where high mechanical strength is needed.

After researching information, we have a comparative table 3.1.9 [60]. For the dog feeder, Surface Mount (SMD) LEDs are the recommended choice. They offer a compact design, are perfect for automated manufacturing, and are highly efficient. Given the low mechanical stress on the indicator LEDs, the superior mounting strength of THT is not necessary. The reduced size of SMD LEDs frees up valuable board space for other components

Also, when comparing LED options, package size is the first factor we considered. We went with the 0603 package because it keeps the board compact and fits neatly near the Wi-Fi module and motor driver. The SML-512 series is a 0603 LED with a diffused lens that provides a wide viewing angle, making it easy to see from different positions. Larger sizes like 0805 or 1206 are brighter and easier to hand-solder but take up more space, which could clutter the front panel.

Color choice also matters. Green is ideal for showing that everything's normal since it's easy for the human eye to detect and still visible through slightly tinted plastic covers.

Overall, through consideration, we found that compact, low-power, diffused green LEDs strike the best balance for a reliable, visible, and durable indication system in an automatic pet feeder.

Our decision for the LEDs using in Sit & Chow is: SML-512PWT86A [61]

3.1.10 Button

Although our project had planned for the feeding time to be pre-programmed to schedule feedings and eject food, we still wanted to have a backup. This would be a push button in case the machine had a software error and does not dispense food on time. This would allow the pet or owner to press a button to still deliver food. Instead of the user having to open the machine and retrieve food from the food storage, the push button would be a more convenient function as it could be delivered straight from the storage to the food tray.

The push button is a basic component of the user interface for an automatic dog feeder. It provides a means for the user to easily interact with the device to perform the action of manually feeding the dog.

When considering the choice of a push button, we also considered factors such as durability, seamless integration with the project's electronics.

There are types of physical buttons we consider:

Table 3.1.10: Types of Button

Feature	Tactile Switch	Metal Push Button
Use for	PCB mounted	Panel mounted
Durability	Moderate. Can be damaged by heavy or repeated pressure.	Very high. Designed to withstand heavy use and resist vandalism
Tactile Feedback	Clear, audible "click."	Distinct, high-quality "push" feel.
Aesthetics	Utilitarian. Not typically visible to the end user.	Premium, industrial appearance.
Mounting	Requires soldering to a specific footprint on the PCB.	Requires a circular cutout in the enclosure and a fastening nut.
Cost	Very low.	Moderate to high, depending on material and features.
User Experience	Best for internal testing or user interfaces where the button is covered.	Provides a robust, confident interaction for the end user.

Table 3.1.10 will be easier to compare [62, 63, 64]. For this smart dog feeder project, I decided to use the tactile switch as the main control interface. I chose this specific switch because it offers a great balance between practicality, cost, and performance.

One of the biggest reasons for choosing this switch is its seamless integration into the system. It can be mounted directly onto the feeder's main PCB and placed beneath a flexible silicone or plastic overlay. This setup keeps the entire enclosure sealed, protecting the electronics without needing extra panel holes or exposed mechanical parts. It also gives the design a cleaner and more professional look, which is important for products that will be used regularly in households. Another major advantage is cost and simplicity. The tactile switch is inexpensive and easy to solder, which helps reduce both production costs and assembly time. The minimal parts and easy integration also make troubleshooting and replacements simpler if needed. Finally, the switch provides adequate performance for this type of application. While it's not as heavy-duty as a metal push button, it still delivers a satisfying tactile click and offers a long operational life of over 100,000 cycles. When paired with a protective overlay, it gains reasonable water resistance, making it perfectly suitable for indoor environments where the feeder won't be directly exposed to moisture. Overall, the tactile switch strikes an ideal balance between durability, usability, and design efficiency for this smart dog feeder project.

For the buttton, we decide to use: TS02-66-70-BK-160-LCR-D [65].

3.1.11 Sensing Storage Capacity

Food storage monitoring is an essential component in automated feeding systems, as it ensures that sufficient food is available for scheduled feeding events. In Sit & Chow project, accurately determine the remaining quantity of food in the hopper allows the system to notify users when refilling is required, preventing feeding delays or malfunctions. Without such measurements, the system can run empty during a scheduled feeding, or waste energy by overfilling. By checking storage levels, they dog feeders can trigger refill alerts, adapt feeding schedules, or avoid dispensing when supply is insufficient.

A sensor is a transducer that detects changes in a physical property and converts them into a measurable electrical signal. In an automatic dog feeder, the sensor's role is to determine how much food remains inside the hopper or storage bin. There are some ways to approach the distance are: detect weight, detect distance, or dielectric change.

Table 3.1.11: Types of Sensors

Feature	Load cell Sensor	Infrared Sensor (IR)	Capacitive Sensor	Ultrasonic Sensor

Definition	A load cell measures the mass of fool by converting mechanical strain into voltage changes using a strain bridge circuit. When food is added, the metal base slightly deforms, and the resistance change is converted into an electrical output.	An IR sensor uses emitted and reflected light to detect proximity. It sends out infrared light and measures how much is reflected back from the food surface.	A capacitive sensor detects changes in dielectric constant between air and the food materials. As food fills the bin, capacitance increases.	The ultrasonic sensor operates by emitting a high-frequency sound wave and measuring the time of flight of the echo reflected from the food surface
Accuracy	Very high, ideal for precision weight measurements. Must be properly calibrated for the specific load	Fair for simple presence detection, but can be unreliable with different food colors, textures, or ambient light.	Good for level sensing in a controlled environment, but can be affected by food density and moisture.	Good for distance measurement and non-contact level sensing
Environment Factors	Mostly immune, especially with hermetic sealing. Sensitive to vibrations if measuring minor	Prone to interference from dust, direct sunlight, and changing ambient light.	Sensitive to temperature, humidity and material build-up on the sensor.	Can be affected by extreme temperature changes, but generally reliable in typical home environments.
Installation	Require mechanical integration under the food hopper. Mounting is critical to ensure accurate, long-term performance.	Must be placed inside the hopper, with a clear line of sight, often at a fixed point to detect a specific level	Probes must be mounted to the interior wall of the hopper or as a strip running down the side	Can be mounted at the top of the food hopper, firing down to the food surface

Pros	Measures actual weight for precise portion control; Can be very accurate and reliable for batching.	Inexpensive and easy to implement.	Low cost, durable, and suitable for non-conductive materials.	Non-contact method is hygienic and not affected by food type or color; Affordable, widely available, and simple to interface with microcontrollers.
Cons	More complex mechanical design and calibration required; Increased cost for high-precision models.	Unreliable for continuous level measurement; False reading from ambient light or dusty conditions	Requires calibration for the specific food type; Performance degrades with material build-up on the sensor.	Not as accurate as a load cell for weight, and has a minimum detection range

After doing the research and comparing all the types of sensors, we made a table to see which sensor would be fitted for our project.

Each sensor technology has unique benefits, but the environment of a pet feeder imposes specific constraints: dust, non-uniform particle shapes, and variable lighting. Load cells are mechanically accurate but prone to drift if the feeder is moved. IR sensors are too dependent on surface reflectivity. Capacitive sensors can be unstable under varying humidity levels. In contrast, ultrasonic sensors perform consistently across these conditions because they rely on sound wave reflections rather than optical or physical contact.

For checking how much food is in the storage, we use: Ultrasonic Distance Sensor - 3V or 5V - HC-SR04 compatible - RCWL-1601 [66].

3.1.12 Depth Module

Before detecting if the dog is sitting, it is important to determine that the dog is close enough to the feeder. If the dog is not close enough to the feeder, the motor may trigger some type of false positive, such as the dog sitting too far away. It also helps the detection model perform better by being able to distinguish the features of the dog more clearly to determine its output.

Ultrasonic Sensor

The ultrasonic sensor determines distance by emitting high-frequency sound waves and measuring the time it takes for it to return. The travel time is converted to distance and provide a detection range from 2cm to 4m with about a ±1cm accuracy under optimal conditions [67, 68]. For Sit & Chow, an ultrasonic sensor can detect if the dog is within a specified range before the feeding mechanism is triggered. It is low cost and GPIO-based that can pair well with the ESP32-S3 microcontroller. However, different fur can effect the sensor output, potentially producing false readings [67, 68].

https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf [a]

https://www.elecfreaks.com/blog/post/hc-sr04-ultrasonic-module-user-guide.html [b]

Infrared

Infrared (IR) proximity sensors operate on the principle of triangulation. The sensor emits an IR beam, and a position-sensitive detector measures the angle of the reflected light to estimate the distance [69, 70]. These sensors consume little power and use analog inputs for the MCU. The effectiveness of the sensor depends on how reflective the dog's fur coat is. Therefore, darker coats can be much harder to detect because the dark fur absorbs more IR light.

https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf [a]

https://www.pololu.com/product/136 [b]

Time-of-Flight

Time-of-Flight (ToF) sensors operate by emitting a short pulse of infrared laser light and precisely timing its reflection. Unlike IR triangulation, ToF sensors directly measure the IR light's round trip time between varying ranges, such as up to 400cm or 2m [71, 72]. However, since it uses IR, it may struggle with dark fur as well. They are compact, low-powered, and use an I^2C interface that is compatible with the ESP32-S3. It is higher in cost compared to the other sensors, but the sensor is more accurate as it can work on dark fur coats.

https://www.st.com/resource/en/datasheet/vl53l0x.pdf [a]

https://www.st.com/resource/en/datasheet/vl53l1x.pdf [b]

mmWave

Millimeter-wave (mmWave) sensors operate by using Frequency-Modulated Coninuous Wave (FMCW) radar to detect objects and measure distance by analyzing the Doppler shift of reflected electromagnetic waves [73, 74]. They can function in any lighting condition and are unaffected by fur color and communicate through UART or SPI [73, 74]. It can potentially be sensitive enough to detect breathing motion.

https://www.infineon.com/part/BGT60LTR11AIP [a]

https://www.infineon.com/assets/row/public/documents/24/49/infineon-bgt60ltr11saip-datasheet-en.pdf?fileId=8ac78c8c7c9758f2017cb7b72ee5268e [b]

Table 3.1.2.1: Types of Depth Sensors

Sensor	Advantages	Disadvantages
Ultrasonic Sensor	Cheap; Simple to integrate with GPIO	Sensitive to noise reflections like the bowl or fur
Infrared Sensor (IR)	Compact and low power; Good for short range detection	Sensitive to ambient light; Dark fur color can reflect poorly
Time-of-Flight Sensor (ToF)	Works in dark or light environments; Easy to integrate with I^2C connection	Probes must be mounted to the interior wall of the hopper or as a strip running down the side
mmWave	Unaffected by light and fur color; Cand detect subtle motions	High power; High cost; More complex data parsing

After evaluating multiple depth sensors, the ToF sensor was chosen as the best sensor for Sit & Chow. The measurement principle results in high accuracy, repeatability, and immunity to lighting variations, make it ideal when detecting when a dog is within range to the feeder. It easily integrates with the microcontroller and is compact in size, so it does not impact the placement of other components like the camera. It is a little more expensive than the ultrasonic or IR sensor, but cheaper than the mmWave.

Table 3.1.2.2: ToF Sensor Comparison

Category	VL6180X	VL53L0X	VL53L5CX	TMF8701
Range	0 – 200mm	20 – 2000mm	0.05 – 4m	2cm – 1m
Accuracy	±5mm	±25mm	±15mm	±20mm
Features	Great for short distances; Too short for feeder	Slightly reduced accuracy beyond 1m; Stable up to 2m; Low power	Higher cost and power; Strong ambient light immunity	Extremely small; Low power; Less accurate with dark fur
Cost	$7 – 9	$7 – 10	$25 – 30	$10 – 13

The TMF8701 ToF sensor was chosen as the most optimal depth detector for the Sit & Chow. The TMF8701 in particular emits short burst of IR light and calculates the time-of-flight of returned photons, achieving a detection range of approximately 2cm to 1m. Its I^2C interface provides seamless connection with the microcontroller. This minimizes the wire complexity and overhead with low power. In conclusion, the TMF8701 was chosen for its balance in accuracy, implementation, cost, and size to determine if the dog is close enough while sitting before triggering the motor.

3.2 Power Management

3.2.2 Power Delivery Topology

A well-structured power delivery system is essential for maintaining stable and noise-free operation in an automated dog feeder. A typical setup begins with a 12 V DC input, which directly powers the motors (especially stepper motors) through their drivers with current limiting. From this main line, the voltage is stepped down using buck converters to provide 5 V for logic circuits and sensors, and then further regulated using low-dropout regulators (LDOs) or another buck converter to 3.3 V for components like the MCU or Time-of-Flight (ToF) sensors. To ensure accuracy for sensitive modules such as the HX711 load cell amplifier, it's best to include generous input capacitance and use a star-grounding layout, which minimizes electrical noise and interference.

Adapters (Wall Supplies)

For reliability and safety, a regulated 12 V wall adapter should be rated for at least 3 A. This provides sufficient overhead for handling motor startup currents and ensures stable voltage under load. Common "wall wart" or external power supply (EPS) adapters are compact and easy to integrate. When selecting an adapter, focus on matching the required voltage, ensure adequate continuous current capacity, and confirm connector compatibility [75, 76]. Modern USB-C Power Delivery (PD) adapters have also become widely available and can deliver up to 240 W, making them an attractive, flexible option [75]. Using a USB-C PD input module allows the system to work seamlessly with universal adapters and modern power sources.

Power Jack

For physical connection, the standard in most DIY and maker projects is the 5.5 mm outer diameter × 2.1 mm inner diameter center-positive barrel jack. A reliable model such as the CUI PJ-102A offers a 2.5 A current rating and supports up to 24 V, making it suitable for most 12 V systems [77, 78]. Always mark the polarity clearly - center-positive is the standard convention across hobby and embedded electronics ecosystems. Using a sturdy through-hole jack design ensures mechanical stability, especially when the device will be plugged and unplugged frequently.

Power banks

For primary power, a wall outlet remains the best choice, offering unlimited runtime and minimal complexity. However, alternative energy sources can be useful for backup or portable versions:

- Li-ion batteries (1–3S configurations) provide high energy density and are well-suited for backup power, though they require a battery management system (BMS) and proper charging circuits for safety.
- NiMH or Alkaline batteries are simpler but offer lower energy density. NiMH cells suffer from higher self-discharge, and Alkaline batteries tend to sag under motor loads, limiting their usefulness to low-power sensors or real-time clocks.
- Lead-acid batteries, while durable and inexpensive, are heavy and less suitable for compact or kitchen-based feeders, though they can serve as stationary backups.

Table 3.2.2: Comparision of Power Sources

Power Source	Voltage	Advantages	Disadvantages
12V Wall Adatper	12V at 3A	Simple and reliable; Provides sufficient current overhead for motors; Safe, regulated output	Requires access to wall outlet; Must match connector polarity and sizing
USB-C Power Delivery Adapter	5-20V (selectable), up to 240W	Universal and widely available; Compact and standardized; Allows using laptop / phone chargers	Requires PD negotiation module; Module adds cost/complexity
Li-Ion Battery	3.7 - 11.1V	High energy density; Rechargeable and compact	Requires charging and protection circuitry (BMS); Safety considerations
NiMH Cells	1.2V per cell	Simple and safe; Easy to replace	Small auxiliary loads only
Alkaline Cells	1.5V per cell	Very common and inexpensive	Only suitable for ultra-low-power modules
Lead-Acid Battery	6 - 12V	Durable, tolerant of load spikes	Stationary emergency backup, not recommended for compact feeder designs

3.2.3 Voltage Regulation

Voltage regulation means converting a higher/variable input into stable, low-noise rails.

- Buck converters to step 12V down efficiently to 5V. Classic Buck to 5 V.
    - Older non-synchronous parts such as TI's LM2596 (150 kHz, 3 A, up to 40 V in) are proven and ubiquitous; modules abound, though diode-rectified topologies waste a bit more heat [79]. The non-synchronous buck converter, such as the LM2596 by Texas Instruments, is known for its ruggedness and simplicity. It's one of the most forgiving regulators to work with and is widely available in modular form, making it a good choice for quick prototyping or DIY applications. Its main drawbacks come from using larger magnetic components and a diode-based rectifier, which leads to higher power losses and heat generation - especially under light-load conditions [79]. Still, if the design has adequate space and airflow, this type of regulator delivers reliable performance with minimal setup effort.
    - Newer synchronous bucks (e.g., TPS54202, 2 A, up to 28 V in) integrate both FETs for better light-load efficiency and smaller footprints - useful in a compact feeder enclosure [80]. The synchronous buck converter, represented by models like TPS54202 or MP1584, offers superior efficiency across a wide range of loads. These designs integrate internal MOSFETs and feature soft-start capability, resulting in compact, efficient, and smooth operation. However, because of their faster switching edges, they can be more prone to electromagnetic interference (EMI) if the PCB layout isn't handled carefully. These converters are especially suited for "kitchen-quiet" feeder designs where thermal performance, compactness, and low noise are priorities.
- Buck-boost converters when the input may wander above/below the target. For example, if we power from a battery or want universal USB-C PD operation where profiles change, a single-inductor buck-boost (e.g., TPS63060) holds a 5 V rail steady across 2.5–12 V inputs with ~93% efficiency peak—great for "always-on" logic that must ride through dips and surges [81]. The buck-boost converter, such as the TPS63060, provides flexibility by maintaining a steady 5 V output even when the input voltage dips below or rises above the desired level. [81]. This makes it ideal for systems powered by batteries or variable supplies, where voltage sags or brownouts may occur during motor activity. While it's slightly more complex and comes at a higher cost than a standard buck, its ability to sustain operation through fluctuations makes it invaluable for feeders that must complete a dispensing cycle without interruption.

- LDOs for final “polish” on noise- sensitive rails, sometimes trading efficiency for simplicity and low output noise. A quiet LDO like LT1763 is a strong last mile for the HX711 and MCU rail. Keep the dropout and thermal budget in mind if feeding it from 5 V [82]. The low-dropout (LDO) regulator, like Analog Devices' LT1763, shines when ultra-low noise is critical [82]. It’s incredibly easy to integrate and deliver a clean, stable output—perfect for sensitive analog circuits such as ADCs or sensors [82]. The trade-off lies in efficiency: the power loss is proportional to the voltage drop across the regulator multiplied by the current load. Adequate thermal management is therefore important. Despite this, an LDO remains the best choice where signal integrity and measurement accuracy are more important than saving every milliamp of power.

3.3 Software

3.3.1 Mobile Application Stack

3.3.1.1 Backend

Choosing the backend for this project is very important as it dictates where all the data will be stored. We are primarily looking for a backend that is easy to work with, is cheap, is reliable, and is efficient. The most important aspect of the backend is that it should be easy to work with as we do not have much experience when it comes to working with a mobile application. We will also need to host a machine learning algorithm on the backend, by taking in the video feed from the camera and then feeding it to the algorithm.

AWS Lightsail

Our first consideration for the backend of our mobile app is AWS Lightsail. AWS Lightsail is a simplified cloud service from AWS specifically designed for easy deployment of virtual private servers and other resources [83]. There are many advantages to using this product for our backend. Firstly, it has a fairly simple interface, which makes it easy to launch and manage servers. There is also full control and flexibility over the backends’ environment and frameworks. The performance of the AWS servers is predictable and easy to track, helping us understand how powerful the backend can be. It also has good security with built-in SSL, firewalls, and AWS IAM integration [84]. Furthermore, it has good media handling, integrating easily with AWS S3 for image/video storage.

Unfortunately, there are several disadvantages which make this consideration not as ideal as other candidates. First, the API has to be setup manually for mobile and ESP32 integration. While this is usually a common task, some backends setup the API automatically, reducing the workload substantially. AWS Lightsail also has higher maintenance costs, making us responsible for OS updates, server patches, and backend monitoring. Additionally, scaling is not automated and requires manual configuration. While this is not a big issue as we are currently operating on a very small scale, it is something to consider. There is also no free tier with AWS Lightsail. This is not ideal as other backends bring comparable if not better performance while also giving a generous free tier. This is a big part of why we wouldn't choose Lightsail. Also, authentication setup is manual and can be more complex to setup than other backends. To stream, there must be extra setup and configuration of the backend. Lastly, there is a steeper learning curve for those who haven't used AWS before. We have not used AWS, so this impacts us greatly.

Overall, AWS Lightsail gives us a large amount of flexibility, control and scalability for backend development, but needs more setup, configuration, and maintenance. This is good for projects which want to expand to many devices, have custom APIs, or need low-level control. However, since we want a fast and simple integration, other considerations fit better.

https://hackernoon.com/a-beginners-guide-to-aws-lightsail-pros-cons-and-use-cases-hrq32d2 [a]

https://cloudchipr.com/blog/aws-lightsail [b]

Supabase

Our second consideration will be Supabase. Supabase is an open-source backend-as-a-service platform that acts as a database-first development platform. There are several advantages that come with using Supabase as our backend. First of all, it has quick integration with Flutter, REST, and JS SDKs, requiring minimal setup. We will be using Flutter as our front end, so this is a great advantage to have. Supabase is built on PostgreSQL and has real-time sync and has the flexibility of SQL [85]. When it comes to authentication and security, Supabase has built-in JWT/OAuth support, making it easy to manage multiple devices and users securely [85]. Supabase also has simple file storage for images and clips with Flutter apps. Furthermore, since Supabase uses PostgreSQL, it can automatically scale horizontally for APIs and authentication. It also has a fully managed backend. An important advantage to consider with Supabase is that it has a generous free tier, including database, authentication, and storage. If that is not enough, it also offers a predictable pay-as-you-grow model. Supabase is also open source, with full SQL and API access and easily integrates with edge functions for backend logic.

After considering the advantages, we move on to the disadvantages of using Supabase as the backend. Firstly, if we wanted to integrate Supabase with the ESP32, we would need a REST API or MQTT bridge as there is no native SDK. The real-time performance of Supabase can lag under high write frequency, and if we wanted to scale the database beyond the free tier, we might need to manually tune it. Furthermore, if we had complex security rules, it would further increase the complexity of fine-tuning database policies. Another disadvantage of using Supabase is that it is not ideal for continuous video streaming, and the file size and bandwidth are limited on the free tier. While our project needs to handle video streaming, it will be handled by something like WebRTC, which is better suited for handling real-time communication for mobile applications. Additionally, if we wanted to increase our scale, we would need to upgrade out of the free tier and even then, it is not as globally distributed as other backend products. Lastly, Supabase is not as popular or widespread as other backends and as such has fewer official extensions.

Overall, Supabase is fairly easy to use and has decent power and flexibility, making it ideal for structured data control, SQL querying, and self-hosting. It would be good if we wanted an open-source, real-time backend with a reliable database and a moderate learning curve. Its downfall is that it is not good for continuous video streaming and is not cheaply scalable. Therefore, we will be moving on from Supabase as the backend.

https://supabase.com/docs [c]

Firebase

Our final consideration and our choice for the backend is Firebase. Firebase is a platform provided by Google that provides tools for building and growing mobile and web applications without having to manage backend infrastructure. It offers many advantages that we will now consider. First is the ease of setup. It has an extremely fast setup bringing native Flutter SDKs and requiring no server management [86]. It also has a real-time database and, along with Firestore, can provide instant cloud sync. Additionally, its authentication is built-in and has many options such as email, username/password, Google, anonymous, and other authentication options. Using Firebase Storage allows users to handle images/videos securely and easily. Furthermore, scaling is completely automatic with its Google Cloud infrastructure and has strong uptime [87]. Firebase also has a very generous free tier and can be upgraded to a pay-as-you-go plan that scales with usage. It also has seamless integration with cloud functions, analytics, and a machine learning kit.

While there are many advantages, Firebase does have some disadvantages to consider. First, it has limited flexibility for custom protocols if we want to use them. This may be a hindrance as we might need custom protocols for the ESP32, but it is something we are willing to accept considering the great advantages Firebase has. Firebase also has high-frequency updates which may increase the cost and latency of the project. But, since our project is on such a small scale, it is unlikely to affect us. Firebase is also not ideal for live streaming, but rather better for snapshots. However, since we will be handling the live stream with another product, this is a non-issue. The major downside to using Firebase is that since the backend is handled for us, there is limited customization available to us. While we do need to have the machine learning algorithm on the mobile app, it does not need to be on the backend and can instead be something hosted on the mobile app itself.

Overall, Firebase is our choice as the backend for our project because it provides real-time synchronization, secure user authentication, cloud storage, and easy integration with Flutter. It has very low maintenance due to its automatically scaling database, allowing fast development. Since we will likely not need to customize the backend and it efficiently manages data, using Firebase is a great choice for our project.

https://firebase.google.com/docs [d]

https://cloud.google.com/firebase [e]

Figure 3.3.1.1: Backend Hosts

Category	Firebase	Supabase	AWS Lightsail
Ease of Setup	Easiest setup; no servers to manage; seamless with Flutter and mobile SDKs.	Simple setup with SQL-like structure; integrates quickly with Flutter via REST or SDKs.	Requires manual API and server setup; more configuration needed for databases and hosting.
Real-Time Communication	Built-in real-time sync with Firestore and Realtime DB.	Supports real-time updates using PostgreSQL's replication features.	Must implement custom WebSockets or MQTT for real-time data.
Authentication & Security	Built-in user auth (Google, email, custom tokens) and rule-based security.	JWT/OAuth auth with role-based access via SQL policies.	Uses AWS Cognito or custom JWT; full control over IAM and firewalls.
Database Model	NoSQL (Firestore) and JSON structure; ideal for live sync.	SQL (PostgreSQL) with relational querying and constraints.	Choose any (MySQL/PostgreSQL/MongoDB) — manual setup required.

Storage & Media Handling	Firebase Storage handles snapshots and logs easily.	Supabase Storage for files and media, similar to Firebase.	S3 integration for media; supports HLS streaming and large file handling.
Scalability	Auto-scales seamlessly with traffic; fully managed.	Scales with PostgreSQL cluster upgrades; moderate management.	Manual scaling by upgrading instances or adding load balancers.
Cost	Free tier + pay-as-you-go; cost increases with high read/write frequency.	Generous free tier; predictable pricing.	Fixed monthly per instance; predictable but may cost more idle time.
	Minimal maintenance (serverless).	Low maintenance; open source with optional self-hosting.	Higher maintenance—updates, security patches, backups managed manually.

3.3.1.2 Frontend

Since we intend to make this application a mobile app, we need to choose our frontend assuming so. We specifically want to host our mobile app on iOS devices mostly because we are familiar with these devices. We specifically want to work with a frontend that is easy to learn. If it is easy to learn and can host our mobile app sufficiently, we will be satisfied with such a technology. We will consider three different frameworks for our frontend: React Native, Ionic, and Flutter.

React Native

React Native is an open-source UI software framework developed by Meta Platforms used to create applications for Android, iOS, macOS, Windows, and other operating systems [88]. It allows the developer to use the pre-existing React framework along with native platform capabilities. It is used by many companies, such as Facebook and Microsoft, to develop Android and iOS applications [88]. It also has easy integration with other technologies such as REST APIs, Firebase, and cloud backends. It is almost identical to React, so those with experience in React will likely find this easier to work with than other frameworks. However, none of our group members have worked with React before, so this, if anything, hinders us from learning it. There are also some platform specific issues that may require native code, increasing the complexity should we encounter them. This is a strong choice for a frontend for a mobile app, but due to the unfamiliarity to React, we will not be moving forward with this consideration.

https://en.wikipedia.org/wiki/React_Native [a]

Ionic

Ionic is a framework that is slightly different from others as it allows developers to build native-like apps for iOS, Android, and the web in the same codebase [89]. Using this framework means there would be no need to split the web and mobile applications into two different development environments. This is especially helpful for applications which want to be used across different operating systems in the same way. It uses HTML, CSS, and JavaScript as its main programming languages [89]. So those who have experience with those languages have a clear advantage in using this framework. However, due to its multi-platform accommodations, there must be careful testing on each of these platforms as the UIs behavior can differ. Once again, none of our group members have experience with these languages and there is no need for us to have a web application, so there is no sense in using Ionic as our framework.

https://ionicframework.com [b]

Flutter

Flutter is Google's official open-source framework designed to create native apps for iOS, Android, and the web in the same Dart codebase. Flutter is highly valued for its fast development, consistent UIs, and shared logic across platforms. It has near native performance through ahead of time compilation to native ARM code [90]. Its custom rendering engine, Skia, ensures a consistent UI across all platforms [90]. It also has a hot reload function that allows for rapid iteration during development. Flutter has seamless integration with Firebase and a growing plugin ecosystem. This makes it a strong choice for real-time updates and IoT data integration. The only possible downside is that its programming language, Dart, is not mainstream and is likely to have a learning curve for the developer. Fortunately, we do have experience with using flutter due to previous coursework. So, not only do we have familiarity with this technology, but it also provides seamless support for our backend and works perfectly across different operating systems. Therefore, we will be moving forward with Flutter as our front-end framework.

https://github.com/flutter/flutter [c]

Figure 3.3.1.2: Frontend Applications

Category	React Native	Ionic	Flutter
Language	JavaScript / TypeScript (React)	HTML, CSS, JavaScript / TypeScript (Angular, React, or Vue)	Dart
Rendering Approach	Uses native UI components bridged from JavaScript	Uses WebView to render UI elements (hybrid approach)	Uses Skia rendering engine for fully custom native rendering

Performance	High performance, close to native; slight overhead from JS bridge	Moderate performance due to WebView reliance; best for lightweight apps	Near-native performance via AOT-compiled Dart and direct rendering
UI Consistency	Matches native look using system components	Mimics native look using web components and CSS	Pixel-perfect, consistent UI across all platforms via custom widgets
Development Speed	Fast — shared codebase, hot reload, large ecosystem	Very fast — uses standard web technologies and tools	Fast — hot reload, unified widget tree, rich tooling
Learning Curve	Easy if familiar with React/JS	Easiest for web developers (HTML/CSS/JS skills reusable)	Moderate — requires learning Dart and widget-based UI paradigm
Ecosystem & Libraries	Mature, vast NPM ecosystem, backed by Meta	Large plugin base, strong web developer community	Growing rapidly, strong official support from Google
Integration with Backend (Firebase, REST APIs)	Excellent — native Firebase SDKs and REST API support	Good — uses JavaScript SDKs; works well with Firebase	Excellent — first-class Firebase integration and SDKs
Best Use Cases	Cross-platform apps needing near-native performance and native UI look	Lightweight apps, prototypes, or those reusing web code	High-performance, visually rich, and consistent cross-platform apps
Drawbacks	Occasional native bridge issues, dependency maintenance	Slower rendering for complex UIs, limited native feature access	Larger app size, Dart adoption required, steeper learning curve

3.3.2 Embedded Software

The embedded software controls the mechanical and sensory operations of Sit & Chow. The ESP32-S3 microcontroller runs firmware developed in C or C++ under the FreeRTOS environment. The system can be structured around a finite-state machine (FSM) or a real-time operating system (RTOS). The FSM approach is a rule-based sequence of operation, such as moving from “Idle” to “Alert” and “Wait for sit detection” to “Dispense.” It creates an explicit flow of control and simplifies debugging. Alternatively, using RTOS allows for multiple concurrent tasks. For example, the motor, and LED light are triggered simultaneously. C/C++ are the industry standard due to performance efficiency, low-level hardware control, and timing behavior.

The hardware components it controls:

- Camera capture for live video stream and sending frames to detection model.
- Speaker to alert when feeding time and allow owner to speak to dog.
- LED indicator when dog is confirmed sitting.
- Motor to dispense food.
- Wi-Fi module to transmit data to mobile application.

C

C was designed for system-level programming for microcontrollers and real-time devices. Its ability to interact directly with registers and drivers allows us to manage the timing needed for Sit & Chow’s mechanics. When the motor releases food, precise timing can determine is the feeder dispenses smoothly or mishandles due to jams or delays. C ensures deterministic execution, which is essential to real-time control loops. It is the most common language for embedded software and real-time control with low overhead [a].

https://www.mdpi.com/1999-5903/17/6/269

C++

C++ extends the advantages of C by offering abstraction through object oriented design and modularity. By implementing C++ with FreeRTOS, tasks such as camera capture, feeding control, audio signaling, LEDs, and wireless data transmission can run concurrently. A structured multitasking environment reduces blocking operations, allowing for multiple action to occur simultaneously, but has limited CPU speed and memory [a]. For example, dispensing the food and changing the LED light indicating the task was successful.

https://forums.freertos.org/t/combine-freertos-tasks-and-c-objects/6414/2

Python

Micropython is a potential python-base solution used for fast prototyping. The ESP32-S3 has limited memory, so code must be executed efficiently. Micropython requires more memory to execute since Python does not compile into machine code, and it does not allow for direct management of memory [a]. It is much slower in performance and execution for embedded tasks compared to C/C++ [b], meaning significant runtime overhead.

https://stackoverflow.com/questions/62506302/why-developers-use-c-c-for-embedded-systems-rather-than-high-level-language-li [a]

https://www.mdpi.com/2079-9292/12/1/143 [b]

Table 3.3.2: Comparision of Embedded Software

Category	Use Case	Advantages	Disadvantages
C/C++	Low-level embedded control for MCU	Direct access to hardware registers and peripherals; Runs efficiently with minimal memory; High performance and timing percision	Manual memory management; Limited abstraction; Longer development cycles
C/C++ with FreeRTOS	Run components concurrently	Real-time scheduling ensures deterministic behavior; Enables parallel task maanagment;	Requires careful task priority tuning; More complex
MicroPython	Python-based embedded control for MCU	Extensive built-in libraries for networking and sensors; Readable syntax	Much slower execution; Limited real-time control; Higher memory requirement
TinyML Frameworks	Compressed machine learning detection model for MCU	Low latency; Offline operation; Enhances data privacy	Limited to small CNNs; Low accuracy

In conclusion, C offers the best balance between performance and real-time responsiveness for Sit & Chow's embedded software. C enables deterministic control over the motor and sensors with low memory and consistent timing. It will ensure that when the detection confirms the dog is sitting, the servo motor and LED indicators respond immediately. With no Free RTOS, there is no delay due to context switching or thread management, thus, reducing computational costs. The firmware will be structured around a main control loop with interrupt service routines that handle real-time events. Since C allows for direct memory control, it makes efficient use of the ESP32-S3 limited RAM.

3.3.3 Detection Model

The detection model used in Sit & Chow is a computer vision-based deep learning algorithm designed to detect the dog's posture. Specifically, the model will determine if the dog is "sitting" before food is dispensed. The model follows a typical object detection and classification pipeline for a machine learning (ML) model:

- Image Input and Preprocessing

  The Sit & Chow camera captures a frame or image. The frame is resized and normalized to match the model's input dimensions, ensuring consistent scale and lighting for analysis. We use OpenCV, an open-source Python library for computer vision, image processing, and video analysis [91]. It handles the preprocessing, allows visual debugging, and enables real-time analysis, and serves as the bridge between the camera hardware and the ML model, preparing and sending the image for inference.

https://docs.opencv.org/4.x

- Feature Extraction

  After preprocessing, the image is passed into a deep-learning framework, such as PyTorch or Tensorflow. These frameworks are used to build, train, and run neural networks or ML algorithms [92, 93]. They support tensor math, automatic differentiation, GPU acceleration, and model deployment. The image passes through multiple convolutional layers that extract visual features, such as edges, texture, and contours. These help distinguish the dog from the background, and these patterns allow the model to generalize across different breeds, sizes, and colors.

  https://docs.pytorch.org/docs/stable

  https://www.tensorflow.org

- Object Detection and Localization

The model identifies a region in the image that has a dog by using a bounding box and confidence score [94]. This will be estimated by the detection model and drawn on the image with OpenCV. During the training process, the model will be given a very large set of annotated images of dogs in various positions, locations, and backgrounds to learn what a dog looks like and how to distinguish it from the environment. The model will predict the coordinate position of the bounding box. In OpenCV, (0,0) starts at the top left of the screen and the max resolution at the bottom right of the screen [95]. Therefore, if the image is 640x480, then the coordinate plane is from (0,0) to (639, 479). After the model has estimated the bounding box, it will be sent to the posture classification algorithm.

https://arxiv.org/abs/2304.00501

https://docs.opencv.org/

- Posture Classification

Within the bounding box region, the model performs the posture classification algorithm to determine whether the dog's current position corresponds to the position "sit." This is accomplished through analyzing body orientation, limb angles, and relative position of the torso to the legs [96]. The classifier is trained with labeled data (0 for "sitting" and 1 for "not sitting"). In neural networks, there are parameters called weights and biases. Weights determine how strong each pixel or feature in the input image influences the final output, and bias adds an adjustable offset to the weighted sum of inputs before the activation function (e.g., ReLU or Sigmoid), allowing the output to be meaningful even if the inputs are difficult to predict one way or the other. While the model is training, the model's weights change after each image. The weights changing is the model "learning" what it needs to classify.

Here is a breakdown of the training process [97]:

- Forward Pass: The input image passes through each neural network layer, multiplies by the weights at each step and adds bias, then the activation function
- Loss Calculation: The model compares its prediction with the correct label, and the loss function measures how far off the prediction is.
- Backpropagation: The ML framework calculates how much each weight contributed to the error. Then uses the gradient information to adjust each weight slightly to reduce the loss.
- Optimizer: Applies the small changes to the weights and gradually "shifts" the model's parameters to improve accuracy.
- Once the model is trained and ready to use, the weights no longer change. It will apply the weights to make predictions, called inference. Essentially, the trained model will apply its "knowledge" to new images. Later, the model can be fine-tuned, meaning that if a higher accuracy is needed or needing more labels like standing or laying down, the model can be retrained by loading the previous weights and training from there.

https://www.mdpi.com/2673-4591/92/1/41

https://www.geeksforgeeks.org/deep-learning/training-of-convolutional-neural-network-cnn-in-tensorflow/#

- Decision Output
  At inference, the classifier produces a confidence score. If the confidence score exceeds a defined threshold (e.g., 0.50), the output will be "sitting" [98]. After, it will send the result to the backend to trigger the feeding mechanism. The outcome is logged through the mobile app. After inference, there will be a confidence score that will be compared to a threshold. To reduce false positives, 5 consecutive frames must have confidence scores over the threshold. The result triggers the feeder's dispensing mechanism and logs the outcome to the mobile app.
  Example: confidence score of 0.75 > 0.5 label = "sitting" dispense food

NOTE: If the model runs on the microcontroller, the model will have to be converted from PyTorch or Tensorflow to a TinyML model due to hardware limitations [99]. The conversion would be done through a framework called ONNX [100], which can convert any deep-learning framework into another framework.

https://arxiv.org/abs/1506.02640

https://www.electronicdesign.com/techxchange/article/55238406/edge-computing-and-tinyml

https://onnx.ai

Training From Scratch Vs. Fine-Tuning Pretraining Models

There are two primary approaches to developing a model for a specific task: training from scratch and fine-tuning [101]. Training from scratch means initializing all model weights randomly and allowing the network to learn every visual feature directly from the dataset. This approach requires a large amount of data and computation but enables the model to learn features specific to the objective. In contrast, fine-tuning starts with a pretrained model, so it has already learned general features from a large dataset like COCO or ImageNet. It adjusts the weights slightly using task-specific data, leveraging previously learned features (edges, shapes, textures, etc,) to accelerate learning and improve performance with a smaller dataset.

There will be a model for the detection and another for the classification. For the detection and bounding box, using a pretrained model like YOLOv11 that is trained on multiple different classes , such as dogs, with high accuracy and saves the intensive computation of training a new model with potentially lower accuracy.

Performance metrics show YOLOv11 reaching an mAP 50-95 about 55% of the time for the multi-class dataset COCO and is faster compared to other models performance.

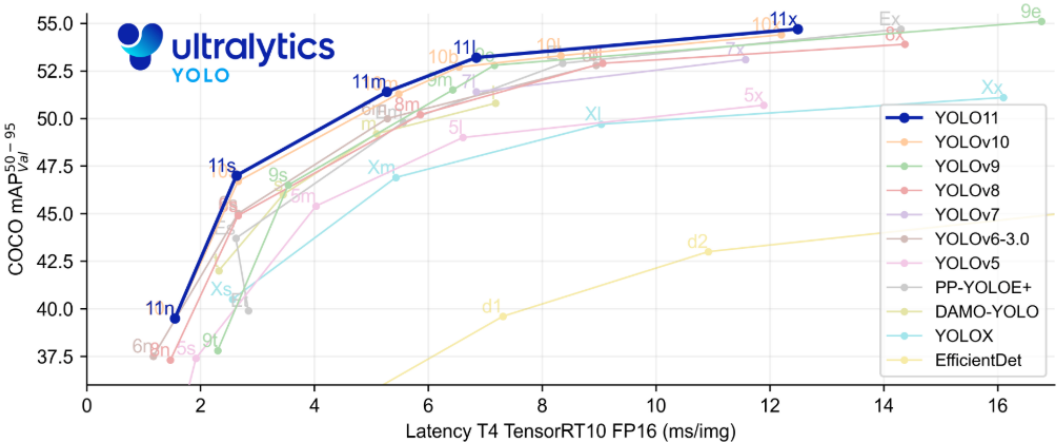


***Figure 3.3.3: Ultralytics YOLO Performance Metrics** [2]*

https://arxiv.org/html/2408.06663v1

Table 3.3.4.1: Training From Scratch vs. Fine-Tuning

Aspect	Training From Scratch	Fine-Tuning Models
Data Requirements	High: typically thousands to tens of thousands of labeled example images to converge	Lower: fewer task-specific images required
Training & Computation Time	Longer: must learn simple to complex features from scratch; slower to converge	Shorter: high accuracy; already "knows" general visual patterns; more resources
Feature Specialization	Learns specific features of your environment (e.g., dog breeds, lighting)	Carries bias from pretrained dataset (e.g., generic dog poses, backgrounds)
Resource Demand	High: more epochs and memory are required	Moderate: fewer training iterations and smaller learning rates
Control & Flexibility	Full control over architecture, initialization, and learning dynamics	Limited to pretrained structure (e.g., YOLOv11)
Risk of Overfitting	Higher: Small training datasets lowers overall performance so data may not generalize	Lower: already pretrained with high performance

When To Use	Have a large, diverse, labeled dataset; need full control over architecture	Have limited labeled data; want fast, high-accuracy results

Training the neural network from scratch is the preferred approach since it allows the model to learn the features that are unique to the feeder's camera perspective. Although fine-tuning a pretrained model is a faster approach with less data, training from scratch provides control over the architecture as well. This approach will have a highly optimized and adaptive model tailored to Sit & Chow's environment, minimizing bias and improving long-term reliability.

Frame Composition and Detection Reliability

For accurate detection, the dog must be sufficiently visible in the camera's field of view. The model relies on body cues, so incomplete or cropped poses can reduce detection confidence or cause misclassification

To maintain reliable performance:

- At least 70% of the dog's bog should be visible within the frame [102], so the dog should be positioned about 1-2 feet away from the camera.
- The head and upper torso should not be cropped. This reduces overall confidence level [102].
  https://www.mdpi.com/2073-431X/11/1/2
- At least the upper body and front legs must be in view since the detection relies on the angle between the body and legs.

MCU vs. Cloud

Latency & Responsiveness

- MCU
  - Path:

    Camera ⟶ Local Inference ⟶ Actuator
  - TinyML models were created to implement machine learning models on low-power microcontrollers, such as Arduino and ESPs [103].
    https://www.mdpi.com/1424-8220/25/10/3191
  - Runs offline so no network variability.
- Cloud
  - Path:

    Camera ⟶ Wi-Fi ⟶ Server Inference ⟶ Result ⟶ Device
  - Allows for higher quality inputs (e.g.,480x640) and more model capabilities running on either CPU or GPU [104].
    https://arxiv.org/abs/2111.06061
  - Introduces network latency.

    - Estimated by image capture and encoding time, data transmission time (upload speed), network round-trip delay, server processing time (inference), result transmission [105], and action time.

https://arxiv.org/abs/2405.15079

Accuracy & Model Capacity

- MCU
    - Model size must be compressed with a smaller input size (e.g., 96-160 px) [106].
    - Supports binary classification (e.g., sitting vs. not sitting).
    - Lower accuracy and more sensitive to noise [106] (e.g., light, texture).
    - Only runs on CPU [106]

  https://hongyumiao.github.io/assets/sec22-nn-mcu.pdf
- Cloud
    - Can run high-level detection models (e.g., YOLO) and larger input sizes [107].
    - Higher accuracy and recall [107].
    - Possible GPU use instead of CPU [107].

https://acecloud.ai/blog/high-performance-computing-with-cloud-gpus

Power & Hardware Load

- MCU
    - More workload doing both detection and embedded functions.
    - Avoids constant uploads.
- Cloud
    - Offloads detection computation, where MCU only needs to capture, compress, and transmit frames.

Privacy & security

- MCU
    - Local raw frames preserves privacy.
    - Only events/labels are open since the data needs to be sent to the mobile app. (e.g., “sit detected 9:00AM”)
- Cloud
    - Images/video require secure transport between client and server [a].
    - Must be securely stored [b] (e.g., encrypted).

https://firebase.google.com/docs/rules

[b] ISO/IEC 27018:2019 – international standard

Table 3.3.4.2: MCU vs. Cloud Trade-offs

Aspect	MCU	Cloud

Computation Power	Limited: memory, processing speed, power consumption constraint	High: able to run deep learning models with large architectures and higher accuracy
Model Accuracy	Lower Accuracy: requires small and simple TinyML model	High Accuracy: supports complex models; more resources
Latency	Low-None: inference occurs offline	Higher: network transmission and server response time
Privacy & Security	High: data is local	Possible risk: image data leaves the device; must be securely stored
Maintenance	More difficult: updates through firmware or manually	Simple: centralized monitoring and debugging
Power Consumption	Higher: continuous inference	Lower: power usage offloaded to server

After comparing the trade-offs, hosting the model on the backend server serves the most practical, accurate, and efficient solution for the Sit & Chow smart feeder. Running the model in the cloud allows for higher computational resources, offering the possibility of using advanced deep-learning architectures (e.g., YOLO) that would not be possible on an MCU. This enables broader coverage in areas like dog breeds, environments, and lighting. By running the detection model on the backend, retraining and model updates can be deployed quickly while working seamlessly in conjunction with the mobile app. Although cloud inference introduces network dependency, the feeder will operate with Wi-Fi, and the response latency will typically be under one second. Furthermore, the hardware design will be simpler by reducing MCU power load, memory constraints, and firmware complexity.

Chapter 4 – Standards and Design Constraints

4.1 Standards

4.1.1 Material Standards

Food & Drug Administration regulates the food of both humans and animals. Any part of the feeder that comes in contact with the dog's food should ideally be made of food-grade materials. They also need to be safe for the contact of the animal during feeding time and overall. Parts including the storage area, turning wheel, ramp, food bowl, etc all need to be made of food-grade plastics. Items made with metal parts need to also be made using stainless steel, food-grade aluminum, and not rusting items. There also needs to be no types of toxic coatings or paints that can be harmful for the dog or animal. Where these types of coatings are only allowed on internal and non-touching parts to the dog or the dogs food. Feeders that allow chemicals or bacteria to grow within the feeder and attach itself to the animals food could lead to making the dog extremely sick and by using FDA approved food-grade materials it helps to keep the health and safety of the pet.

https://www.fda.gov/animal-veterinary/animal-health-literacy/fdas-regulation-pet-food

UL 94 summarizes the flammability of the different plastic materials that are in our design. It details that standard on how easily a plastic piece of our design catches fire, how easily that fire can spread, how easily the fire can be extinguished, and whether our design is susceptible for dripping if something were to catch fire and can ignite something else. Our feeder is designed to house all of the components inside of it so that the owner and the dog are unable to reach such components, improving the overall safety of the design. However, since those parts are confined within the plastic they could be susceptible to heating up and catching fire. UL 94 also ensures that when choosing plastics we use resistant and self extinguishing plastics to prevent fire from occurring and spreading.

https://www.specialchem.com/plastics/guide/flammability-ul94

ANSI/FM 4910 elaborates on the cleanroom materials and the test protocol for flammability. This standard also talks about how far and fast flames spread along with heat release and toxic gas. Since our feeder will be inside this has an impact on the materials that we are allowed to use. Where this standard will test the specific categories to determine if the materials are safe to use around the house. If the feeder were to catch on fire it creates a risk for surrounding materials to also catch on fire. The standard also elaborates on the specific traits that a material needs to have in order for it to be deemed safe. The first trait is being resistant to ignition, then the limit of heat generation per area of the material, and a limit on the generation of smoke from the material that is being used. When deciding what type of material we are going to use to fit the housing for the feeder this will be important. Where we are going to use the type of plastic that we see fit to ensure no heat risk are potential throughout the whole system.

https://www.professionalplastics.com/professionalplastics/4910ansi1-2004.pdf

4.1.2 Electrical Safety

UL 60730 and IEC 60950 talks about electrical safety requirements if your running a DC power of 5V or 12V. Where if you're using one of these adapters you are required to design the system with safe installation and grounding practices. The first thing that is required is UL listen power adapter and the components that have been verified. Which also includes resistors that limit current to stop the overcurrent in the motors being used. It is also required that any exposed wire needs to be insulated so that it doesn't put the dog or the owner in danger when using the device. It also requires that there must be proper enclosing of all electrical components being used in the device. Which would include our camera, pcb, motor, and all other components. This is to prevent spilling of any liquid or any type of moisture from being exposed to the electrical components. All of these standards are required to prevent fires when the system is operating, help keep the safety of the animal and the owner, and to help keep the reliability of the system.

https://www.ti.com/lit/an/spradi6/spradi6.pdf

ANSI Z535 is a series of different safety signs with colors and labels being emphasized. Some of the different types of information that it regulates are hazard levels, colors for severity of the hazard, different symbols that represent different hazards. For our feeder this would impact areas like the motor and the ramp so that we make sure that it is warned not to put your hands or feet close to that area, although with our design the motor shouldn't be easily accessible to hands and feet. The power supply area would also be affected where if needed we would have to put a warning sticker next to it to symbolize shock. In our feeder the power supply and components that could be susceptible to this would be inside of the feeder which would eliminate the danger that it presents. Another area this would affect is the lid that keeps the storage of the food contained. Where we would have to put a warning sticker to keep the lid closed so no bacteria could get inside. It would also have to warn about cleaning and maintenance of the storage area to prevent a build up of bacteria.

https://blog.ansi.org/ansi/ansi-z535-4-2023-product-safety-sign-or-label/

ANSI UL 60730-1 summarizes the electrical controls within the household when they are applied automatically. Since our dog feeder will be plugged in using power when the owner is away from home this will apply to our design. One of the important things it talks about is the prevention of the owner and the dog from being susceptible to electric shock. Most notably that they shouldn't have easy access to these parts that can cause such shock. With our feeder, the parts that could cause this will be placed inside of the feeder and out of the way from human and dog interaction. Another thing is talked about is limiting the temperature within the design. Specifically designed to keep the plastics and wiring from being exposed to too much heat and either catching fire or being faulty. An example of this is if the motor on the feeder is running for too long or gets jammed which could lead to significant heat building up in the feeder. The next thing it summarizes is the testing requirements for our design. Where we have to specifically test for temperature rising, endurance, and the overload of our system. Where each scenario has to be safely controlled if it were to fail during operation. For our feeder, this would relate to different situations that kept the owner and the dog safe if the system were to fail. For example, If our ESP32- S3 fails during its feeding logic it doesn't just continuously feed the dog.

https://store.astm.org/b0117-19.html

4.1.3 Mechanical safety

ISO 12100 is a standard for risk assessment of different machinery and other hazards that could potentially arise. It determines the limit of the machinery being used and evaluates the level of risk that it could create. It also places a standard on what level those risks are allowed to be until it is determined that the risk is no longer tolerable. For our design, we would have to verify that the motor we are using doesn't present any risks to the owner or the dog when using the feeder. As stated, we plan to integrate a ramp that the motor will dispense food onto where the ramp will guide the food down into the bowl for the dog's consumption, eliminating all risks pertaining to the motor not being at a tolerable level of risk. The standard also elaborates on the procedure to take protective measures. Where the designer should first attempt to eliminate all risks that could take place but if a risk cannot be eliminated the designer must implement a safe guard to eliminate that risk. For example, with our feeder we were first going to have the motor dispense directly into the bowl. But after further discussion our group decided that the motor being that close to the bowl presented a risk for the dog if the dog were to become curious and stick their mouth by the turning motor. To eliminate this risk our group presented the idea of letting the motor feed onto a ramp that would dispense the food. Eliminating the risk that the dog would get too close to the moving motor since it was now further away from the bowl. The standard talks more into different hazards such as thermal, vibration, and noise that also follow the same procedure of attempting to eliminate the risks then safeguard such risks. While also talking about when designing, risk of the intended use of the feeder should be accounted.

https://www.iso.org/standard/51528.html

ASTM B117 is a standard for the evaluation of how resistant a part is to corrosion. It works by putting the specific part in a controlled environment and exposing it to a salt spray for a specific amount of time. Not working as a pass or fail type of test but an estimate of how susceptible to corrosion that specific part is. With our dog feeder it's important to purchase materials that are corrosion resistant since the dog will be using the bowl and the surrounding areas to eat its food. Even if a part doesn't directly come in contact with the food its significantly important to reduce the amount of non resistant materials that we purchase. Non-touching parts that begin to corrode can still find a way to get into the food or the surfaces the food touches which would ultimately make the dog sick.

https://store.astm.org/b0117-19.html

4.1.4 Information Security Management

ISO/IEC 27001 is a standard that summarizes the information regarding security management along with how different organizations should organize their security risks. Stating how organizations should assess different risks and take safeguards to prevent them. Relating to our feeder from the access to the app where the owner will be able to access live feed of the feeder's camera and other features. It designs a step-by-step process from organization to follow that will help them prevent security risks. Stating that the process is to first identity any potential threats that could arise and any areas that are vulnerable or weak, then to attempt solve or create safe guards for those risks that could occur, then once in place to be able to manage those risk if any specific risks do occur. In relation to our feeder, we wil have to be able to monitor the specific safe guards when it comes to the app and overall prevent glitches and security risk from occurring.

https://www.iso.org/standard/27001

4.1.5 Software Cycle

ISO/IEC 25002:2024 defines a comprehensive model for evaluating software quality across eight characteristics, including functionality, reliability, usability, efficiency, maintainability, and security. This standard ensures that all software, from mobile to embedded firmware, meets quality expectations throughout its lifecycle. In Sit & Chow, ISO 25010 relates to maintaining high-quality user experience in the Flutter app, minimizing crashes or latency in the live camera feed, and ensuring that embedded firmware performs reliably during dispensing operations. Following this model helps as a guide for test performance metrics, prevent unexpected behavior, and maintain smooth interaction between the app, cloud, and hardware components.

https://www.iso.org/standard/78175.html

ISO/IEC 29119-1:2022 is a global standard for software testing processes, documentation, and techniques. It defines structured phases such as test planning, design, execution, and closure to ensure that software components meet their intended requirements. This standard is relevant to both the Flutter app and the embedded firmware for our feeder. It ensures that features like scheduled feeding, LED alerts, and Wi-Fi reconnection are verified systematically before deployment. By following this standard, our project will maintain clear testing documentation and validation evidence to guarantee that the system functions correctly under all expected conditions.

https://www.iso.org/standard/81291.html

4.1.7 Wi-Fi Communication

The IEEE 802.11 standard governs wireless local area network (WLAN) communication and defines specifications for Wi-Fi protocols such as 802.11 n and 802.11 ac. It focuses on ensuring reliable and secure wireless data exchange. For our feeder, this standard is critical for communication between the device and the mobile app. Implementing Wi-Fi under IEEE 802.11 ensures that the ESP32-S3 maintains stable and encrypted connections with the cloud through WPA2 or WPA3 protocols, reducing the risk of data interception or unauthorized access. Following these guidelines ensures consistent signal range, throughput, and reliability for remote feeder control.

https://standards.ieee.org/ieee/802.11/7028

4.1.7 Cloud Security

ISO/IEC 27017 provides additional guidelines for information security in cloud computing environments, extending ISO 27001 with cloud-specific controls. It covers responsibilities shared between cloud providers and users, focusing on data protection, service configuration, and access control. In relation to our feeder, this applies to Firebase services such as Firestore, Cloud Storage, and Authentication. It ensures that our data storage follows best practices, such as role-based access, secure backups, and encryption of sensitive data, while clearly defining how our application and Firebase each handle user information.

4.1.8 Artificial Intelligence

ISO/IEC 2382-37 and ISO/IEC 20546 define terminology, governance principles, and lifecycle considerations for AI systems. They emphasize transparency, fairness, reliability, and responsible use of data and algorithms. For our detection model, which recognizes when the dog is sitting, this ensures that training data is ethically collected, models are validated against bias, and performance is monitored continuously. Following these standards ensures that the model’s decisions are explainable, reproducible, and do not compromise user or pet safety.

https://www.iso.org/standard/68301.html

ISO/IEC TR 24029-1 provides guidelines for evaluating the robustness and trustworthiness of AI models. It focuses on adversarial resistance, interpretability, and model reliability. In the context of our feeder, this standard helps assess whether the posture detection model remains accurate under varying lighting conditions, dog breeds, and angles. Implementing its recommendations allows us to validate the model's consistency, ensuring that it triggers feeding only when the dog's "sit" posture is genuinely detected.

https://www.iso.org/standard/77608.html

ISO/IEC 30170 standardizes Python's syntax and behavior, ensuring that software developed in Python is portable and consistent across platforms. This is relevant for the backend model server running the posture detection model in Python. Adhering to this standard ensures code interoperability, predictable behavior during inference, and compatibility with other standardized Python environments, helping maintain a stable backend system.

https://www.iso.org/standard/57918.html

4.2 Design Constraints

4.2.1 Camera Placement

The placement of our camera onto our product is significantly important in the reliability and effectiveness of our feeder. The OV5640 camera that we have decided to use has a field of view of roughly around 60-90 degrees. If the dog were to approach the camera from the side or sit down on one or the other side it creates a risk that the camera might not be able to pick up that the dog is by the feeder, which would result in the communication to feed the dog because it has sat from not going through and ultimately the dog not being feed. To minimize this problem, we placed an importance on the camera being positioned directly in the middle of our feeder so that if the dog were on either side of the feeder, the camera would be able to pick up a portion of the dog. Before we changed the positioning of the camera to the center, our first thought was to place the camera on the left side of the feeder so that the storage system wouldn't get in the way of wiring the camera or placing the PCB in the right spot. After talking it over, we ultimately agreed that if we were to place the camera on the left side of the feeder and the dog approached the camera from the right side the chance of the camera not picking up the dog approaching or sitting increased significantly. This results in us switching the positioning of the camera from the left side to directly in the middle.

4.2.2 Food Storage Placement/Size

With the change in the positioning of our camera, it also impacted the placement of the storage area for the dogs' food. Originally, we had the storage area taking up most of the right side of the feeder. With the new spot for the camera now being directly in the middle, we needed to create some room for the camera and the components the camera needed to connect to. We ultimately decided to shrink the width of the storage container so that it only will take up half of the width, allowing the camera and the components related to the camera to be able to fit comfortably without any problems occurring.

With the movement of the storage bin, it also decreases the size of the storage container significantly. With a smaller storage bin, it creates a problem with making sure that there is enough food for the dog over an extended period of time. It also creates a hassle for the owner now having to make sure that the storage bin is full more frequently than if the container was larger. To solve this problem our group decided to add an ultrasonic sensor to track the amount of food that is in the container at all times. Where the owner will be given a message through our mobile app that communicates with them that the storage bin is running low. In return, creating a more stable and reliable feeder for the owner and the dog.

4.2.3 Reliability

In order for the dog to be fed the right portion of food and the feeder to be reliable, the motor we picked out played a significant role in that. We needed a motor that could be easily connected to the storage container in order to limit the cause of the system getting jammed. With the rotating motor, we will be able to place it directly under the storage area. Having the rotating wheel directly under the storage container helps to improve the reliability of our feeder by limiting the space between them. So those pieces of food aren't able to get stuck between the motors. Since the storage area is going to be positioned on the right side of the feeder, the turning motor will also be on the right side of the container and wil have to turn counterclockwise in order for the food to dispense to the middle of the feeder. Once the command has been given to feed the animal, the motor will begin to dispense and rotate into a small ramp that will feed the food to the bottom of the cup designated for the dog to eat directly in the middle of the product. It was important that dispensing of the food ultimately reached the middle of the feeder so that the dog would be directly in the line of the camera in order and the owner could watch over the dog, and the camera could pick up the dog.

The speed of the motor is also important when our group evaluates the reliability of it. A fast-turning motor has a higher probability of getting jammed or stuck when the command has been given out to feed the dog. So, we decided to use a slower motor to limit the problems that could arise from this action. A slower motor would also feed the dog slower, which in the overall idea of the feeder is also significantly important so that the dog doesn't choke or eat their food too quickly.

Another feature that will contribute a significant part to the feeder being reliable is the performance of our camera. Since our system relies on the camera to be able to pick up the dogs positioning, for example sitting or approaching the feeder, the camera must operate under a minimum resolution and pixel size. If the camera were to lack sufficient resolution and performance, it could result in the entire procedure of the dog being fed to not go through. In order to combat this problem, our group emphasized the importance of picking out the perfect camera fit for our feeder. The one we ultimately decided to use was the OV5640, which had sufficient resolution and performance to carry out the specific tasks that we had designed for it. The OV5640 also increases the reliability of our feeder from the extended peripherals that it has. It is easily integratable into the MCU that we decided to use, which was the ESP32-S3, which will allow for connection and communication between the two to be easy and reliable. It also has an extensive environment for learning how to use and operate the camera and the MCU, allowing us to focus more on what we want the camera to be able to do and pick up instead of working on simple tasks to initialize the camera. For example, teaching the camera and the MCU to pick up the dog approaching the feeder or when the dog has sat down in order to receive its food. Ultimately, improving the reliability of the camera to MCU connection and the reliability of the feeder as a whole.

Since our feeder operated without the owner having to be present during feeding times, another constraint that we have is making sure that the dog actually gets fed and the dog is okay during feeding times. To combat this problem our group decided to allow the owner access to an app that allowed them to access a handful of features. The first feature that the app contains is the ability to check in through the camera to see their dog during feeding times or whenever they desire. This would allow them to make sure that their dog is getting fed during their designated feeding times, and it would also allow them to check in to make sure their dog is okay while away from the house. The second feature the app contains is the ability to schedule feeding times and the portion of food that they wanted to feed their dog. Solving the problem if the owner accidentally forgot to set the times or the portion size while at home with the physical feeder, increasing the reliability of the system as a whole to make sure that their dog got fed.

4.2.4 Resource Limitation

A major constraint in Sit & Chow comes from the microcontroller. The ESP32-S3 microcontroller has limited RAM, flash storage, and processing power, which restricts the complexity of the algorithms that can run on the device. A TinyML was considered to lighten the computation load of the detection model, but it was ultimately not chosen since the computation load would likely still be too much for the microcontroller to handle alone. Advanced operations such as computer vision or large-scale data buffering are unable to run locally. As a result, the machine learning model had to be offloaded to the cloud server, so the MCU is limited to essential control tasks (e.g., motor, led, camera, etc.). The embedded firmware must be optimized for low power consumption, so unnecessary loops, delays, or polling can be avoided. Power-efficient programming techniques, such as interrupt-driven routines, will extend operational life.

4.2.5 Real-Time Performance

Sit & Chow must be able to respond to events with predictable timing. When the dog approaches, the depth sensor must detect its presence, the camera must capture the current frame, and the detection model determines whether the dog is sitting within seconds. Because of this, there are constraints applied to the embedded software and backend server. Precise control is needed for the feeding mechanism and signaling while the detection model must be able to process frames fast enough to trigger the motor within 1 minute of the scheduled time. In order to consistently have low latency demands, an optimized pipeline, efficient network protocols, and appropriate resource allocation between the MCU and mobile application.

4.2.6 Network and Connectivity

The feeder relies on Wi-Fi communication to perform the necessary functions and tasks of the feeder. Therefore, its performance is directly affected by the strength and stability of the network environment. Packet loss or signal interference could result in missed commands or delayed notifications, especially during a live camera stream. To ensure reliability, the software must include mechanisms such as automatic reconnection and local buffering for scheduled commands. Bandwidth plays a major role since it can constrain video quality, which may lead to longer posture analysis. These network constraints showcase the need for a communication layer capable of handling variable connection conditions while maintaining a responsive user experience.

4.2.7 Data Storage and Management

Firebase is a cloud-based server. While cloud-based servers offer scalability, there are cost requirements for reads, writes, and bandwidth. These cost requirements restrict the amount of sensor data and logs that are uploaded to the mobile application. In order to manage this, data lifecycle policies, like periodic log compression or deleting outdated entries. Furthermore, synchronization between the feeder and backend server must avoid conflicts and redundant updates that might cause inconsistent feeding schedules. Database indexing must be tracked to remain in the storage restraints and maintain performance. Data could be saved to the MCU itself through non-volatile memory, but it does not have enough memory to hold feeding logs.

4.2.8 Animal Safety

One of the most important constraints is the ability to create a feeder that is safe for the dog. The first way we aim to accomplish this is by placing the motor and the turning wheel a safe distance away from the bowl that the dog will be eating out of. Where the food will collect in each of the pockets within the turning wheel and dispense onto a ramp, where the food will slide down into the bowl for eating. Ensuring the motor and turning wheel are away from the dog bowl decreases the risk that the dog will come in contact with this motor where it could get hurt. Another way we aim to keep the feeder safe and healthy is by using food-grade materials when designing areas of the feeder that will come into contact with the dog or the dog's food. Materials such as stainless steel, grade aluminum, and materials that are not susceptible to rust. We also will not be using any toxic coatings or paints on parts that will come in contact with the dog or the dog's food to help prevent chemicals from attaching themselves to the bowl or the food.

Since one of the most important constraints is creating a product that keeps the animal safe and healthy and feature that we had to add was the ability to check in with their animal through the app. Owners might feel uncomfortable feeding their dog automatically or miss their dog after a long day at work or away from the house. To combat this problem so that the owner knows their dog is safe, we created an app where the owner will be allowed to join a live feed of their dog while either eating or just randomly. For example, if the owner was away for an extended period of time and they joined the live feeder and found their dog was sick or hurt, the owner would be able to take action, increasing the overall safety of the dog from this feeder.

Another area that is important when it comes to improving the overall safety of the feeder is creating a system that exposes no electrical components to the dog or the owner. The owner of the dog being away during feeding times presents a safety hazard if there are exposed electrical components that the dog might get into when eating. It also presents a problem when it comes time for the owner to clean the feeder or refill the storage area. In order to combat this constraint, we designed our feeder to be able to fit all of the electrical components inside of the feeder and its specific spots that we deem safe and far away from human or dog interaction. The first specific example of this is the motor being placed to the right side of the feeder and not directly next to the dog bowl since we are using a ramp to get the food to the dog bowl. This will allow us to comfortably fit the motor and all of the connections to the motor without worrying about putting our electrical components too close to the bowl. The next example is that we are using the back left side of the feeder to place the rest of our electrical components and our PCB. Allowing us to connect and place all of our components without getting too close to the dogs' bowl and the owner's storage bin. All of our electrical components will also be covered with healthy and safe materials which will help to ensure a safe environment for the dog and the owner.

4.2.9 Budget

Another constraint that has a significant impact on the parts that we order, and the overall feeder, is the amount of money that we spend on every part. Overall, we are on a relatively strict budget which has a significant impact on the specific parts we choose. Where we have to juggle having the best of the best part and having a part that we predict will be able to accomplish the designed task. For example, when picking out a camera we could have gone with the highest resolution, fastest camera but that would have cost a significantly higher price than the OV5640 that we ultimately chose. Where we determined that the OV5640 was sufficient enough and cheap enough to take out every task that we needed it to take out. The PCB design also plays an important role in the budget of our product. After doing research, our group has already determined that the full price of the PCB's will take up a large part of our planned budget, having a direct impact on the price that the rest of the parts can.

4.2.10 Time

One of the most impactful constraints that have been placed on us as a group is the time limitations that the senior design course has placed on us. We have been given a specific set of time to be able to create a group, organize with that group, brainstorm a project, research that project, order parts for the project, implement those parts, and test our project. Which emphasizes the importance of staying organized and ahead of schedule throughout the whole entire time period that we have been given.

Managing our time and staying ahead of schedule also plays a significantly important role in the success of our group. This senior design course is one of the first times that we have been given free will to organize and manage our time without having due duties every week to get small parts of the project done. Instead, we have a few that require a large portion of the project to be turned in. Which ultimately punishes people who decide to procrastinate and people who are disorganized with their time. It also forces each student in the group to trust their partners to get their portion of the project done on time. If a couple of the people in a group procrastinate, it results in a direct impact on the other members' grades.

The time we have, and the management of our time, is also important from the newness of this senior design class. Other classes operate by having a guide on how to accomplish a task where students just have to follow along. Senior design allows students to do their own research, decide their own product, use whatever parts they see fit, and allocate time to see it. Adapting to this new type of class is extremely important in order to be successful. As students, we have also never created or implemented our own PCB before. The only time we have been required to work on a PCB is junior design class where we were given step by step on how to create one. There was never any question if we were doing the right design or using the right parts. Senior design introduces us to creating our own and doing extensive research to verify that our PCB will operate with the parts that we have chosen, making allocating our time and staying ahead of schedule extremely important.

Another important time constraint is ordering the materials that we see fit for the project. Ordering parts can take a significant amount of time depending on the availability of the part and the location that the part is being delivered from. One way that our group solved this problem was by choosing specific parts that were widely available and having a larger ecosystem. Which would allow for easier access to ordering the parts and getting them in a shorter amount of time. But it also plays an important role in choosing those parts that we are going to implement and order. We aren't able to sit and dwell on which part we think would be the best fit for our project for an extensive amount of time. Taking too long to choose a part could result in us not getting the part in time and ultimately failing the class.

4.2.11 Functionality

In overall, being able to have a functional project can be considered the most important constraint that we have as a senior design group. The ultimate goal of our pet feeder is being able to feed the dog on time, dispensing the food when the dog has sat, integrating the app for parietal control, and a safe and healthy feeder for the dog. Failure of these goals will result in the failure of the class, so emphasizing the importance of being successfully able to integrate the different parts to form our feeder is a priority. Specifically, we need our camera to be able to pick up when the dog has entered the camera's view, then we need it to be able to identify when the dog has approached the feeder and sat down. Once the dog has sat down, we need our feeder to be able to realize this, then send a message saying the motor can now start dispensing the food for the dog. We also need to be able to use the app to alter the portioning of the food and the time that the animal will be fed.

Chapter 5 – Application of ChatGPT and Other Platforms

5.1 Case Study 1

Please refer to Appendix E, [1a] and [1b] for the Case Study 1 given prompt and outcomes from ChatGPT.

When we look at the answer to Case Study 1b, we can see that it's a good explanation. The good part is that it covers all the major systems we would need for a dog feeder project: electrical, and software, and it connects them nicely. We like that it starts with the big picture of "automating pet feeding efficiently" before breaking things down. The electrical are really detailed that using a 12V adapter, regulators for 5V and 3.3V, and protection features like a fuse and switch make it sound like the writer really knows how to keep things stable and safe. We also like that it suggests realistic components such as the ESP32 or Arduino Uno, which are common and affordable choices for projects like this. The part about sensors is especially nice when it gives a variety of options like load cells, ultrasonic, and ToF sensors, and that shows awareness of different ways to measure food levels. Finally, adding IoT features such as app control through Wi-Fi or Bluetooth makes the project feel modern and practical which will be useful by pet owners today.

However, what's missing is a sense of why each decision was made. It feels like a list of parts and features without explaining the reasoning behind them. For example, why use ESP32 over Arduino if Wi-Fi isn't needed? Why might an ultrasonic sensor fail compared to a load cell in certain lighting or humidity conditions? It also doesn't talk about challenges or limitations such as motor torque not being enough to handle heavy kibble, or the issue of dust interfering with sensors. Finally, while the ending summarizes the project well, it would be stronger if it included a short reflection: maybe something about how automation can improve pet care or how designing such a system taught something new about engineering integration.

In addition, when we asked this question, we did not specifically mention that this project was for students of electrical and computer engineering. So, when it came up with the answer, it also included the mechanical part. Although we did not use it, this was also a part that needed to be noted so that the food would not have problems or get stuck when being pushed out.

Overall, for the first steps of how to build this project, this will be a pretty good answer as ChatGPT has explained the basic software and hardware used to build this. However, since it is not very clear why to choose those software or hardware, we need to research, compare to see which part is most suitable and make the final choice.

5.2 Case Study 2

Please refer to Appendix E, [2a], [2b] and [2c] for the Case Study 2 given prompt and outcomes from ChatGPT and Google AI.

In this part the answers of ChatGPT and Google AI are very different. At first when we read the answers, we felt both were reasonable, but this made us confused because we did not know which speaker is suitable for this project.

Starting with Case Study [2b], we think the explanation feels practical and easy to apply. The active buzzer is presented as a simple, energy-efficient solution, and that's a real advantage when building a project that might run on limited power or batteries. I like how it only needs a single digital signal from the microcontroller. That would be helpful when we're still prototyping and don't want to complicate things. However, its main drawback is that it's too simple. The monotone beeping doesn't really engage pets, and over time, it could even make them ignore the sound. The second option, the 8Ω, 1W speaker with a DFPlayer Mini, feels more professional and flexible. It gives the project a smart touch since it can play recorded sounds or voice prompts, which could make feeding time more interactive. However, it also introduces more work: extra wiring, coding, and power management.

Case Study [2c], on the other hand, dives deeper into the technical reasoning, which we really appreciate. It explains how dynamic speakers work and how they interact with the amplifier (like the MAX98357A). Personally, we find that very useful because it helps us understand not just what to use but why certain components fit together electrically. The dynamic speaker offers a nice balance between cost, sound quality, and availability. It's also durable and produces more natural sound compared to piezo speakers, which the explanation correctly points out as unsuitable due to their high-pitched tone and amplifier incompatibility. The best part is how the case connects impedance and power ratings, explaining why an 8Ω, 3-5W speaker is ideal. That kind of detail prevents costly mistakes in real builds, like blowing a speaker or overheating the amp. The only limitation I noticed is that it doesn't mention the sound enclosure or real-world placement much (like how you mount the speaker affects the clarity and volume a lot)

The plus side of the answers is that they tell our team where to start or what types of speakers we need to look into (e.g. we can use active buzzer or speaker). It also explains some of the advantages of using them. With Google AI, the plus side is that it gives us options and ranges of impedance and voltage when choosing speakers, and it also gives other options for users to think about or explain why they are suitable or not. Google AI also suggests where to mount the speakers on the project.

The minus side is that, with both ChatGPT and Google AI, they only recommend options that they think are reasonable. This limits the amount of information we need to learn. ChatGPT does not give a specific comparison of the pros and cons when making a choice between active buzzer and speaker. In addition, with speakers, there are types that use different power sources or other parameters, ChatGPT does not give any opinion why they should not be used, but just chooses the option it thinks is good. With Google AI, after giving us information to help us understand more about the speakers and compare them, they did not cite the source. This made us have to learn more and reconsider whether this answer was correct or not, then we started to compare and choose.

In both platforms, they tried to help us visualize which speakers are suitable and what their functions are. However, we also need to check whether this amount of information is suitable or not to use for research.

5.3 Case Study 3

Please refer to Appendix E, [3a], [3b] and [3c] for the Case Study 3 given prompt and outcomes from ChatGPT and Google AI.

A good feature of ChatGPT's answer is that it clearly guides us when we start with 12V DC and convert it to 5V for use and 3.3V for other components to ensure the performance is not too hot compared to linear. In contrast to [3b], [3c] started with AC power and converted it to DC, which is quite good as it helps users visualize how to use the product or build the PCB. Google AI also suggests using barrel jack or USB-C to minimize the risk of high voltage circuit or causing damage to the inside of the product. Like ChatGPT, Google AI also suggests converting the power down to 5V and 3.3V for use

For [3b], when referring to backup power, ChatGPT suggested using Li-ion or NiMH. However, Li-ion should not be used in projects like this because they are prone to fire and explosion. In addition, there are other types such as Alkaline and lead-acid that are not mentioned. Whether they are used or not, ChatGPT did not provide any reasons or references. In [3c], Google AI only suggested one option for battery backup, which is lithium-ion or LiPo cell. We do not appreciate this suggestion because it is easy to explode and burn.

In the beginning, both [3b] and [3c] are quite complementary to each other to complete the circuit board. Both start from converting the current down to match the other components while still ensuring enough current to run them. In addition, they also contribute to helping my team visualize what we should have on the circuit board when making PCBs. Although the suggestions give a lot of good ideas, it also does not give too many options to choose from for battery backup. They also don't include examples of existing products or detailed suggestions on how to choose components. We got a good idea of how to build the circuit, but we should also consider which components are appropriate for the project.

5.4 Case Study 4

Please refer to Appendix E, [4a], [4b] and [4c] for the Case Study 4 given prompt and outcomes from ChatGPT and Google AI.

In this case study, we asked ChatGPT and Google's AI to give us a list of technologies that could be used for our project. Here we are trying to gauge how useful these AIs can be when talking about technology as a whole. We want to see if it truly understands how these technologies work and how they fit into certain scenarios. If we look at the response given by ChatGPT in [4b], we can see that it gave us an in-depth list sorted by the kind of technology that it should be listed under such as frontend, backend, and others. This kind of response is incredibly useful for us as it gives us a list of possible solutions for our project while listing the characteristics of each framework. This lets us analyze which framework would fit the best in our project without having to go through extensive research on each technology. It even covers the less important aspects of our project such as authentication, analytics, and even machine learning options. Furthermore, at the end of the list of technologies, ChatGPT gives us several suggested stacks we could use for our project. It briefly gives us each technology and what they would fit into and why we would use all of them in tandem. When it comes to designing the project, this is an incredibly useful tool to be able to see how everything would come together.

Now we will analyze the response from Google's AI in [4c]. This response is quite similar to ChatGPT's in that it gives us a list of frameworks that we could use for each section. The main difference is that Google's AI has a less extensive list, giving us just three options for each section and then only having three sections in total. The response only covers the front end and two different parts of the backend. There is quite a lot less material given to us by Google's AI. However, it does tailor the response to why we would use each technology for a dog feeder app. ChatGPT's response simply gave us the pros and cons of each technology, Google's AI gives us why it would be good in our specific scenario. Then at the end of the response, it also gives us recommendations for which frameworks we should consider using.

After observing the response of both AIs to the prompt, we can see that using AI as a way to gauge which technologies to use in a project is extremely effective. If you use each one individually, you can absolutely achieve strong results. But if you use multiple AI responses in tandem, you can get a large overview of the technologies you can use for your project and come to an educated conclusion as to which ones you should use. Overall, using AI in this way is incredibly useful.

5.4 Case Study 5

Please refer to Appendix E, [5a], [5b] and [5c] for the Case Study 5 given prompt and outcomes from ChatGPT and Google AI.

In this case study, we gave both AIs what we wanted out of the mobile application. We gave them the functionalities it should have and what things should be included. Then we requested each AI to tell us how we should construct the mobile application. What we intended was for them to tell us which pages we should have and what should be included on each page. If we first look at Google's AIs response in [5c], this is exactly what we get. It gives us what we should have in our frontend and the other technologies we should use, and then it gives us a section on how the app should flow. It gives us three sections. One for the login screen, another for the devices screen, and the last one for the feeding schedule screen. It tells us what we should have on each screen and how each of these functions should work. It even tells us how we can implement Riverpod to continuously update the mobile app while we are on it. This is what was expected out of the response and is very useful.

However, we got a very different response from ChatGPT in [5b]. ChatGPT starts very similar to Google's AI in that it assumes certain technologies and then gives us an overview of how everything will flow. However, after that point it diverges from the response of Google's AI. ChatGPT decides to give us a complete file tree structure that tells us what the app will look like when we construct it. It tells us where each file should go and which folders should be created. After that it starts to give us the code for some of the files. We did not specify that we wanted the code given to us, nor did we give it instructions on how we want the code to be formatted. But it decided to give us the code anyways. After that, it did something similar to Google's AI in that it went over functions to be implemented on certain pages. The difference is the amount of detail that it gives us. ChatGPT goes incredibly in depth as to how each function will work with new functionalities for each one. Then it finishes by giving us instructions on what to start with and how we should continue moving forward.

There are two ways to interpret this response. The first is that this is incredibly useful and heavily reduces the amount of work necessary to construct the mobile application. The second is that this is scary in that it reduces our own autonomy and allows us to completely rely on the AI to complete our work. However, we don't know that the code it generated works, and we should assume that it doesn't. If it does, then ChatGPT seems to be capable of creating the mobile app in its entirety as long as you instruct it to change certain things along the way. This can be useful for those who wish to reduce the amount of work to be done, but it also takes away from the learning experience of creating a mobile app from scratch. So, using AIs to create a mobile app is a double-edged sword and should be used with caution. Although I think if you use the response from Google's AI, it is much less likely to hinder your learning and will likely help you to understand how the app should flow in general.

Chapter 6 – Hardware Design

6.1 Power Supply Delivery

6.1.1 Power Jack

The PJ-082BH is a horizontal, through-hole DC power jack designed for up to 10 A at 36 V DC, making it suitable for power-input circuits in medium-to-high-current applications such as a dog feeder's main 24 V line.

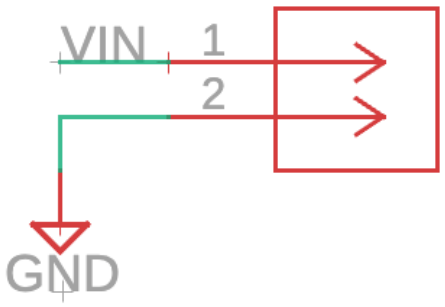


Figure 6.1.1: Power Jack Schematic

6.1.2 Battery backup

Backup power for embedded consumer devices is the secondary energy source that maintains essential functions when the primary supply fails. In a dog feeder, backup prevents missed feed schedules, preserves RTC and settings, and can notify the outage.

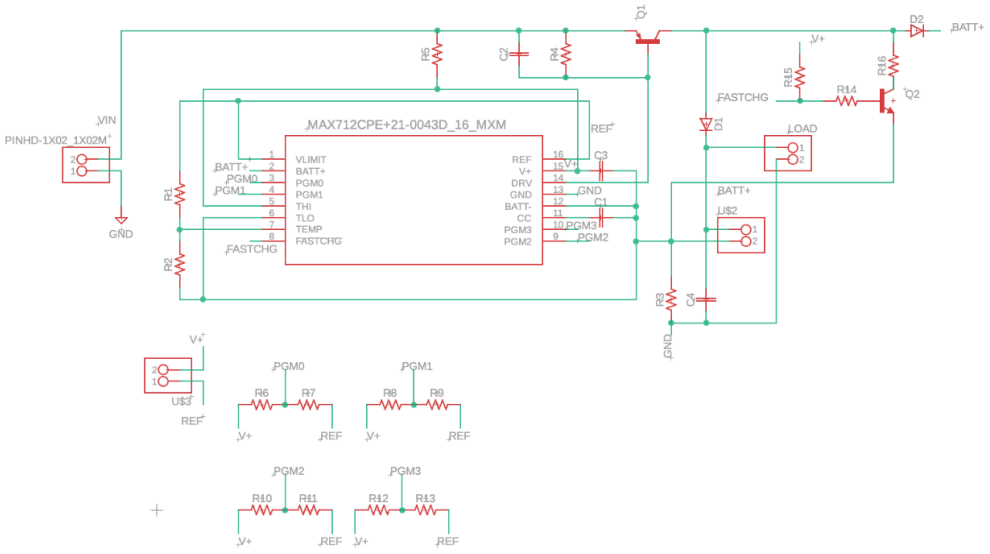


Figure 6.1.2: Battery Backup Schematic

6.2 Motor Driver

The TMC2209 PCB design revolves around integrating Trinamic's ultra-silent stepper motor driver in a compact, thermally efficient layout that minimizes noise and EMI. The TMC2209, packaged in a 5 × 5 mm QFN-28 with an exposed pad, can handle up to 2.8 A peak (2 A RMS) motor current at supply voltages from 4.75 V to 29 V. It includes internal MOSFETs, a 5 V regulator, UART interface, and diagnostic features—allowing simplified external circuitry and reduced component count

In the PCB layout, power integrity and grounding are critical. Both top and bottom ground pours should be used and stitched with multiple vias to the exposed pad to dissipate heat efficiently. Sense resistors (RSA, RSB) and filter capacitors must be placed as close as possible to the IC pins to reduce inductive paths. Low-ESR capacitors (typically 100 µF bulk and 100 nF ceramic) near the VS pins stabilize the chopper supply, while a 2.2 µF capacitor on the 5 VOUT pin ensures a stable internal regulator. The datasheet recommends connecting the exposed pad directly to the ground plane with numerous vias for optimal thermal and electrical performance

From a functional layout standpoint, the STEP/DIR and UART control lines should be routed away from the high-current motor traces to minimize interference. Traces to the motor outputs (OA1, OA2, OB1, OB2) should be wide and symmetric to balance current distribution. If using UART mode, the PDN_UART line should be kept short and shielded from the motor lines to maintain signal integrity. Proper decoupling between analog (VREF) and power sections helps reduce noise in current sensing circuits.

Finally, mounting holes should be connected to the ground pour when chassis grounding or EMI suppression is required. The TMC2209's quiet operation modes depend heavily on clean layout, short return paths, and solid ground planes, ensuring stable and silent stepper control even at high currents

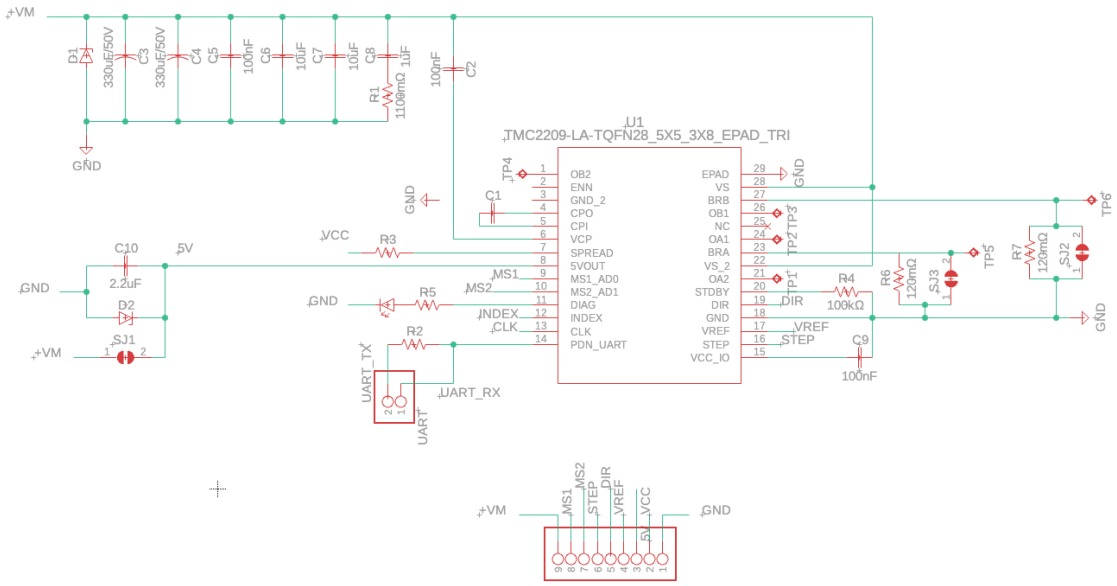


Figure 6.2: Motor Driver Schematic

6.3 MCU

The esp32-S3 is our MCU of choice for this project which connects to a wide range of peripherals that are needed in our design. We are using 2 regulators at values of 3.3 and 5 volts that connect to a wall jack which allows our feeder to be connected to outlets. For our speaker, we routed the 3.3 regulator to the amplifier of the speaker then connected the speaker to the output pins of that amplifier. We used two LED's which are connected both to our 5v and 3.3 volt regulator. The 3.3V LED is connected in series with a resistor then to a N chanel mosfet then back to the ESP32-S3. We have included a simple button in our design which connects to our IO40 node which acts as an override to the online app and allows for feeding the dog in person. The button also connects to a 3.3v input line which allows for clean stable logic when the user decides to press the button. We also included a programming header connected to the TXD0 and the RXD0 which will allow for data transfer. We also have a camera interface that requires a 2x8 male header to ensure a stable connection between the MCU and the camera. We are using the 3.3V input source to power the camera which is fed through a decoupling capacitor in order to have a stable voltage transfer.

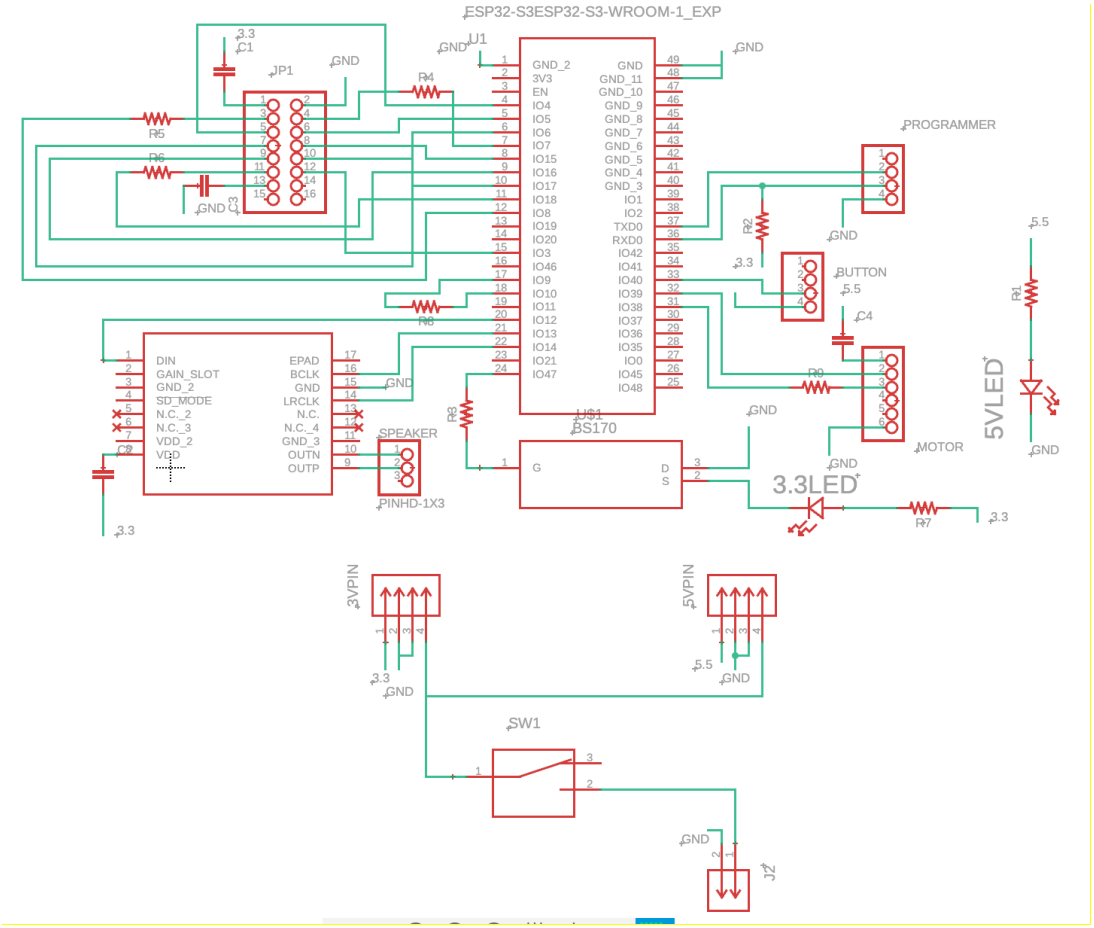


Figure 6.3: MCU Schematic

6.4 Regulators

In our design, we are going to be using the 12v jack power outlet which allows our dog feeder to be plugged into the wall at the owner's home. The regulators that we will be using are going to be 5v and 3.3v. Where we will be regulating the voltage coming into the different peripherals that we have in our design. The 3.3v regulator will connect to our camera, our amplifier which connects to our speaker, the 3.3 led, and our programmer. Where our 5-volt regulator will connect to the motor that we are going to be using to feed the dog and the 5-volt led. In our design, our voltage regulators are one of the most important components that we will be using. If we were to not use these regulators or these regulators were to fail it would result in our system failing and breaking.

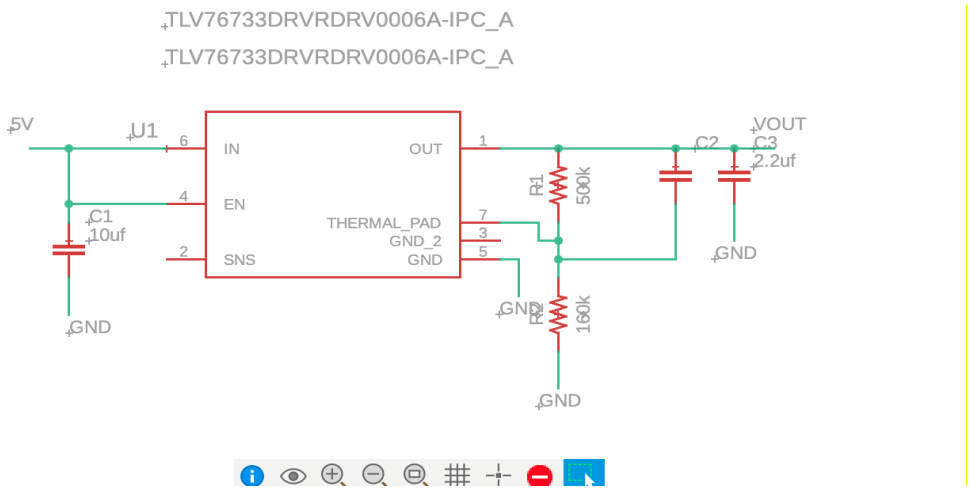


Figure 6.4.1: 3.3V Regulator Schematic

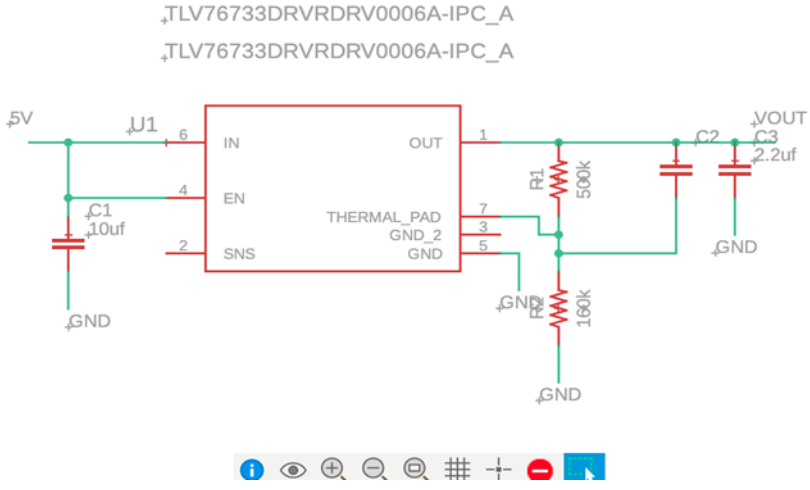
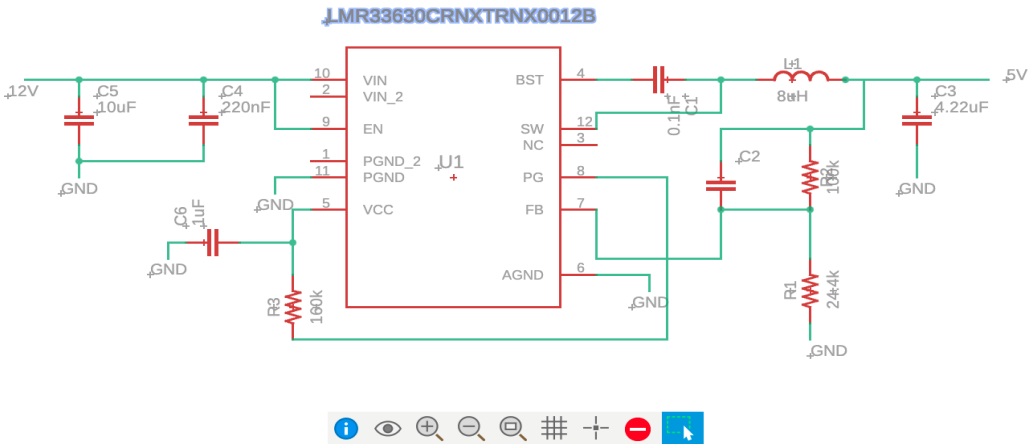


Figure 6.4.2: 5V Regulator Schematic


LMR33630CRNXTRNX0012B
12V
C5
10uF
C4
220nF
GND
VIN
VIN_2
EN
PGND_2
PGND
VCC
U1
BST
SW
NC
PG
FB
AGND
0.1nF
C1
L1
8uH
5V
C3
4.22uF
C2
R2
100k
R1
24.4k
C6
1uF
R3
100k

Chapter 7 – Software Design

This project involves the creation of software in multiple parts. There are three main parts to the software. This includes the embedded system, mobile application, and machine learning algorithm. The embedded system is how the MCU will handle all of the peripherals such as the camera, motor, speaker, etc. The mobile application is how the user will interact with the system. The mobile application will have to communicate with the MCU in order to facilitate the control of the system. Lastly, the machine learning algorithm will be hosted at the backend of the mobile application and will thus communicate with the mobile app and the MCU. Each of these has its own set of goals and tasks that must be met to complete the project.

7.1 MCU

The MCU will be in charge of handling all of the peripherals of the device. This means it will be communicating with the camera, the speaker, the amplifier, the motor, the motor driver, the button, and the depth module. It will also have to handle communication to and from the mobile application to receive the commands for these peripherals. The primary aspect of this project is the scheduled feed and will be connected to most of the peripherals.

To accomplish this feed, we first arrive at the scheduled time. At this time, the mobile app will send a signal via MQTT to the ESP32 to prepare the feed. The ESP32 will then turn on the camera and activate a live feed. Note that the ESP32 will also store the size of the feed and will translate this into the respective amount of turns on the motor. This feed will be sent back to the mobile app so that the machine learning algorithm can start analyzing the feed and search for a sit position. The web app will also send a signal to the MCU to start recording the data from the depth module for the machine learning algorithm. This will go on for 5 minutes, or some other period of time, and if a sitting position is detected, the food will be immediately dispensed, and the feed will be logged as a success. If no sitting position is detected, the food will be dispensed anyways, but the feed will be marked as a failure. Whenever the food is signaled to dispense, the ESP32 will use the stored information on the amount of turns necessary and then send the command to the motor. At this point, the scheduled feed has passed, and the information will be logged on to the mobile app. The diagram below shows the workflow of the entire process.

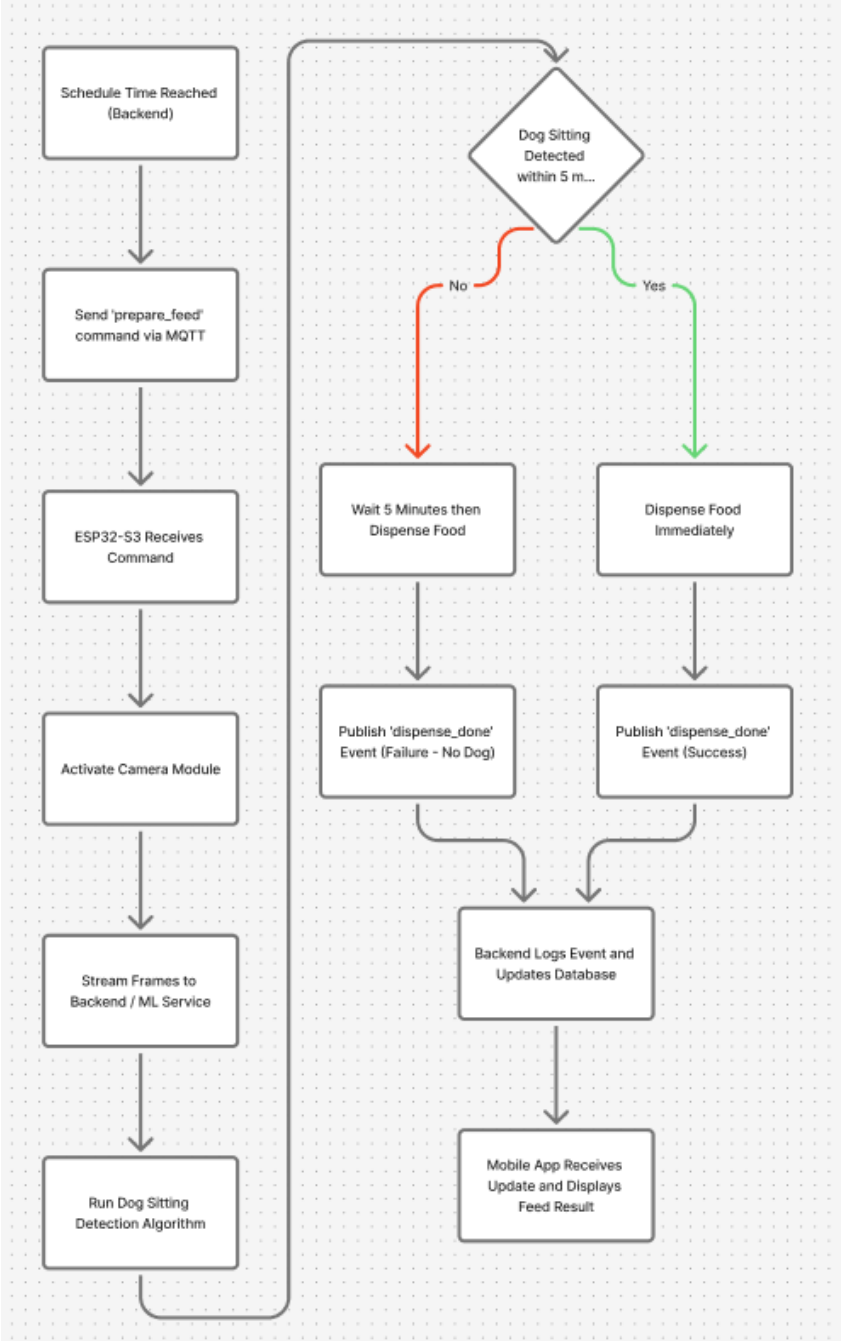


Figure 7.1.a: Scheduled Feed Workflow

There will also be a capacity check on the food after every scheduled feed. The ESP32 will turn on an ultrasonic sensor to check the capacity. If it is not low, then simply turn the sensor off and continue with normal activities. Otherwise, the ESP32 will send a message to the backend of the mobile app through an MQTT publish. The backend will then send a notification to the user that the food is low.

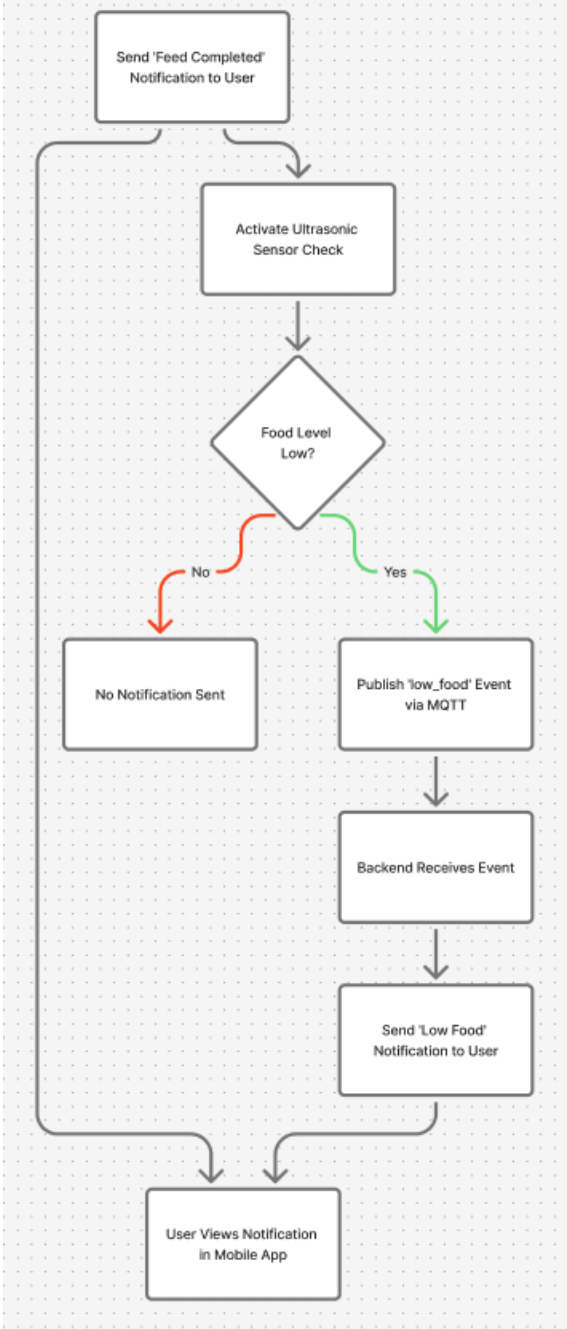


Figure 7.1.b: Notification Workflow

The ESP32 must also handle the flow of information from the camera to the mobile app should the user desire to view the live feed. This begins when the user enters the live feed on the mobile app. This will send a signal to the ESP32 to turn on the camera. At this point, the ESP32 will collect the data from the camera and send it as a JPED file through HTTP POST to the mobile app. The mobile app will receive the data and begin displaying the JPEG files continuously, resulting in a live video feed in the mobile app. The diagram below shows the whole workflow of the live stream.

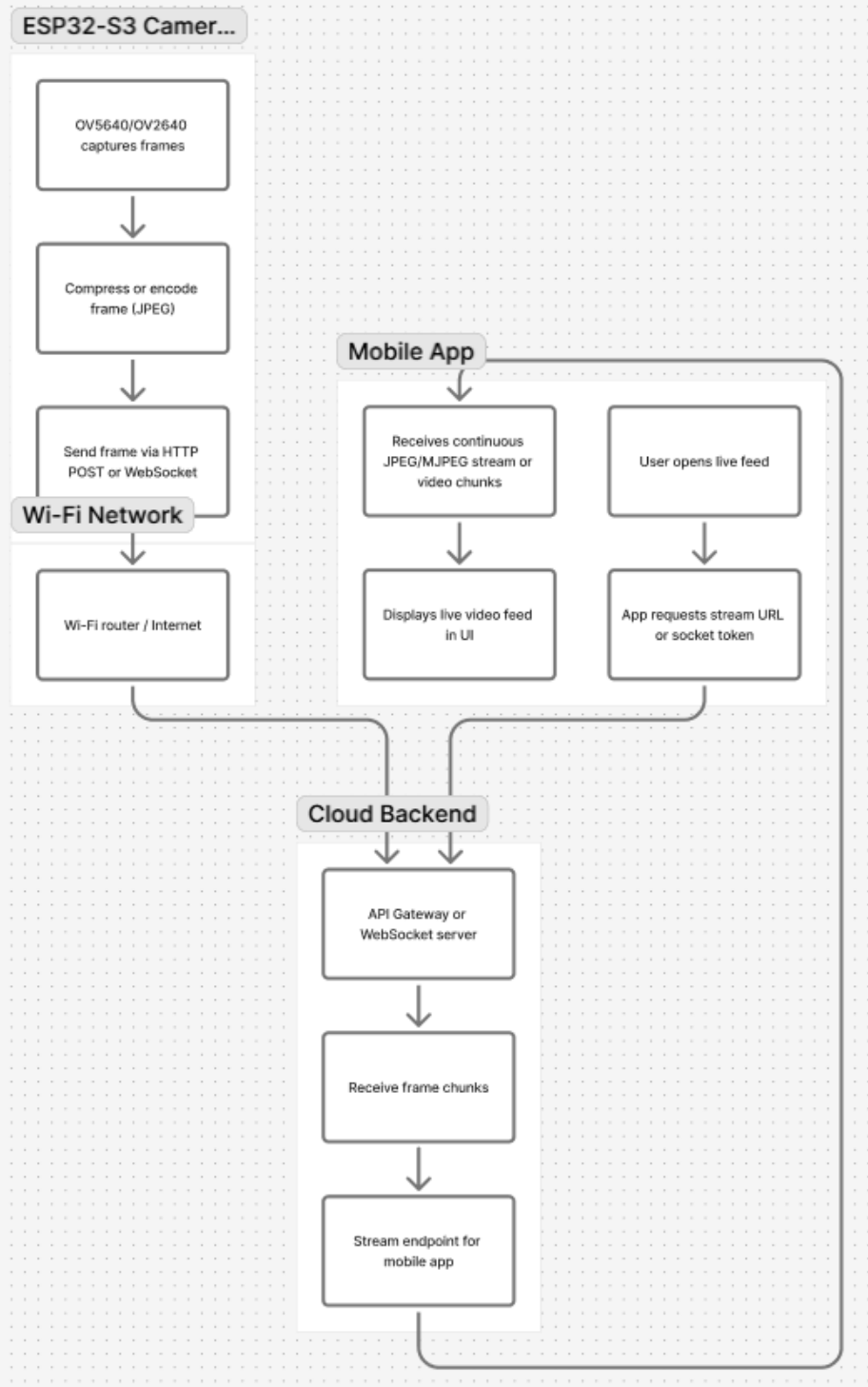


Figure 7.1.c: Live Stream Workflow

There is also a microphone functionality for the user to speak through the feeder. The user will tap on a microphone icon on the mobile application which allows them to speak into their phone. Once they do, the microphone will send a command to the ESP32 to ready it and will then start capturing audio and sending it to the speaker. The workflow is shown below.

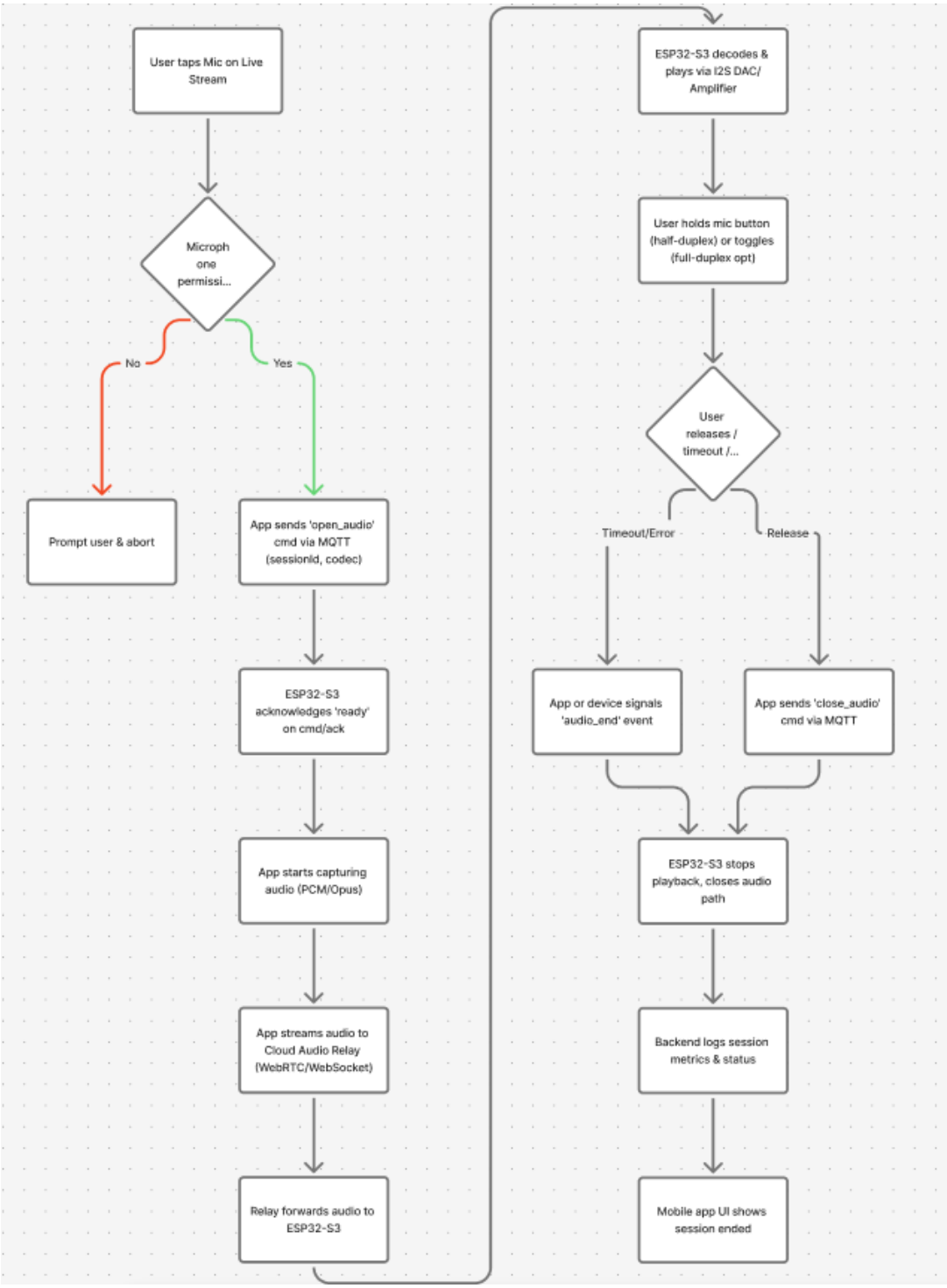


Figure 7.1.d: Microphone Workflow

7.2 Mobile Application

The mobile application is tasked with hosting the user interface such that the user can properly use the dog feeder. This means that it must have several capabilities. First, it must be a user-friendly interface such that the user can use it with a clear understanding. This means that all the interfaces and labels must be accurate and easy to identify. Second, it must be capable of handling the input and output of the necessary data such as the schedules, users, feeding logs, devices, and so forth. Third, it must be capable of communicating to both the MCU and the machine learning algorithm. Without this, the project is obsolete and cannot function.

The mobile application will be constructed using flutter as the primary programming language and will have a backend hosted by firebase. There will be a Realtime Database hosted by firebase which will be communicated to by flutter. We will have it so that the data is not very nested due to the performance costs of doing so with firebase. We will need to store users, schedules, devices, and logs in the database. We first need to configure the flutter app in such a way that it has a backend to communicate with. Once the app is configured, we can use functions from firebase to send information to and from the mobile app and the database. This is how the app works on a basic level.

Our mobile app needs to have several pages and functionalities. The first page is the landing page where you must login or register. We do this so that users can save the devices and schedules attached to the account, ensuring there is no data loss. When a user registers for an account, they input their full name, email address, and password. This gets sent to the database and can then be used to authenticate the user in the login process.

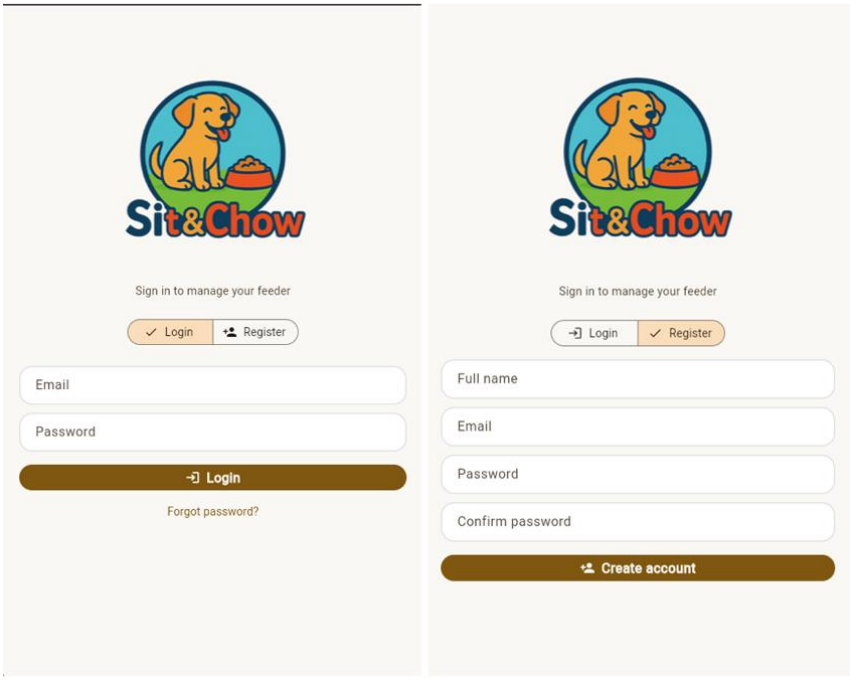


Figure 7.2.a: Login and Register Page

Once we have logged into the account, we land on the schedule page. Here we will have an app bar on the bottom and top of the screen. The top bar will have the name of the tab, and only on the schedule page will it have an icon to show feeding history. This allows the user to check to see if the feed was successfully accomplished or not. The bottom bar will store all of the tabs: schedule, devices, camera, and settings. We will be able to maneuver between all of the tabs, establishing full functionality of the mobile app.

On the schedule page we will have a quick feed option where the user will have the capability of choosing a device that has been connected to and then immediately dispending the desired amount, bypassing the machine learning algorithm. This allows the user to have quick and easy access to the dispensing mechanism should they need it. Then, there will be a schedules collection underneath said quick feed option. This is where all the scheduled feeds will be displayed. This lets the user easily see what feeds will occur.

These feeds will have an edit option where the user can do several things. The first is to reassign devices, allowing certain feeds to occur on certain devices. The second is to edit the feed itself, this lets the user change the time, size, and scheduled days of the feed. The third option is to delete the scheduled feed entirely. The last part of the schedule page is the add schedule function. This lets the user set the feeding time, the feeding size, the days in which the feed occurs, and the devices the feed applies to. There is also a link device option that allows the user to quickly link a new device to the feed.

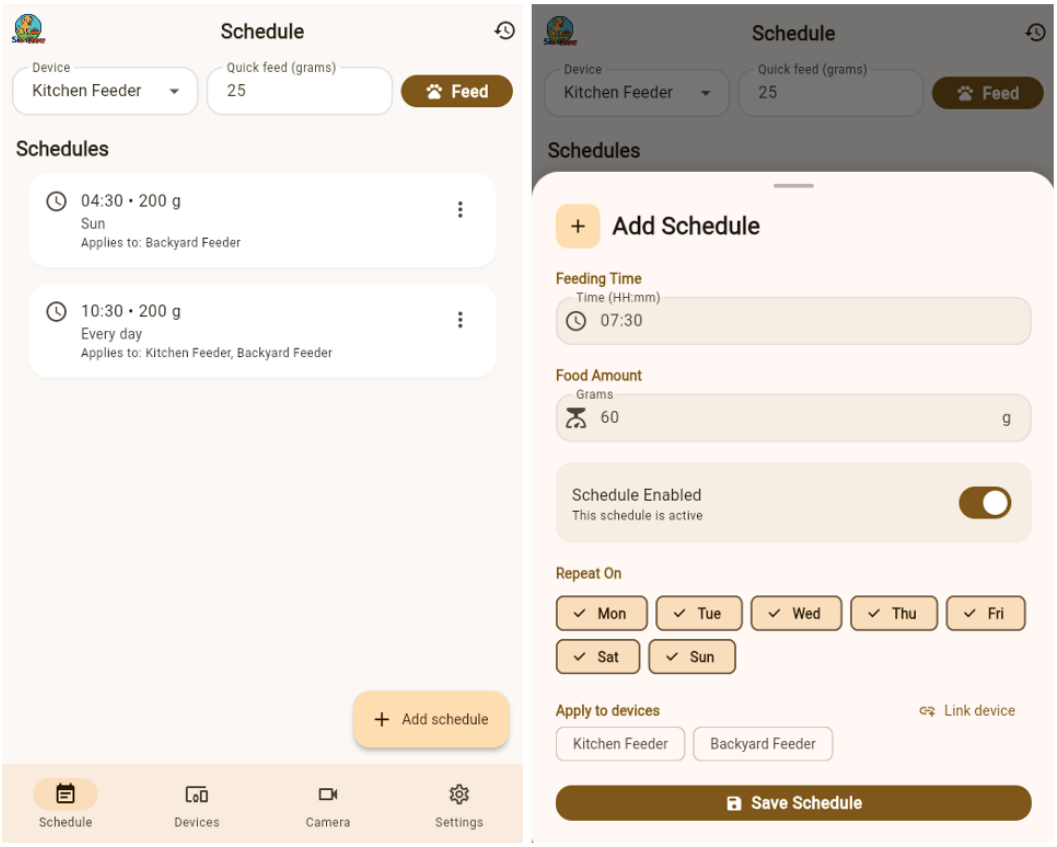


Figure 7.2.b: Schedule Page and Add Schedule Widget

As previously stated, the schedule page will have a history icon which the user can use to check the history of feeds. What this shows is the success rate of the machine learning algorithm. How this will work is once we arrive at the scheduled feeding time, the camera will activate and produce a live feed. This live feed will be fed to the machine learning algorithm where it can detect if the dog is in the sitting position. If after a certain amount of time the algorithm has not detected a sitting position, the feeder will dispense the food and mark a failure in the log. We dispense the food so as not to starve the dog if the algorithm is not working. If the algorithm does detect a sitting position, then the food will dispense immediately and then mark a success in the log. The user can then see the previous feeds in the log and determine if the dog needs more training.

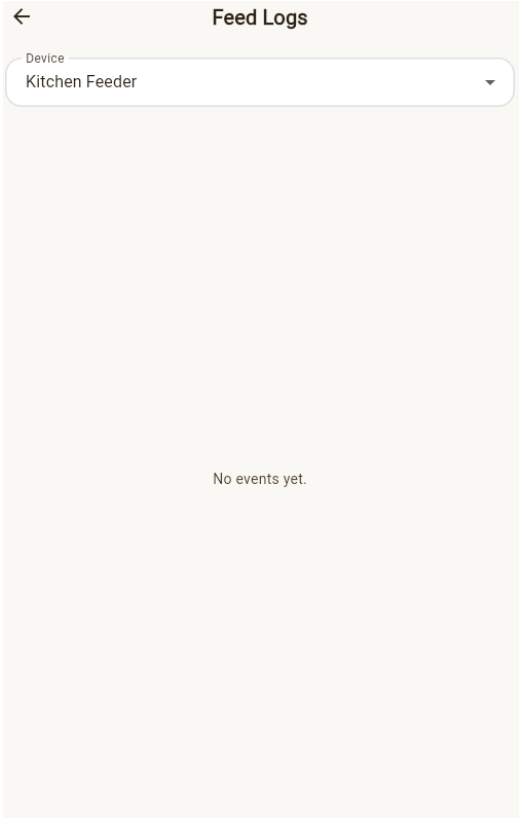


Figure 7.2.c: Feed Log Page

There are also aspects to discuss on the device page. This is where the user can establish a connection to a new device or configure the connection of existing devices. The user will have a collection of devices displayed to them and have the capability of editing or deleting the devices. There will be a link device option in the bottom right of the screen so the user can add a new device. The user will input the Device ID and can optionally assign a name to the device. The Device ID is currently a random number that can be used as a check by the device itself, allowing the app to connect to the device with the ID. Should we desire to pursue this project as a startup, we will likely either use a randomly generated QR code or a randomly generated ID and give it to the user along with the device to connect to it. Currently, the Device ID is there for the sake of proving the concept.

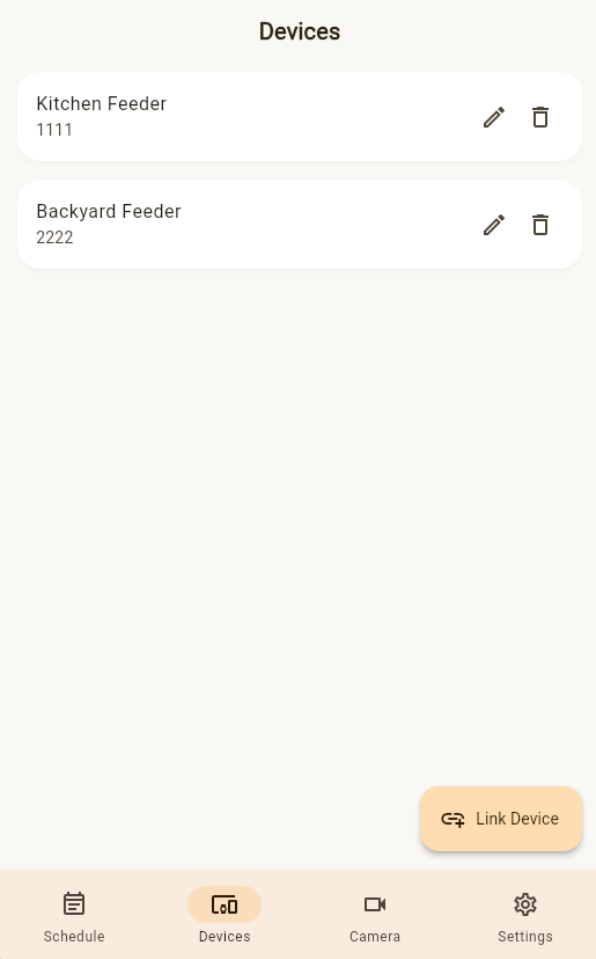


Figure 7.2.d: Device Page

We will also have a camera page. This is where the live feed of the camera will be displayed. Note that we have an additional button in the center of the page that the user must click to view the live feed. This is so that the user does not have to load the live feed every time they click the camera tab. This is to provide extra security for cases where the user accidentally clicks the camera tab but does not truly want to see the live feed. Once we click the button, we enter the live feed. There will also be a microphone button on the live feed that allows the user to speak through the phone to the dog through the feeder. This can help to get the dog to sit as the user can command them through the app.

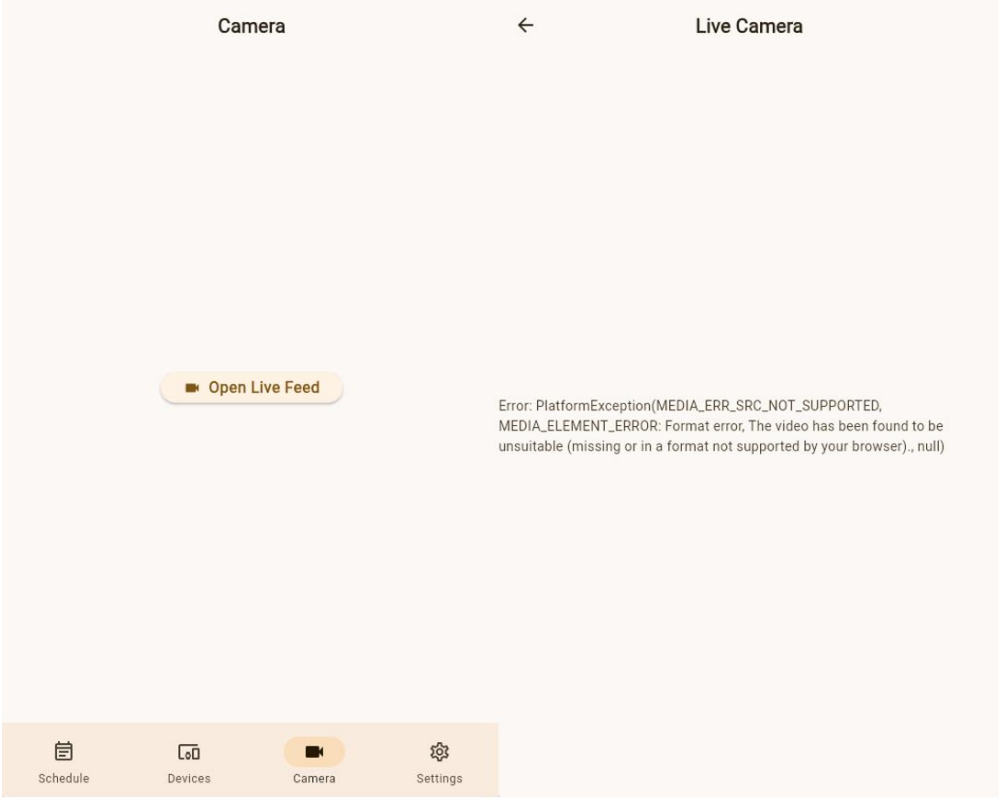


Figure 7.2.e: Camera and Live Camera Page

Lastly, on the mobile application we have the settings page. This is where the user can make adjustments to their account. They can see their name and email address on this page and have the option to edit their account name. They will have the option to verify their email address, which for our purposes is not of the utmost importance, but the option is there. They will also have the option to reset their password should they wish to change it. Note that they can also change their password on the login page if they forget their password while logging in. They can also choose to turn on notifications on this page. These notifications are sent when a scheduled feed occurs or when low food is detected. The last function of this page is the logout function which lets the user log out of their account and go back to the landing page.

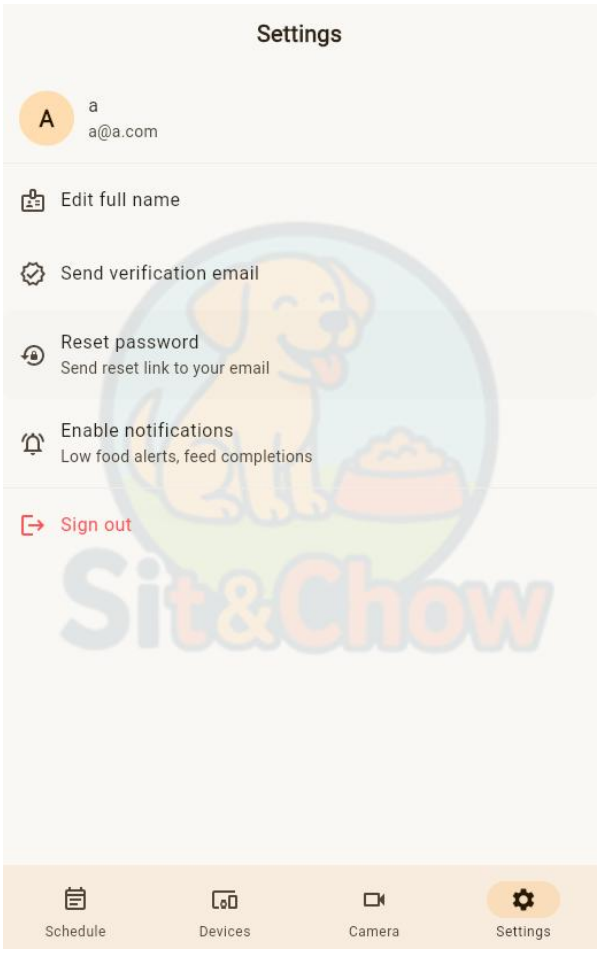


Figure 7.2.f: Settings Page

7.3 Machine Learning Algorithm

The detection model is responsible for determining if the dog is sitting in front of the feeder before dispensing food. The software performs real-time image processing, object detection, posture classification, and decision logic to confirm, with sufficient confidence, the dog's position and posture before triggering the feeding mechanism.

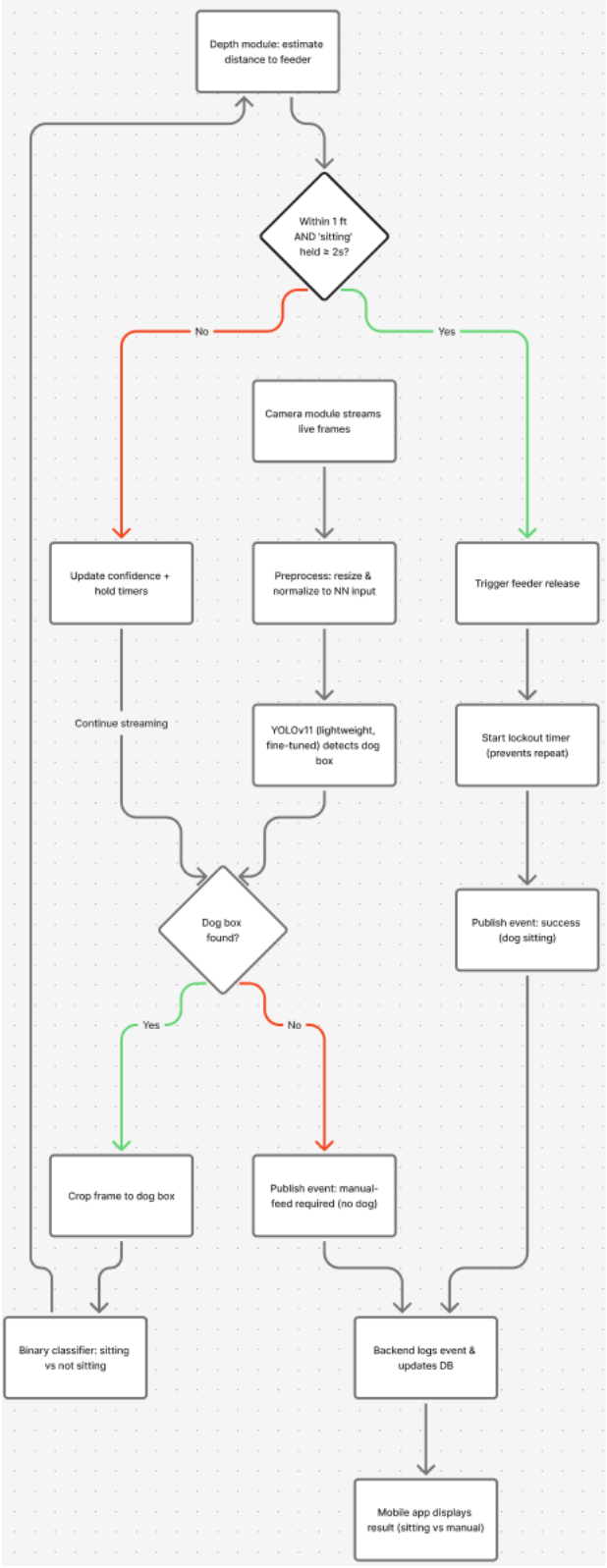


Figure 7.3: Detection Algorithm Flowchart

The model operates in five main stages: getting the image, preprocessing, decision logic, and event publishing. Live frames are streamed from the camera module and are resized and normalized to the neural network's input dimensions. A lightweight fine-tuned YOLOv11 model will identify the presence and location of a dog in the frame with a box. The frame will be cropped to the box and passed into a binary classification model trained to distinguish "sitting" and "not sitting," and a depth module will confirm that the dog is close enough to the feeder.

The binary classification algorithm will be trained with a large dataset of dogs sitting and not sitting and deployed on Firebase. If the algorithm maintains a confidence score and applies hold-and-release thresholds. For example, if the dog is within 1 foot and continuously classified as "sitting" for 2s of the feeder, then the feeding mechanism will release, and a lockout timer prevents the dispenser from repeating. Whether or not the dog is sitting, an event message will be sent to the mobile application that contains if the dog was sitting or if the dog had to be manually fed.

Chapter 8 – System Fabrication/Prototype Construction

8.1 Battery Backup

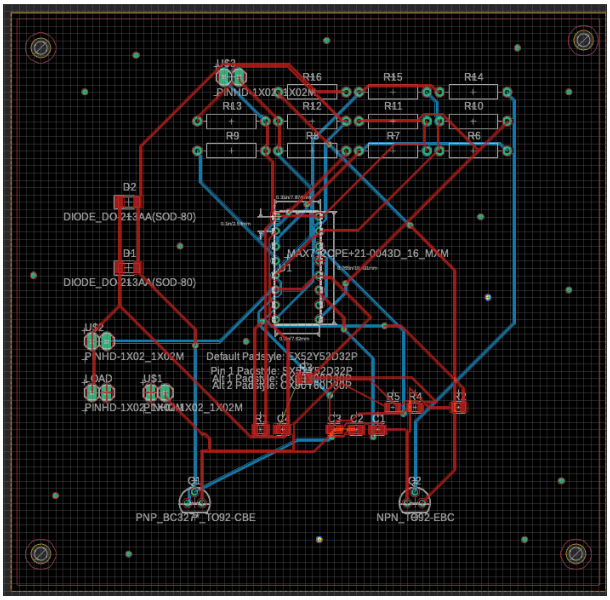


Figure 8.1.a: Battery Backup PCB

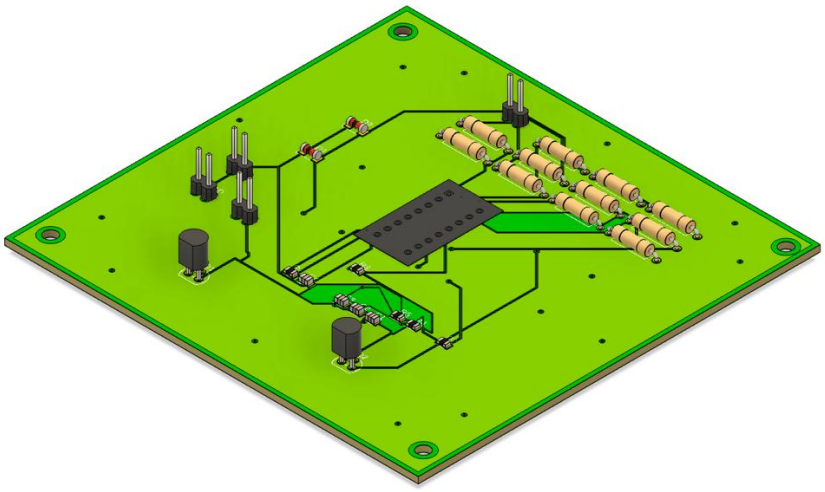


Figure 8.1.b: Battery Backup 3D model

8.2 Motor Driver

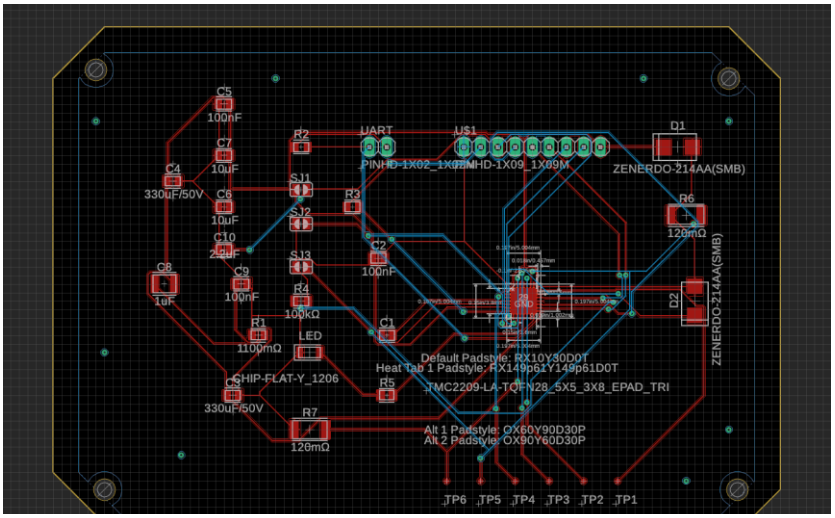


Figure 8.2.a: Motor Driver PCB

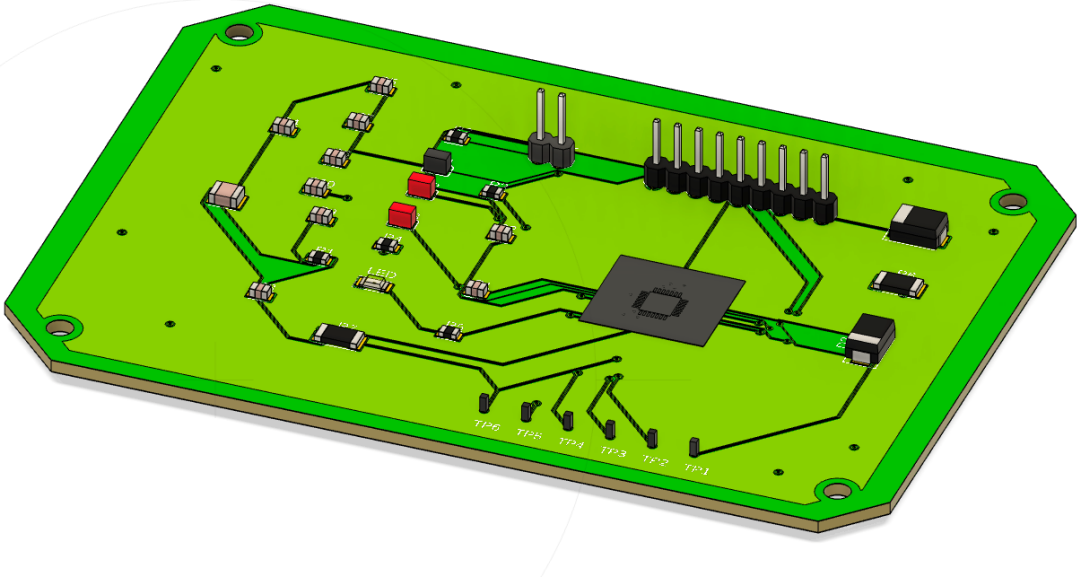


Figure 8.2.b: 3D Motor Driver Board

Chapter 9 – System Testing and Evaluation

Chapter 10 – Administrative Content

10.1 Budget

For our senior design project, developing a smart dog feeder requires a carefully planned budget to ensure both functionality and affordability. Our project budget is capped at $500. As electrical and computer engineering students, we must allocate funds for essential components such as microcontrollers, sensors, actuators, power supply units, and communication modules, which form the core of the system. Additional costs include materials for the feeder structure, such as food storage containers and dispensing mechanisms, as well as PCBs, wiring, and protective enclosures to ensure safety and durability. Software development tools and cloud integration services may also require minor licensing or subscription fees. To stay within a reasonable student budget, we will prioritize cost-effective parts while maintaining quality and reliability, aiming to balance innovation with financial responsibility. This budget not only outlines the material and component costs but also emphasizes the importance of resource management as part of the engineering design process.

10.2 Bill of Materials

Table 10.2: Bill of Materials

Component	Part name	Quantity	Unit price	Total price
Electrical Engineer (EE)				
Servo/Stepper Motor	Nema 17 Stepper Motor	1	$3.33	
Motor Driver	TMC2209	1	$6.995	
LEDs	SML-512PWT86A	1	$0.52	
Power Supply (12V adapter)	12V 3A AC to DC Adapter	1	$9.99	
Power Jack	CUI PJ-102A	1	$0.65	
Camera	OV5640	1	$13.76	
5V Voltage Regulator	TPS563200DDCR	1		
3.3V Voltage Regulator	TPS564257DRLR	1		

Speaker	Adafruit Speaker - 3" Diameter - 4 Ohm 3 Watt	1	$1.95	
Amplifier	Adafruit I2S 3W Class D Amplifier Breakout - MAX98357A	1	$5.95	
Button	TS02-66-70-BK-160-LCR-D	1	0.10	
Ultrasonic Sensor	RCWL-1601	1	$3.95	
Depth Module	TMF8701 ToF	1	$13	
Computer Engineer (CpE)				
Microcontroller	ESP32-S3-WROOM-2-N32R8V	1	$10.67	
EE & CpE Collaboration				
Mechanical Assets (Casing, feeding mechanism, etc.)			~$75	~$75
Breadboards	4PCS Breadboards Kit	1		
Wires	ELEGOO 120pcs Multicolored Dupont Wire	1		
Tapes	Electrical Tape	1		
Total Cost				
Electrical Engineer	$43.66			
Computer Engineer	$6.56			
EE & CpE Collaboration	$90.83			
Grand Total	$141.05			

10.3 Milestones

10.3.1 Senior Design 1 Milestones

Table 10.3.1: Senior Design 1 Milestones

Milestone Number	Milestone Description	Start Date	Anticipated End Date	Dependencies & Requirements

1	Group Finalization	08/19/25	08/26/25	Organizing/Deciding our Final Group Members
2	Project Idea Finalization	08/20/25	08/28/25	Research EE, CPE. Decide project idea to do
3	Design specifications	08/20/25	09/04/25	Finish D&C, obtain Dr. Saleem Sahawneh meeting.
4	Divide and conquer Document	08/27/25	09/05/25	Project overview and Organization (First 10 Pages)
5	Midterm Milestone Report	08/27/25	10/30/25	First 60 pages of our report.
6	Midterm Report Revision	10/30/25	11/07/25	Revision/Correction of Report
7	Project Report (120 Pages)	08/27/25	11/12/25	Full 120 Pages of the Report
8	Final Project Report	08/27/25	11/25/25	Fully revised Final report (Full 120 Pages)
9	Mini Demo Video	09/22/25	11/25/25	Demonstration of Fully Working Product
10	Project Design	11/25/25	12/06/25	Individual system design, Breadboard Prototyping, PCB ordering

10.3.2 Senior Design 2 Milestones

Table 10.3.2: Senior Design 2 Milestones

Milestone Number	Milestone Description	Start Date	Anticipated End Date	Dependencies & Requirements
1	PCB Testing	01/16/26	01/31/26	Continue work on PCB and research
2	System Testing	02/01/26	03/01/26	Check for potential mistakes
3	Project Demo	03/02/26	03/16/26	Putting devices together
4	Document	01/16/26	04/04/26	Finalize document
5	Practice Final Presentation	04/05/26	04/12/26	Practice and ready for presentation

6	Final Presentation	TBD	TBD	TBD

10.3.3 Schedule

Table 10.3.3: Schedule

Week	Task	Responsibility
08/28/25	-Finish D&C document -Prepare D&C meeting	Natalie Nguyen Maria Tran Andres Arzola Andrew Balmes
09/09/25	-Research EE, hardware -Select and acquire equipment: Servo, Sensors, etc	Natalie Nguyen Andrew Balmes
	-Research CpE, software -Select and acquire equipments	Maria Tran Andres Arzola
09/22/25	-Draft hardware block diagram -Review with advisor	Natalie Nguyen Andrew Balmes
	-Draft software architecture - Review with advisor	Maria Tran Andres Arzola
09/29/25	-Select and order motors, sensors, power supply -Approve component list, procurement	Natalie Nguyen Andrew Balmes
	-Select microcontroller/embedded platform -Build the mobile app -Approve component list, procurement	Maria Tran Andres Arzola
10/06/25	-Build motor control circuit, set up sensors, test power distribution -Integrate hardware + firmware for initial protype -Basic prototype that dispenses food on command	Natalie Nguyen Andrew Balmes

	-Write basic firmware -Integrate hardware + firmware for initial protype -Basic prototype that dispenses food on command	Maria Tran Andres Arzola
10/13/25	-Testing and debugging PCB, motor module, power module, ultrasonic sensor - Improve hardware reliability (safety, noise filtering, enclosure wiring). - Implement wireless communication (WiFi/Bluetooth).	Natalie Nguyen
	-Testing and debugging mobile app - Improve hardware reliability (safety, noise filtering, enclosure wiring). - Implement wireless communication (WiFi/Bluetooth).	Andres Arzola
	-Testing and debugging button, camera - Develop mobile/web app for scheduling and monitoring. - Implement wireless communication (WiFi/Bluetooth).	Maria Tran
	-Testing and debugging LEDs, speaker - Develop mobile/web app for scheduling and monitoring. - Implement wireless communication (WiFi/Bluetooth).	Andrew Balmes
10/20/25	-Midterm report – 60 pages -Test app scheduling and features -Demo -Conduct system integration testing	Natalie Nguyen Maria Tran Andres Arzola Andrew Balmes
10/31/25	-Revision -Prepare for presentation -Collect test data	Natalie Nguyen Maria Tran Andres Arzola

		Andrew Balmes
11/13/25	-Final Report -Mini Demo Video	Natalie Nguyen Maria Tran Andres Arzola Andrew Balmes

Appendices

Appendix A – References

[1] "PETKIT," YumShare Solo Automatic Feeder with Camera, 2023. [Online]. Available: https://petkit.com/products/yumshare-solo-with-camera.

[2] G. Jocher and J. Qiu, "Ultralytics," Ultralytics, 2024. [Online]. Available: https://docs.ultralytics.com/models/yolo11/.

[3] G. Jocher, "Ultralytics," 2020. [Online]. Available: https://docs.ultralytics.com/models/yolov5/.

[4] M. Radwan, I. Martinez, B. Y. J. Cyrille, F. Harte and M. David, "Autodesk Instructables," 26 12 2023. [Online]. Available: https://www.instructables.com/DIY-Pet-Feeder-and-Entertainer-Robot/.

[5] M. Hamoud, "SCRIBD," 2025. [Online]. Available: https://www.scribd.com/document/466378656/DCMI.

[6] STMicroelectronics, "ST," 2025. [Online]. Available: https://www.st.com/resource/en/product_training/STM32L4-Peripheral-Digital%20Camera%20Interface%20(DCMI).pdf.

[7] "ARDUINOSTORE," Arduino, 2024. [Online]. Available: https://store-usa.arduino.cc/products/arduino-uno-rev3?srsltid=AfmBOor5inf8t38-eZyog3-mSaj8grSfcmEOqBT8wB99sR5cMM3nMlpb.

[8] "Microchip," Microchip Technology Inc, 2025. [Online]. Available: https://www.microchip.com/en-us/product/atmega328p#Documentation.

[9] "Mouser," 2025. [Online]. Available: https://www.mouser.com/datasheet/2/830/doc8161-2490229.pdf?srsltid=AfmBOoqEKug7B219di9A3pmooomjrN0oBlcq4AgtY5vH6kMpkGLjP0d3.

[10] L. Jackson, "ArduCam," 25 06 2021. [Online]. Available: https://blog.arducam.com/stm32-cameras-that-support-dcmi/.

[11] "Raspberry Pi," [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

[12] N. Contino, "Raspberry Pi," 30 10 2024. [Online]. Available: https://www.raspberrypi.com/news/raspberry-pi-product-series-explained/.

[13] "Espressif," Espressif, [Online]. Available: https://www.espressif.com/en/products/socs/esp32.

[14] S. WARD-FOXTON, "EDN," 30 01 2025. [Online]. Available: https://www.edn.com/top-10-edge-ai-chips/.

[15] "Espressif," [Online]. Available: https://documentation.espressif.com/esp32-wroom-32_datasheet_en.pdf?utm_source=chatgpt.com.

[16] "Components101," 31 10 2022. [Online]. Available: https://components101.com/modules/esp32-cam-camera-module.

[17] "Espressif," [Online]. Available: https://documentation.espressif.com/esp32-s3_datasheet_en.pdf?utm_source=chatgpt.com.

[18] "Lenovo," Lenovo, [Online]. Available: https://www.lenovo.com/us/en/glossary/what-is-bluetooth/.

[19] "Wikipedia," 20 10 2025. [Online]. Available: https://en.wikipedia.org/wiki/Zigbee.

[20] "CSA," [Online]. Available: https://csa-iot.org/all-solutions/zigbee/.

[21] "Cisco," [Online]. Available: https://www.cisco.com/c/en/us/solutions/internet-of-things/lorawan-solution.html.

[22] "Wikipedia," 21 10 2025. [Online]. Available: https://en.wikipedia.org/wiki/Wi-Fi.

[23] "Sipeed," [Online]. Available: https://tang.sipeed.com/en/hardware-overview/ov2640-camera/.

[24] "Uctronic," Omnivision, 28 02 2006. [Online]. Available: https://www.uctronics.com/download/cam_module/OV2640DS.pdf.

[25] "Sparkfun," Omnivision, 26 05 2011. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/OV5640_datasheet.pdf.

[26] "Kamani," 10 12 2017. [Online]. Available: https://download.kamami.pl/p1180726-OV5640_Camera_Board_%28B%29_User_Manual_EN.pdf.

[27] "Realtek," 07 02 2025. [Online]. Available: https://www.realtek.com/images/ar/2024_Annual_Report_.pdf.

[28] "Robu," Realtek, 03 03 2023. [Online]. Available: https://robu.in/wp-content/uploads/2023/03/Realtek-AMB82-Mini-USER-MANUAL.pdf.

[29] "Polulu Robotics & Electronics," Stepper Motor: Bipolar, 200 Steps/Rev, 42×38mm, 2.8V, 1.7 A/Phase, [Online]. Available: https://www.pololu.com/product/2267.

[30] "Polulu Robotics & Electronics," Stepper Motor: Unipolar/Bipolar, 200 Steps/Rev, 42×48mm, 4V, 1.2 A/Phase, [Online]. Available: https://www.pololu.com/product/1200.

[31] "PBC Linear," PBC Linear, [Online]. Available: https://media.pbclinear.com/pdfs/pbc-linear-data-sheets/data-sheet-stepper-motor-support.pdf.

[32] "ESI Motion," ESI Motion, 08 07 2024. [Online]. Available: https://esimotion.com/blogs/news/key-differences-between-dc-and-stepper-motors.

[33] "SINBAD MOTOR," Dongguan Sinbad Motor Co., 17 04 2025. [Online]. Available: https://www.sinbadmotor.com/news/automatic-pet-feeders-how-drive-systems-and-motor-selection-simplify-pet-feeding/.

[34] "Texas Instruments," Texas Instrument, 07 2014. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv8825.pdf.

[35] "Analog," Analog Device, 16 02 2023. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/tmc2209_datasheet_rev1.09.pdf.

[36] P. D. Reynolds, "Xecor," 18 02 2024. [Online]. Available: https://www.xecor.com/blog/tmc2209-pinout-datasheet-alternatives-uses.

[37] "NLFX PROFESSIONAL," 22 07 2025. [Online]. Available: https://www.nlfxpro.com/blog2/understanding-speaker-specs-ohms-watts-sensitivity-explained-simply/.

[38] O. Jørgensen, "DYNAUDIO," DYNAUDIO, 07 2023. [Online]. Available: https://dynaudio.com/magazine/2023/july/impedance-ask-the-expert.

[39] "FDB," FDB Audio Manufacture Co., 12 06 2025. [Online]. Available: https://www.fdb-proaudio.com/What-s-the-Difference-Between-a-Mid-Range-Speaker-and-a-Full-Range-Speaker-id46657306.html.

[40] "ARENDAL," Arendal, [Online]. Available: https://arendalsound.com/guide/understanding-speaker-impedance-and-its-impact-on-sound-quality/.

[41] "Arylic," Arylic, 07 09 2023. [Online]. Available: https://www.arylic.com/es/blogs/news/differences-between-4-ohm-speaker-and-8-ohm-speaker.

[42] "SONOS," Sonos, [Online]. Available: https://www.sonos.com/en-sa/blog/guide-to-speaker-impedance-and-ohms.

[43] "MISCO," MISCO, 17 12 2024. [Online]. Available: https://blog.miscospeakers.com/how-speaker-wattage-distance-sensitivity-impact-loudness.

[44] M. McDonough, "Sweetwater," 22 05 2022. [Online]. Available: https://www.sweetwater.com/insync/importance-wattage-choosing-pa-speakers/.

[45] "Adafruit," Adafruit, [Online]. Available: https://www.adafruit.com/product/1314.

[46] "Sweetwater," 01 07 2024. [Online]. Available: https://www.sweetwater.com/insync/amplifier/.

[47] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Power_amplifier_classes?utm_source=chatgpt.com.

[48] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Class-D_amplifier.

[49] "Ooberpad," 24 02 2022. [Online]. Available: https://www.ooberpad.com/blogs/audio-video-tips/what-are-the-differences-between-class-a-ab-and-class-d-amplifiers-explained.

[50] "Electronics Technician Training," 28 07 2021. [Online]. Available: https://www.etcourse.com/news-blog/difference-between-class-b-ab-and-c-amplifiers.

[51] B. Pomerantz, "Crutchfield," [Online]. Available: https://www.crutchfield.com/S-IMu2CKByIvD/learn/which-amplifier-class-is-best.html.

[52] S. Munz and G. DellaSala, "Audioholics," 14 09 2018. [Online]. Available: https://www.audioholics.com/audio-amplifier/amplifier-classes.

[53] "Audioholics," [Online]. Available: https://audioholics.co.za/decoding-amplifiers-key-factors-types-advantages-drawbacks/.

[54] "Diode," [Online]. Available: https://www.diodes.com/part/view/PAM8403.

[55] "Analog," [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/max98357a-max98357b.pdf.

[56] "Adafruit," Adafruit, [Online]. Available: https://www.adafruit.com/product/3006.

[57] "Energy Star," [Online]. Available: https://www.energystar.gov/products/learn-about-led-lighting.

[58] "U.S. DEPARTMENT of ENERGY," [Online]. Available: https://www.energy.gov/energysaver/led-lighting.

[59] M. Thakur. [Online]. Available: https://www.educba.com/advantages-and-disadvantages-of-led-lighting/.

[60] "Candor Cicuit Board," 12 08 2022. [Online]. Available: https://www.candorind.com/surface-mount-vs-through-hole/.

[61] "Digikey," Digikey, [Online]. Available: https://www.digikey.com/en/products/detail/rohm-semiconductor/SML-512PWT86A/2337212.

[62] "Omron," [Online]. Available: https://components.omron.com/us-en/products/switches/tactile-switches/tactile-switch_features.

[63] "ONPOW," 06 05 2023. [Online]. Available: https://www.onpowbutton.com/news/metal-push-button-switch-a-comprehensive-guide-on-benefits-and-applications/.

[64] "GQEM," 16 04 2025. [Online]. Available: https://www.gqele.com/metal-push-button.html.

[65] "Digikey," Digikey, [Online]. Available: https://www.digikey.com/en/products/detail/same-sky-formerly-cui-devices/TS02-66-70-BK-160-LCR-D/15634243.

[66] "Adafruit," [Online]. Available: https://www.adafruit.com/product/4007?srsltid=AfmBOorhDl6aVqFoAWxCv2r21iTbt64qy5ODgcxVzO1FupobmlBq2rGl.

[67] "Sparkfun," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

[68] "Elecfreaks," [Online]. Available: https://www.elecfreaks.com/blog/post/hc-sr04-ultrasonic-module-user-guide.html.

[69] "Sharp," [Online]. Available: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf.

[70] "Pololu," [Online]. Available: https://www.pololu.com/product/136.

[71] "STMicroelectronics," 03 06 2023. [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l0x.pdf.

[72] "STMicroelectronics," 09 08 2024. [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l1x.pdf.

[73] "infineon," [Online]. Available: https://www.infineon.com/part/BGT60LTR11AIP.

[74] "infineon," 18 10 2024. [Online]. Available: https://www.infineon.com/assets/row/public/documents/24/49/infineon-bgt60ltr11saip-datasheet-en.pdf.

[75] R. Cong, "Jameco Electronics," [Online]. Available: https://www.jameco.com/Jameco/content/walladapter.html?srsltid=AfmBOopyY91IZO8UY16JBCdS_TkLPndwOSzI3M05Y1kWW4HOHR2Sepbt.

[76] P. Sagsveen, "Digikey," 30 05 2017. [Online]. Available: https://www.digikey.com/en/articles/picking-the-correct-wall-adapter.

[77] "Mouser," CUI devices, 30 10 2019. [Online]. Available: https://www.mouser.com/datasheet/2/670/pj_102a-1778834.pdf?srsltid=AfmBOoqM7q_i3zKcumwtkhhUMonnotoW_rQ-Gcg8f55BxaqPO7DSl9DA.

[78] "Digikey," Digikey, [Online]. Available: https://www.digikey.com/en/products/detail/same-sky-formerly-cui-devices/PJ-102A/275425.

[79] "Texas Instruments," Texas Instrument, 03 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2596.pdf.

[80] "Texas Instruments," Texas Instruments, 04 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps54202.pdf.

[81] "Texas Instruments," Texas Instruments, 09 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps63060.pdf.

[82] "Analog," Linear Technology, [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/1763fh.pdf.

[83] M. Shivanandhan, "Hackernoon," 20 12 2019. [Online]. Available: https://hackernoon.com/a-beginners-guide-to-aws-lightsail-pros-cons-and-use-cases-hrq32d2.

[84] "Cloudchipr," 10 12 2024. [Online]. Available: https://cloudchipr.com/blog/aws-lightsail.

[85] "Supabase," Supabase, [Online]. Available: https://supabase.com/docs.

[86] "Firebase," [Online]. Available: https://firebase.google.com/docs.

[87] "Firebase," [Online]. Available: https://firebase.google.com.

[88] "Wikipedia," 09 10 2025. [Online]. Available: https://en.wikipedia.org/wiki/React_Native.

[89] "ionicframework," [Online]. Available: https://ionicframework.com.

[90] "Github," [Online]. Available: https://github.com/flutter/flutter.

[91] "OpenCV," [Online]. Available: https://docs.opencv.org/4.x/.

[92] "PyTorch," [Online]. Available: https://docs.pytorch.org/docs/stable/.

[93] "TensorFlow," [Online]. Available: https://www.tensorflow.org.

[94] J. Terven and . D. Cordova-Esparza, "Arxiv," 02 04 2023. [Online]. Available: https://arxiv.org/abs/2304.00501.

[95] "OpenCV," [Online]. Available: https://docs.opencv.org.

[96] E. Nolasco, A. C. Aldea and J. Villaverde, "Dog Activity Recognition Using Convolutional Neural Network," 30 04 2025. [Online]. Available: https://www.mdpi.com/2673-4591/92/1/41.

[97] "GeaksforGeeks," 23 07 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/training-of-convolutional-neural-network-cnn-in-tensorflow/.

[98] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "Arxiv," 09 05 2016. [Online]. Available: https://arxiv.org/abs/1506.02640.

[99] W. G. Wong, "ElectronicDesign," 10 10 2014. [Online]. Available: https://www.electronicdesign.com/techxchange/article/55238406/edge-computing-and-tinyml?utm_source=chatgpt.com.

[100] "ONNX," [Online]. Available: https://onnx.ai.

[101] K. Sun and M. Dredze, "Arxiv," 13 08 2024. [Online]. Available: https://arxiv.org/html/2408.06663v1.

[102] S. Raman, R. Maskeliūnas and R. Damaševičius, "MDPI," 24 12 2021. [Online]. Available: https://www.mdpi.com/2073-431X/11/1/2.

[103] S. Heydari and Q. H. Mahmoud, "MDPI," 19 05 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/10/3191.

[104] J. Yao, S. Zhang, Y. Yao, F. Wang, J. Ma, J. Zhang, Y. Chu, L. Ji, K. Jia, T. Shen, A. Wu, F. Zhang, Z. Tan, K. Kuang, C. Wu, F. Wu, J. Zhou and H. Yang, "Arxiv," 11 11 2021. [Online]. Available: https://arxiv.org/abs/2111.06061.

[105] M. Threadgill and A. Gerstlauer, "Arxiv," 23 05 2024. [Online]. Available: https://arxiv.org/abs/2405.15079.

[106] H. Miao and F. X. Lin, "Github," 12 2022. [Online]. Available: https://hongyumiao.github.io/assets/sec22-nn-mcu.pdf.

[107] J. Karlin, "AceCloud," AceCloud, 15 09 2025. [Online]. Available: https://acecloud.ai/blog/high-performance-computing-with-cloud-gpus/.

Appendix B – Copyright Permission

Appendix C – Data Sheet

TMC2209 - Stepper Motor Driver | Adafruit

SML-512PWT86A - LED | Digikey

PJ-082BH - Power Jack | Digikey

TPS563200DDCR - 5V Voltage Regulator | Digikey

TPS564257DRLR - 3.3V Voltage Regulator | Digikey

Speaker - 3" Diameter - 4 Ohm 3 Watt - Speaker | Adafruit

Adafruit I2S 3W Class D Amplifier Breakout - MAX98357A | Adafruit

TS02-66-70-BK-160-LCR-D - Button | Digikey

Ultrasonic Distance Sensor - 3V or 5V - HC-SR04 compatible - RCWL-1601 | Adafruit

ESP32-WROOM-32

ESP32-S3 Series

Appendix D – Software Code

Appendix E – Application of ChatGPT and Other Similar Platforms

[1a] ***Case Study 1 Prompt***: *"What do we need for a dog feeder project?"*

[1b] ***Case Study 1 ChatGPT Outcome****: "The dog feeder project integrates mechanical, electrical, and software systems to automate pet feeding efficiently. Mechanically, it includes a food storage hopper, a dispensing mechanism such as a servo or stepper motor-driven gate, and a feeding bowl. These components must be food-safe, easy to clean, and designed to prevent clogging.*

The electrical system provides stable power using a 12V wall adapter and voltage regulators to supply 5V and 3.3V for control electronics. Power management components such as a power jack, switch, and fuse ensure safety and continuous operation. The control system, built around a microcontroller like the ESP32 or Arduino Uno, manages timing, sensors, and motor movement. It may include an RTC module for scheduling, an LCD display, and input buttons for user control.

Sensors play a key role in automation, monitoring food levels and portion sizes. Load cells with HX711 amplifiers provide accurate weight measurement, while ultrasonic or ToF sensors detect food levels in the hopper. Optional IR or capacitive sensors can confirm food presence in the bowl.

The software coordinates the feeding logic, triggering the motor at specific times, checking food weight, and optionally sending notifications through Wi-Fi or Bluetooth. Using the ESP32 enables IoT features for remote control via apps like Blynk or Firebase. Overall, this system ensures accurate, reliable feeding, minimizes user effort, and enhances pet care through automation and smart monitoring."

[2a] ***Case Study 2 Prompt****: "What type of speaker should we use for this project?"*

[2b] ***Case Study 2 ChatGPT Outcome****: "The speaker in a dog feeder project serves as an audio alert system to notify the pet or owner when feeding occurs. For a simple design, an active buzzer is the most efficient choice. It produces a clear beep sound and only requires a digital signal from the microcontroller, making it easy to integrate and energy-efficient. This option is ideal when the feeder only needs to signal feeding times or confirm that food has been dispensed.*

For more advanced functionality, a small 8Ω, 1W speaker can be used with an audio playback module such as the DFPlayer Mini. This setup allows the system to play recorded sounds, such as a custom voice message or tone, to attract the dog during mealtime. Although it consumes slightly more power and requires additional wiring, it enhances user interaction and pet engagement.

Overall, the choice depends on the desired functionality. An active buzzer provides simplicity and reliability for basic alerts, while an 8Ω speaker with an audio module offers a more interactive and customizable feeding experience suitable for smart feeder systems."

[2c] ***Case Study 2 Google AI Outcome:*** *"For an automatic dog feeder, selecting the right speaker is crucial to ensure clear and audible feedback without adding unnecessary cost or complexity. A small, full-range dynamic speaker, typically with an impedance of 4 or 8 ohms and a low power rating of 3 to 5 watts, is the most suitable choice. This type of speaker utilizes a moving-coil mechanism, where an electrical audio signal passes through a voice coil and interacts with a magnetic field, causing a cone to vibrate and produce sound. Its primary advantages are its cost-effectiveness, widespread availability, and robust construction, making it perfect for the low-demand application of simple audio notifications. Unlike more specialized options, a dynamic speaker is easily compatible with standard audio amplifiers, including the MAX98357A chosen for this project.*

Other speaker types, such as a piezoelectric (piezo) speaker, are not recommended for this application. While piezo speakers are very efficient and resistant to overload, their frequency response is poor, producing a tinny or high-pitched sound that is inadequate for recorded voice messages. Furthermore, their high impedance and capacitive nature make them incompatible with standard amplifiers like the MAX98357A, which is designed for a traditional dynamic load. This incompatibility and inferior sound quality make the piezoelectric option an unsuitable choice for the dog feeder's audio system.

When matching the speaker to the chosen MAX98357A amplifier, two key electrical specifications must be considered: impedance and power rating. The MAX98357A is designed to drive speakers with an impedance of either 4 or 8 ohms. Opting for an 8-ohm speaker is generally safer, as it draws less current and reduces heat generation, though the amplifier is capable of driving a 4-ohm load. Regarding power rating, the speaker's power handling capacity should match or exceed the amplifier's maximum output of 3W. A speaker with a rating of 3 to 5 watts RMS is therefore an ideal and safe match, providing ample volume for a project like a dog feeder without risking damage. While a specific enclosure could slightly improve sound quality, for simple audio alerts, mounting a small 1.5-inch to 2-inch full-range dynamic speaker directly into the project casing is perfectly sufficient."

[3a] ***Case Study 3 Prompt****: "If I want to do the power source and supply for a automatic dog feeder. How can I do it, explain what I need, why, how to build it as a PCB?"*

[3b] ***Case Study 3 ChatGPT Outcome****: "The power system for an automatic dog feeder must safely deliver stable voltages to all components — motors, sensors, and control circuits. A 12 V DC wall adapter is commonly used as the main power source since it's reliable and easy to integrate. The 12 V rail can directly drive the motor or stepper driver, while buck converters reduce it to 5 V for the microcontroller, sensors, and display, and to 3.3 V for modules like Wi-Fi or Bluetooth. Using switching regulators ensures high efficiency and prevents overheating compared to linear regulators.*

Input protection is essential. A Schottky diode or MOSFET prevents reverse polarity, and a fuse or resettable PTC protects against overcurrent. Large electrolytic capacitors near the motor driver stabilize power when the motor starts, while ceramic decoupling capacitors close to the regulators reduce noise.

If backup power is desired, include a battery charging circuit (such as TP4056) and a protected Li-ion or NiMH pack. A power-path circuit allows automatic switching between wall and battery power.

To build this as a PCB, organize the circuit into blocks: input protection, voltage regulation, battery charging, and motor driver power. Use wide traces for high-current paths, a solid ground plane, and short loops for switching regulators. Add test points and labeled connectors for safety and ease of testing. This structure ensures the feeder operates reliably, even under variable load or during power interruptions."

[3c] ***Case Study 3 Google AI Outcome****: "An automated dog feeder requires a reliable and stable power supply to operate all its components, from the microcontroller and sensors to the motor and audio amplifier. Designing a custom Printed Circuit Board (PCB) for this purpose is an effective way to ensure a compact, integrated, and reliable power solution. This design will typically involve an external AC-to-DC adapter for main power, efficient DC-to-DC converters to regulate different voltage levels, and potentially a battery backup system for uninterrupted operation.*

For the power source, it is safest and simplest to use an external AC-to-DC wall adapter, which handles the high-voltage mains power externally and provides a lower, regulated DC voltage (e.g., 12V). This external adapter connects to a standard barrel jack or USB-C connector on the PCB, eliminating the need for complex and potentially dangerous high-voltage circuitry within the project enclosure. On the PCB itself, the 12V input from the adapter needs to be stepped down to power the various components. An efficient buck converter, such as one based on the LM2596 integrated circuit, is the ideal component for converting the 12V input to a stable 5V supply for the microcontroller, ultrasonic sensor, and the digital audio amplifier. For sensitive components like the amplifier, which can be susceptible to noise from a switching buck converter, a small, less-efficient linear regulator (e.g., AMS1117) can be used to further clean up the power signal to a required voltage, such as 3.3V, providing a quiet and stable supply.

To make the dog feeder resilient to power outages, a battery backup system can be integrated into the PCB design. This requires a rechargeable battery, such as a lithium-ion or LiPo cell, and a dedicated charging circuit to manage it safely. An IC like the TP4056 is a common choice for single-cell LiPo battery charging and protection. A simple yet effective automatic power-switching circuit can then be implemented, often using a pair of diodes, to prioritize power from the AC adapter when it is present and seamlessly switch to battery power when it is not. This ensures the device continues to function and provide scheduled feedings even when the main power is interrupted, safeguarding against unforeseen service disruptions.

Building this power supply on a custom PCB requires careful design practices to ensure reliability and performance. Good component placement is critical, with all high-current paths kept short and wide to minimize resistance and noise. Using a large, solid ground plane on one or more layers of the PCB provides a low-impedance return path for currents and helps with heat dissipation. Traces for high-current sections, particularly those powering the motor, must be wide enough to prevent overheating. Placing decoupling capacitors close to the power pins of each IC is essential for filtering out high-frequency noise. Once the schematic is finalized and the layout adheres to these design rules, the PCB can be fabricated by a manufacturer from the generated Gerber files, providing a professional and robust power solution for the project."

[4a] ***Case Study 4 Prompt:*** *"Please give me a list of technologies I can use to create a mobile app for an automatic dog feeder and describe why I would use each one."*

[4b] ***Case Study 4 ChatGPT Outcome:*** *"Here's a concise, engineering-oriented menu of proven technologies for a mobile app that controls an automatic dog feeder, plus why you'd pick each. I've grouped them by layer and added a few "stack recipes" at the end.*

Mobile frontend (app UI)

- *Flutter (Dart): One codebase for iOS/Android; fast UI iteration; great widgets; solid camera/network packages. Use if you want consistent look/feel and high velocity with minimal platform-specific code.*

- *React Native (JS/TS): Reuse web skills; rich ecosystem; easy to share logic with a web dashboard. Use if your team is JavaScript-heavy or you want maximum third-party library choice.*

- *Native (SwiftUI for iOS / Kotlin for Android): Best performance and platform APIs day-one. Use if you need tight OS integration (background tasks, BLE quirks) or top-tier UX polish.*

- *Ionic + Capacitor (Web + native shell): Ship quickly with web tech; easy to build admin dashboards too. Use if your team is primarily web developers and the app is UI-driven, not graphics-heavy.*

Backend & database

- *Firebase (Auth, Firestore/Realtime DB, Cloud Functions, Storage): Rapid backend with realtime data, push notifications, and built-in auth. Use for quick MVPs and low-ops maintenance.*

- *Supabase (Postgres, Auth, Row-Level Security, Edge Functions): SQL + realtime channels with strong access control. Use if you prefer relational schemas and SQL analytics down the road.*

- *AWS Amplify (Cognito, AppSync/GraphQL, DynamoDB, S3, Lambda): Opinionated AWS starter with strong scalability. Use if you want to stay "all-AWS" and need fine-grained control later.*

- *Railway/Fly.io/Vercel + Postgres/MongoDB: Simple deployment for custom Node/Go/Python backends. Use if you want custom APIs but easy ops.*

Realtime device control & telemetry

- *MQTT (e.g., Eclipse Mosquitto, EMQX, HiveMQ Cloud): Lightweight pub/sub ideal for ESP32. Use for commands ("dispense 20g now"), state ("hopper=low"), and schedules.*

- *WebSockets (via your backend or Pusher/Ably): Realtime app updates without polling. Use for live UI feedback when MQTT is device-side only.*

Video streaming (live camera)

- *WebRTC (Pion, LiveKit, Agora, Twilio): Low-latency, NAT-friendly. Use if you need near-real-time video from feeder to phone.*

- *RTMP/RTSP → HLS (Nginx-RTMP, Ant Media, Wowza, Cloudflare Stream): Simpler pipeline, higher latency. Use for "good enough" preview/monitoring where sub-second latency isn't required.*

Notifications & scheduling

- *FCM (Firebase Cloud Messaging) / APNs: Push alerts for "feeding complete," "low kibble," or motion detected. Use FCM to unify Android/iOS via one API.*

- *Cloud scheduler + Functions (Firebase Scheduled Functions / AWS EventBridge / Cron jobs): Server-side schedule source of truth. Use to ensure feeds occur even if the phone is offline.*

Authentication & access control

- *Firebase Auth / Supabase Auth / AWS Cognito: Social login + email, device-scoped roles. Use to bind devices to owners and share access with family members.*

- *OAuth 2.0 / OpenID Connect (Auth0): Enterprise-grade auth if you expect org logins or advanced policies.*

Storage & media

- *Cloud Storage (Firebase Storage / S3 / Supabase Storage): Store clips, images, firmware bins. Use signed URLs for secure, time-limited access.*

Analytics, logs, and quality

- *Crashlytics (Firebase) / Sentry: Track crashes in production.*

- *Google Analytics for Firebase / Amplitude: Understand feature usage (e.g., schedule adherence, manual overrides).*

- *OpenTelemetry + Log drains: Trace device→cloud→app flows for debugging.*

ML options (optional)

- *On-device (TensorFlow Lite / Core ML): Simple models (e.g., "dog present") without network. Use for privacy/latency.*

- *Backend inference (Cloud Functions / AWS Lambda / GPU service): Heavier models (pose, detection) with consistent results. Use when the phone/ESP32 shouldn't run ML.*

DevOps & testing

- *CI/CD (GitHub Actions, Codemagic for Flutter, EAS for Expo/RN): Automated builds, tests, and releases.*

- *Testing (Flutter test / Jest + Detox / XCTest + Espresso): Unit + UI tests for scheduling, auth, and command flows.*

- *IaC (Terraform / AWS CDK): Reproducible cloud infra as the project grows.*

Suggested "stack recipes"

1) Fastest MVP (balanced features, minimal ops)

- *Frontend: Flutter*

- *Backend: Firebase (Auth, Firestore, Functions, Storage)*

- *Realtime device control: MQTT (managed broker) + Cloud Function bridge to Firestore*

- *Video: WebRTC via LiveKit Cloud (or Agora)*

- *Notifications: FCM*

Why: Ship quickly; great realtime data; easy push; outsource the tricky video piece.

2) SQL-centric & policy-heavy

- *Frontend: React Native (TypeScript)*

- *Backend: Supabase (Postgres + RLS + Edge Functions)*

- *Realtime: Supabase Realtime channels for app; MQTT for device*

- *Video: WebRTC via LiveKit/Agora*

- *Notifications: FCM/APNs via server function*

Why: Strong relational model, row-level security for multi-user device sharing, and clean TypeScript tooling.

3) Enterprise-ready, long-term scale

- *Frontend: Native (SwiftUI/Kotlin) or React Native*

- *Backend: AWS (Cognito, AppSync GraphQL, DynamoDB/S3, Lambda)*

- *Realtime: MQTT via AWS IoT Core; GraphQL subscriptions to the app*
- *Video: Amazon Kinesis Video Streams (WebRTC) or managed provider*
- *Notifications: SNS + APNs/FCM*

Why: Fine-grained IAM, rock-solid scaling, and integrated IoT.

Helpful implementation notes

- *Keep schedules server-side; phone edits sync to server; server triggers MQTT at feed time → device ACK → push notification.*
- *Use device claims (userId, deviceId) inside auth tokens for secure topic names (e.g., users/{uid}/devices/{deviceId}) and enforce RLS/IAM.*
- *For WebRTC from ESP32-S3 cams, consider a gateway/bridge (small edge box or microservice) to handle signaling and transcoding; direct from microcontrollers is often brittle.*
- *Start with feature flags (remote config) to safely roll out "auto-detect dog" ML features.*

If you want, tell me your team's preferred language (Dart vs JS/TS) and constraints (budget, latency for video), and I'll pick one recipe and sketch the data models, API endpoints, and message topics."

[4c] ***Case Study 4 Google AI Outcome:*** *"That's an exciting project! Creating a mobile app for an automatic dog feeder involves several layers of technology, from the app itself to how it communicates with the physical feeder device.*

Here is a list of key technologies you can use for the mobile app, categorized by their function, and why they would be a good fit for this project:

Mobile App Development Frameworks (Frontend)

These frameworks determine the programming language and structure of the app that runs on the user's phone.

Technology	*Type*	*Why use it for a Dog Feeder App*
React Native	*Cross-Platform*	*Fast Development & Cross-Platform: Allows you to use a single JavaScript/TypeScript codebase to build apps for both iOS and Android. This significantly reduces development time and cost, which is ideal for a new product.*

Flutter (Dart)	*Cross-Platform*	*Excellent UI/UX & Single Codebase: Google's framework provides beautiful, highly customizable user interfaces that look and feel native on both platforms. It's great for real-time data visualization (e.g., food levels, feeding history).*
Kotlin (for Android) & Swift (for iOS)	*Native*	*Best Performance & Native Features: Using the official languages for each platform gives you the absolute best performance, stability, and direct access to native features like background tasks, advanced push notifications, and low-latency Bluetooth/Wi-Fi communication, which is crucial for an IoT device.*

Connectivity & Data Transfer (Backend/IoT)

Your app needs a way to securely send commands (like "Feed Now" or "Set Schedule") to the feeder and receive data (like "Food Low" or "Feeding Complete").

Technology	*Type*	*Why use it for a Dog Feeder App*
MQTT (Message Queuing Telemetry Transport)	*Communication Protocol*	*Lightweight & Efficient: This is the standard protocol for IoT devices. It's designed for unreliable networks and low bandwidth, using very little power and data, making it perfect for battery-powered or resource-limited feeder hardware. It's essential for instant commands and real-time status updates.*
Bluetooth Low Energy (BLE)	*Short-Range Communication*	*Initial Setup & Local Control: Great for the initial setup and provisioning of the feeder (connecting it to the user's home Wi-Fi). It can also be a reliable local control method if the home Wi-Fi goes down, allowing the app to communicate directly when the phone is nearby.*
Wi-Fi	*Local Communication*	*Remote Control: The primary connection for the feeder to the internet. It enables the user to control and monitor the feeder from anywhere in the world (via a cloud service).*

Cloud Infrastructure (Backend)

You need a secure, scalable place to manage user accounts, store feeding schedules, and route messages between the app and the feeder.

Technology	*Type*	*Why use it for a Dog Feeder App*
AWS IoT Core or Google Cloud IoT Core	*IoT Platform*	*Scalability & Managed Services: These platforms are specifically built for managing millions of connected devices. They handle secure device authentication, data ingestion, and routing, saving you significant time compared to building it all from scratch.*
Firebase (e.g., Realtime Database or Firestore)	*Backend as a Service (BaaS)*	*Real-time Data & Speed: A powerful, easy-to-use backend solution. Its real-time database is excellent for storing and syncing user data (feeding times, portions, pet profiles) instantly across the mobile app and the feeder device's cloud connection.*
Push Notification Services (e.g., Firebase Cloud Messaging/FCM or Apple Push Notification Service/APNS)	*Notification Service*	*Critical Alerts: Essential for sending non-time-critical notifications to the user's phone, such as "Food Level Low" or "Feeder Offline," even when the app is closed.*

Summary and Key Considerations

For a project like this, a popular and balanced approach is:

1. Mobile App: React Native or Flutter (for cost-effective, fast, cross-platform development).

2. Cloud/Backend: AWS IoT Core or Firebase (for reliable, scalable IoT management and real-time data sync).

3. Communication: MQTT over Wi-Fi (for remote control and status updates) + BLE (for local setup).

The choice between a cross-platform framework (like React Native/Flutter) and native development (Kotlin/Swift) usually comes down to budget and priority:

- *Cross-Platform: Cheaper and faster to launch.*
- *Native: Best performance and reliability, but higher initial cost and two separate codebases to maintain.*

Would you like me to elaborate on the hardware side of this project, such as what microcontroller and sensors would be used inside the automatic dog feeder itself?"

[5a] ***Case Study 5 Prompt****: "In my mobile application for an automatic dog feeder, I want to do several things. I want to be able to feed the dog immediately as well as have scheduled feeding times. I want to be able to connect to multiple dog feeders from the same app and set each feeding time individually. I want to have a login and register for the application. And I want to have a video feed of the dog. Assume I will be using flutter as the framework for the front end. Can you tell me how to construct the frontend for this mobile application?"*

[5b] ***Case Study 5 ChatGPT Response****: "awesome—here's a clean, production-minded Flutter frontend plan for your multi-feeder dog-feeder app (immediate feed, per-feeder schedules, auth, and live video). I'll assume: Flutter, Firebase Auth for login/registration, and a backend you control (can be Firebase + a small Node/Go service) that exposes REST/WebSocket endpoints.*

High-level architecture

- *State management: Riverpod (simple, testable, compile-safe)*
- *Routing: go_router with guarded routes (auth vs. app)*
- *Networking: dio (REST) + optional web_socket_channel (realtime)*
- *Realtime data: Firestore/RTDB or your own WebSocket (front-end adapts to either)*
- *Local cache: Hive (small, fast, device-scoped data & offline schedules)*
- *Video: HLS (video_player) for one-way streaming; WebRTC (flutter_webrtc) if you need low latency and 2-way talkback*
- *Push: FCM for "feeding executed", "hopper low", "camera motion"*
- *Analytics & crash: Firebase Analytics + Crashlytics*

Folder structure

```
lib/
  app.dart
  main.dart
  core/
    config/app_config.dart
    routing/app_router.dart
    theme/app_theme.dart
    utils/validators.dart
    errors/failures.dart
  data/
    models/
      user_profile.dart
      feeder.dart
      schedule.dart
      feed_action.dart
      video_stream_info.dart
    sources/
      remote/api_client.dart
      remote/auth_api.dart
      remote/feeder_api.dart
      remote/schedule_api.dart
      remote/video_api.dart
      local/hive_boxes.dart
      local/cache.dart
    repositories/
      auth_repository.dart
      feeder_repository.dart
      schedule_repository.dart
      video_repository.dart
  features/
    auth/
      controllers/auth_controller.dart
      views/login_page.dart
      views/register_page.dart
    feeders/
      controllers/feeders_controller.dart
      views/feeders_page.dart        // list + add/rename/remove
      views/feeder_detail_page.dart  // tabs: Dashboard | Schedule | Camera | Logs
      widgets/feeder_card.dart
    schedule/
      controllers/schedule_controller.dart
      views/schedule_editor_sheet.dart
      widgets/schedule_list.dart
    camera/
      controllers/camera_controller.dart
      views/camera_page.dart
```

dashboard/
controllers/dashboard_controller.dart
views/dashboard_page.dart
widgets/
app_button.dart
empty_state.dart
error_view.dart
loading_overlay.dart
l10n/...

Data models (essentials)

```
// feeder.dart
import 'package:freezed_annotation/freezed_annotation.dart';
part 'feeder.freezed.dart';
part 'feeder.g.dart';

@freezed
class Feeder with _$Feeder {
  const factory Feeder({
    required String id,
    required String name,
    required String ownerUid,
    required String firmwareVersion,
    required bool online,
    required int hopperPercent,          // 0..100
    String? locationLabel,
  }) = _Feeder;

  factory Feeder.fromJson(Map<String, dynamic> json) => _$FeederFromJson(json);
}

// schedule.dart
@freezed
class Schedule with _$Schedule {
  const factory Schedule({
    required String id,
    required String feederId,           // "unassigned" if placeholder
    required String title,
    required String cron,               // e.g., "0 8 * * *"
    required int grams,                 // portion size
    required bool enabled,
    String? timezone,                   // backend resolves to UTC
  }) = _Schedule;
  factory Schedule.fromJson(Map<String, dynamic> json) =>
_$ScheduleFromJson(json);
}

// feed_action.dart
@freezed
class FeedAction with _$FeedAction {
  const factory FeedAction({
    required String feederId,
    required int grams,
    @Default(DateTime.now) DateTime requestedAt,
  }) = _FeedAction;
}
```

Routing & guards

```
// app_router.dart
final router = GoRouter(
  routes: [
    GoRoute(path: '/login', builder: (_, __) => const LoginPage()),
    GoRoute(path: '/register', builder: (_, __) => const RegisterPage()),
    GoRoute(
      path: '/',
      redirect: (ctx, state) => ref.read(authControllerProvider).isLoggedIn ? '/feeders' :
'/login',
    ),
    GoRoute(path: '/feeders', builder: (_, __) => const FeedersPage()),
    GoRoute(path: '/feeder/:id', builder: (_, s) => FeederDetailPage(feederId:
s.params['id']!)),
  ],
);
```

State & controllers (Riverpod)

- *AuthController: handles login/register/logout, exposes AsyncValue<User?>.*
- *FeedersController: provides list, add/rename/remove; subscribes to realtime updates.*
- *ScheduleController: CRUD for schedules; supports "unassigned" when device removed.*
- *CameraController: resolves stream URL or WebRTC SDP; handles permission & player state.*
- *DashboardController: derived state (next feed, last feed, hopper gauge, online state).*

Example:

```
final feedersControllerProvider = StateNotifierProvider<FeedersController,
AsyncValue<List<Feeder>>>(
  (ref) => FeedersController(ref.read),
);

class FeedersController extends StateNotifier<AsyncValue<List<Feeder>>> {
  FeedersController(this._read): super(const AsyncValue.loading()) { _init(); }
  final Reader _read;

  Future<void> _init() async {
    final repo = _read(feederRepositoryProvider);
    // subscribe to realtime (Stream<List<Feeder>>)
    repo.watchFeeders().listen((feeders) => state = AsyncValue.data(feeders),
      onError: (e, st) => state = AsyncValue.error(e, st));
  }

  Future<void> rename(String id, String newName) async {
    await _read(feederRepositoryProvider).renameFeeder(id, newName);
  }

  Future<void> remove(String id) async {
    await _read(feederRepositoryProvider).removeFeeder(id);
    // Schedule rule: convert sole-device schedules to 'unassigned'
    await _read(scheduleRepositoryProvider).repointSchedulesForRemovedDevice(id);
  }
}
```

Screens & flows

1) Auth (Login/Register)

- *UI: email/password fields, "Continue" button, "Create account" link.*
- *Validation: live error text; disabled submit until valid.*
- *Edge cases: show auth errors, rate-limit captcha fallback.*
- *Post-login: route to /feeders.*

2) Feeders list

- *Card per feeder: name, online badge, hopper %, last feed time, quick actions:*
  - *Feed Now (grams picker + confirm)*
  - *Open (detail tabs)*
  - *Rename (inline dialog)*
  - *Remove (with confirmation)*

- *Add Feeder: QR scan (Wi-Fi credentials + claim token) or manual code.*

3) Feeder detail (tabs)

- *Dashboard: next scheduled feed, last feed log, "Feed Now" with optimistic UI (spin + toast → confirm result via push/realtime).*
- *Schedule: list grouped by enabled/disabled; FAB "Add schedule".*
- *Camera: full-width player; tap to fullscreen; toggle mute; snapshot (if supported).*
- *Logs: recent events (feed executed, jam, hopper low).*

4) Schedule editor

- *Bottom sheet or page:*
  - *Title, cron builder (simple presets: 7:30 AM daily, Twice/day, Custom → advanced cron)*
  - *Portion size (grams slider)*
  - *Device selector: dropdown of feeders + "Unassigned"*
  - *Enable switch*
  - *Save validates & writes*
- *On device removal: backend publishes schedule patches; frontend mirrors (or calls repository helper to repoint to "Unassigned").*

Immediate feed UX (optimistic)

1. *User taps Feed Now → select grams → Confirm.*
2. *UI shows "Dispensing…" spinner, disables button (cooldown).*
3. *Send POST /feeders/{id}/feed body {grams}.*
4. *On success (or WebSocket/FCM confirmation), show success toast & append to log.*
5. *On error: revert cooldown, show actionable message.*

```
Future<void> feedNow(String feederId, int grams) async {
  _optimisticStart(feederId);
  final result = await _read(feederRepositoryProvider).feedNow(FeedAction(feederId: feederId, grams: grams));
  result.fold(
    (failure) => _showError(failure),
    (_) => _toast('Feeding started'),
  );
}
```

Video streaming (frontend choices)

- *HLS (recommended first): stream URL → video_player (with chewie if you want controls). Good CDN cache, ~3–8s latency.*
- *WebRTC: ultra-low latency & talkback; use flutter_webrtc. Needs signaling server + TURN.*

```
// HLS example
final controller = VideoPlayerController.networkUrl(Uri.parse(stream.hlsUrl));
await controller.initialize();
controller.play();

// WebRTC sketch (pseudocode)
final pc = createPeerConnection({...});
final stream = await navigator.mediaDevices.getUserMedia({'audio': false, 'video': false});
await pc.setRemoteDescription(remoteSdpFromBackend);
final answer = await pc.createAnswer();
await pc.setLocalDescription(answer);
sendAnswerToBackend(answer);
```

Realtime updates

- *If using Firestore: Stream<QuerySnapshot> → map to models.*
- *If using your API: WebSocket topic per user or per feeder to push:*
  - *feeder status (online, hopper %)*
  - *feed execution results*
  - *schedule changes*
- *Riverpod controllers subscribe in init() and update state accordingly.*

Local caching & offline

- *Hive boxes: feeders, schedules, logs*
- *Write-through cache on successful network calls; read-through on load.*
- *Show "Offline" banner; allow schedule edits → queue mutations; process when online.*

Notifications (UX hooks)

- *FCM topics per feeder: feeder.{id} → "Feeding executed", "Jam detected", "Low hopper".*
- *Deep links to FeederDetail/Logs.*

Theming & accessibility

- *Light/dark theme, large tap targets (48dp), semantic labels on critical buttons ("Feed now 50 grams").*

Error handling

- *Central Failure types: AuthFailure, NetworkFailure, ValidationFailure, DeviceBusyFailure.*
- *ErrorView with retry; snackbars for transient issues.*

Testing

- *Unit: repositories & controllers (use dio_adapter/mocktail).*
- *Widget: schedule editor, feeder card interactions.*
- *Golden: feeder list empty vs. populated vs. offline.*
- *Integration: auth → add feeder → schedule → feed now → camera open.*

Example: Schedule editor (UI skim)

```
class ScheduleEditorSheet extends ConsumerStatefulWidget {
 final Schedule? existing;
 final String? initialFeederId;
 const ScheduleEditorSheet({this.existing, this.initialFeederId, super.key});

 @override
 ConsumerState<ScheduleEditorSheet> createState() => _ScheduleEditorSheetState();
}

class _ScheduleEditorSheetState extends ConsumerState<ScheduleEditorSheet> {
 final titleCtrl = TextEditingController();
 TimeOfDay time = const TimeOfDay(hour: 8, minute: 0);
 int grams = 50;
 String? feederId;
 bool enabled = true;

 @override
 void initState() {
  super.initState();
  final s = widget.existing;
  if (s != null) {
   titleCtrl.text = s.title;
   // convert cron → time for display (backend stores canonical cron)
   grams = s.grams;
   feederId = s.feederId;
   enabled = s.enabled;
  } else {
   feederId = widget.initialFeederId ?? 'unassigned';
  }
 }

 @override
 Widget build(BuildContext context) {
  final feeders = ref.watch(feedersControllerProvider).value ?? [];
  return SafeArea(
   child: Padding(
    padding: const EdgeInsets.all(16),
    child: ListView(
     children: [
      TextField(controller: titleCtrl, decoration: const InputDecoration(labelText:
'Title')),
      const SizedBox(height: 12),
      ListTile(
       title: const Text('Time'),
       subtitle: Text(time.format(context)),
       onTap: () async {
```

```
            final picked = await showTimePicker(context: context, initialTime: time);
            if (picked != null) setState(() => time = picked);
          },
        ),
        const SizedBox(height: 12),
        Text('Portion (grams): $grams'),
        Slider(min: 10, max: 300, divisions: 29, value: grams.toDouble(), onChanged: (v)
=> setState(()=> grams = v.toInt())),
        const SizedBox(height: 12),
        DropdownButtonFormField<String>(
          value: feederId,
          items: [
            const DropdownMenuItem(value: 'unassigned', child: Text('Unassigned')),
            ...feeders.map((f) => DropdownMenuItem(value: f.id, child: Text(f.name))),
          ],
          onChanged: (v) => setState(()=> feederId = v),
          decoration: const InputDecoration(labelText: 'Device'),
        ),
        SwitchListTile(value: enabled, onChanged: (v)=> setState(()=> enabled = v),
title: const Text('Enabled')),
        const SizedBox(height: 20),
        FilledButton(
          onPressed: _save,
          child: Text(widget.existing == null ? 'Create schedule' : 'Save changes'),
        )
       ],
      ),
    ),
   );
 }

 Future<void> _save() async {
  final cron = _toCron(time); // e.g., "0 8 * * *"
  final model = Schedule(
    id: widget.existing?.id ?? 'temp',
    feederId: feederId ?? 'unassigned',
    title: titleCtrl.text.trim().isEmpty ? 'Feeding' : titleCtrl.text.trim(),
    cron: cron,
    grams: grams,
    enabled: enabled,
    timezone: null,
  );
  await ref.read(scheduleRepositoryProvider).upsert(model);
  if (mounted) Navigator.pop(context);
 }
```

}

API expectations (so your frontend stays clean)

- *GET /feeders → [Feeder]*
- *PATCH /feeders/{id} {name}*
- *DELETE /feeders/{id} (backend triggers schedule repoint logic or frontend calls helper)*
- *POST /feeders/{id}/feed {grams}*
- *GET /schedules?feederId=... | ?scope=user → [Schedule]*
- *POST /schedules / PATCH /schedules/{id} / DELETE ...*
- *GET /video/{feederId} → {hlsUrl} or WebRTC offer/ICE config*

What you'll implement first (pragmatic order)

1. Auth → 2) Feeders list & add/remove/rename → 3) Feed Now (optimistic) → 4) Schedule CRUD + unassigned logic → 5) Camera (HLS first) → 6) Push notifications → 7) Offline cache & polish.

If you want, I can generate starter files (Riverpod providers, models with Freezed, and a working go_router shell) so you can drop them into your project."

[5c] ***Case Study 5 Google AI Response****: "Dog Feeder Mobile Application: Frontend Architecture Plan*

Framework: Flutter (Dart) Target Platform: iOS & Android Inferred Grade Level: Professional/Advanced (Due to IoT, State Management, and Video Streaming complexity)

I. Recommended Technology Stack

Feature Category	*Recommended Flutter Package(s)*	*Purpose*
State Management	*flutter_riverpod (or bloc/provider)*	*Handles complex, shared state (multiple feeders, schedules, user login status).*
Authentication	*firebase_auth (Assumes Firebase Backend)*	*Manages user registration, login, and sessions.*

Database/API	*cloud_firestore + http/dio*	*Firestore for user/feeder data; http/dio for direct feeder commands (if applicable) or Cloud Function triggers.*
Scheduling/Time	*intl*	*Formatting and handling local/scheduled times.*
Navigation	*go_router*	*Declarative routing for deep linking and simple, scalable navigation.*
Video Streaming	*flutter_webrtc or Platform Channels*	*Handles low-latency video feed (RTSP or WebRTC) from the feeder's camera. This is the most complex integration point.*
Local Notifications	*flutter_local_notifications*	*Alerts the user about feeding completions or errors.*

II. Screen Flow and Feature Breakdown

The application will be structured around five main screens:

1. Authentication Flow

- *Login Screen: Accepts email/password. Navigates to the Feeder List on success.*
- *Registration Screen: Collects user details, creates an account. Navigates to the Feeder List.*

2. Feeder List Screen

- *Displays a list of all feeders linked to the user's account (e.g., "Max's Feeder," "Luna's Feeder").*
- *Action: Tapping a feeder navigates to the Feeder Control Screen.*
- *Action: An "Add Feeder" button initiates the setup flow (pairing).*

3. Feeder Control Screen (The Core)

This screen is the central hub for a single feeder's operations.

Immediate Feed	*A prominent, styled button that triggers an API call (/api/feed/{feederId}/immediate). Must handle loading state and success confirmation.*

Video Feed	*A dedicated Container or widget hosting the live video stream using the streaming package. Should handle connection/disconnection gracefully.*
Schedule List	*Displays all set schedules (e.g., "7:00 AM - 1.5 cups," "5:30 PM - 1.0 cup"). Each entry allows for editing or deletion.*
Add Schedule	*A button that opens a modal or navigates to the Schedule Editor Screen.*

4. Schedule Editor Screen

- *Inputs: Time picker (e.g., 7:00 AM), Portion size slider/input (e.g., 1.5 cups), and Frequency selector (Daily, Weekdays, etc.).*
- *Action: A "Save" button triggers the API call to update the feeder's scheduled feeding times.*

III. State Management Strategy (Using Riverpod)

We use Riverpod providers to manage data flow:

1. userProvider: Holds the current user's authentication state (User object or null).

2. feederListProvider: An AsyncNotifierProvider that fetches and watches the list of FeederModel objects from Firestore/API.

3. feederSchedulesProvider(feederId): A family provider that takes a feederId and provides the current schedule list for that specific device. This is crucial for multi-feeder support.

By using this approach, any change in one part of the app (e.g., updating a schedule on the Schedule Editor) automatically refreshes the Feeder Control Screen and the Feeder List Screen without manual intervention."