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

List of Figures

List of Tables

Chapter 1 – Executive Summary

Sit & Chow is an intelligent, interactive dog feeding and training system. It is designed to address a common problem faced by many pet owners: maintaining consistent feeding schedules and reinforcing proper behavior despite busy and unpredictable lifestyles. Pets are not only companions but also family members. Therefore, the expectation for technology that supports their health, training, and care has grown significantly. Sit & Chow explores this need by combining automated feeding, real-time sensing, and behavior-based reinforcement into one system.

Traditional automatic feeders offer convenience but lack behavioral interaction and situational awareness. They dispense food on a schedule regardless of whether the dog is present, behaving, or safe to feed. Sit & Chow bridges this gap by incorporating computer vision and depth sensing to detect when the dog is near the feeder and sitting before food is dispensed. Additionally, a mobile application allows users to manage feeding schedules, view logs, and monitor their pet's eating habits. Together, these hardware and software components support consistent training, reliable feeding, and smart tracking while adding convenience to taking care of their pet.

When it is the scheduled feeding time, a buzzer sound will alert the dog. The owner is also able to connect to the camera through the mobile app to speak and help aid the dog to eat or check in. The system uses an object detection and classification pipeline to determine if a dog is present and sitting before releasing the dispensing mechanism. A depth sensor is used to verify that the dog is within a certain range to improve the reliability of the algorithm. Once the posture and distance are confirmed, the stepper motor rotates a slotted wheel to dispense the portion size set by the user. Each scheduled feeding attempt, successful or unsuccessful, is automatically logged into the mobile app to be displayed.

Sit & Chow demonstrates how embedded systems, machine learning, and mobile technologies can converge to solve a real-world problem. The design offers a solution for households seeking a more interactive and intelligent approach to pet care. The overall goal is to design a feeder that is accurate, quiet, modular, capable, and safe while operating continuously. The final outcome should not just "work once," it must work every day with reproducibility and consistency, even if no user is available to inspect it.

By the end of this project, we expect to deliver a fully tested, fully documented smart feeding system that can reliably feed a dog on schedule, monitor food consumption, integrate training behaviors, with an integrated mobile application. If successful, Sit & Chow can serve as both an innovative consumer product concept and a reference design for future smart home appliances.

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 with 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 [1].

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 [1]. The camera has a 940nm infrared

LED fill light and high-sensitivity sensor, ensuring accurate detection even in low-light conditions [1].

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 [2]. 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 [3]. Using this information, it will then dispense the specific amount of food that correlates to the animal's weight, allowing for 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 [4].

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 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 a speaker.

Stretch Objectives

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

2.4 Specifications

2.4.1 System Specifications

Table 1: Specification of Sit & Chow

Aspect	Specification
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	0.1 – 0.1 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: Hardware Components

Component	Specification
MCU (ESP32-S3)	Wifi 2.4 GHz, BLE, 520 KB SRAM

Stepper Motor	~ 35 – 45Ncm Torque
Dispensing Wheel	Up to 1 cup
Power Supply	12V, 2A + 5V buck, safe
Speaker	65 – 70 dB at 1m, 2 – 3 sec sound cue
Camera	720p – 1080p, 10 – 30 fps
LED Indicators	Red, 2 – 5mA
Food Storage Container	Maximum 5kg
Manual Override Button	≥150 gf operating force
Ultrasonic Sensor	≤ 0.5 meters
Depth Sensor	≤ 0.5 meters
Backup Battery	NiMH AA pack 1.2V

2.5 System Diagrams

2.5.1 Hardware Diagram

The diagram below (Figure 1) illustrates the Sit and Chow's hardware architecture, with the ESP32 at the center connecting to every hardware component. It is divided into four parts: the power supply, main PCB, peripherals and hardware components, and mobile application. The diagram highlights how the MCU will govern almost every part in the hardware development process with an app to make it easier for users to manage the Sit & Chow feeding system. A manual override button was noted early on for any potential failures the system may have while feeding a dog.

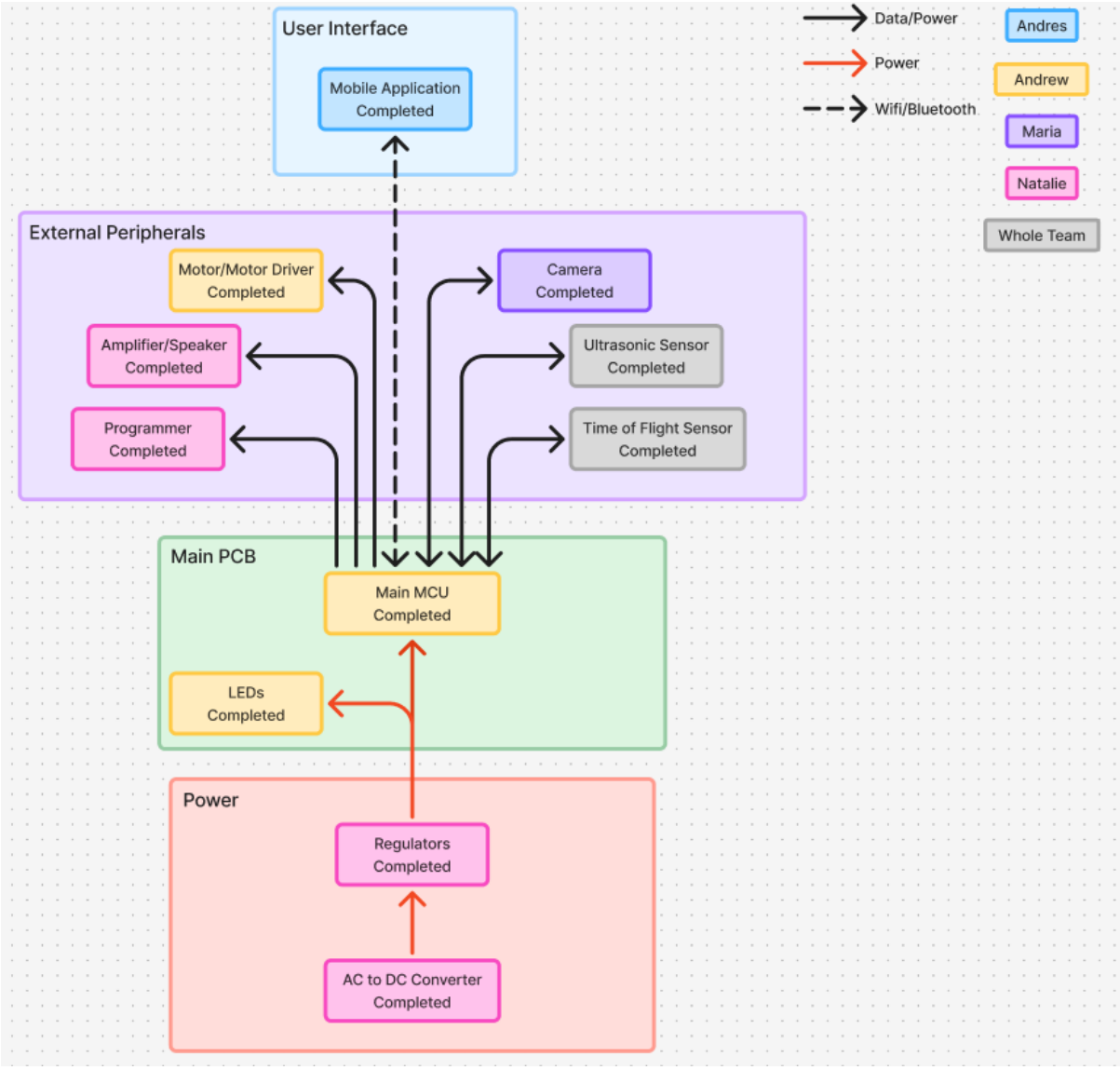

Figure 1: Hardware Diagram by Authors

2.5.2 Software Diagram

The software block diagram below (Figure 2) 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, mobile app communication, logging feeding outcomes, and user notifications for low food supply.

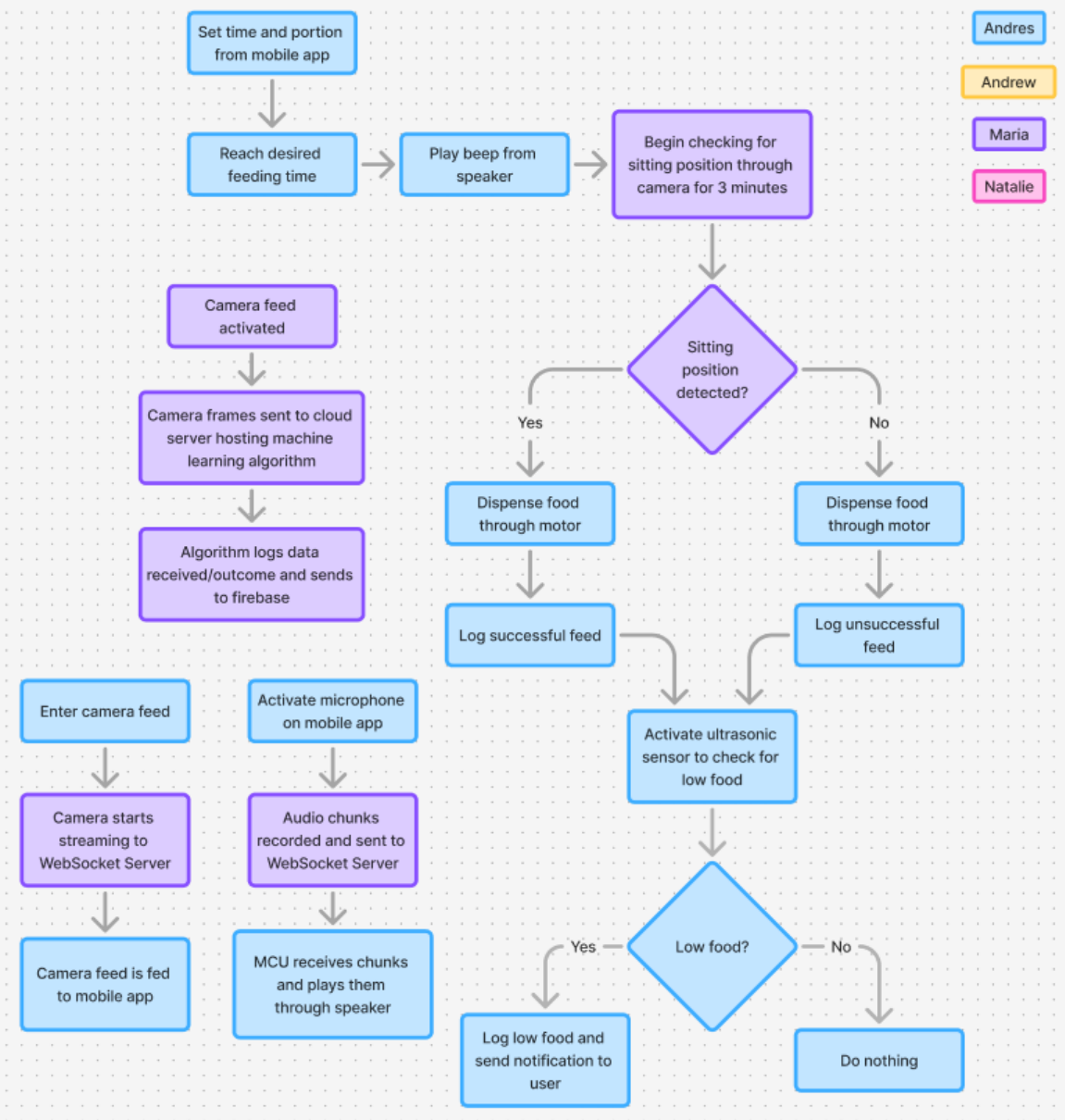

Figure 2: Software Diagram by Authors

2.6 House of Quality

In Figure 3 below, the House of Quality for Sit & Chow illustrates the relationship between customer needs and technical specifications and highlights key design priorities for Sit & Chow.

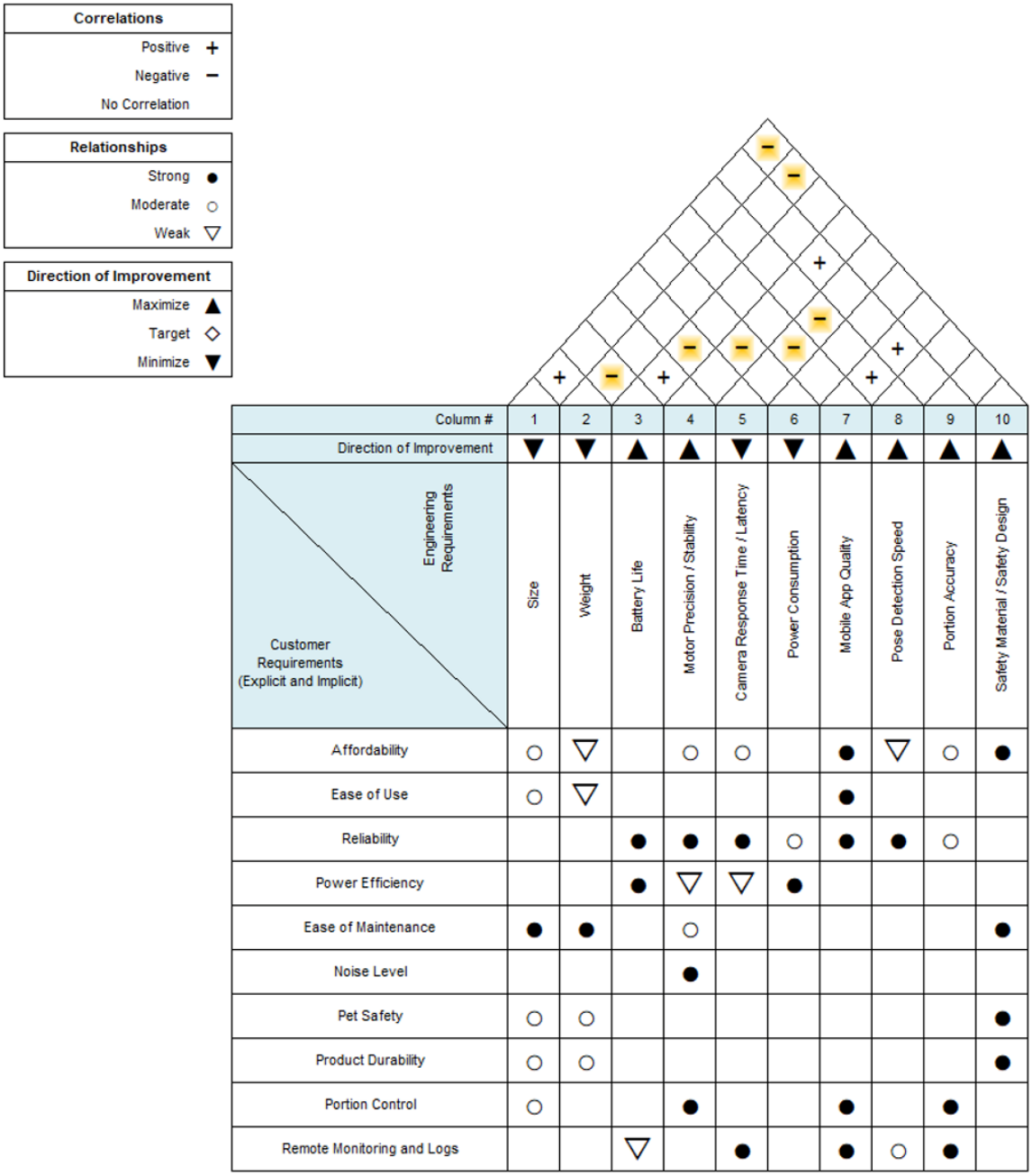

Figure 3: House of Quality by Authors

2.7 Prototype Illustration

Below is the illustrative diagram of the Sit & Chow smart feeder below, showing the integration of key components including the camera module, PCB, motor, and storage area.

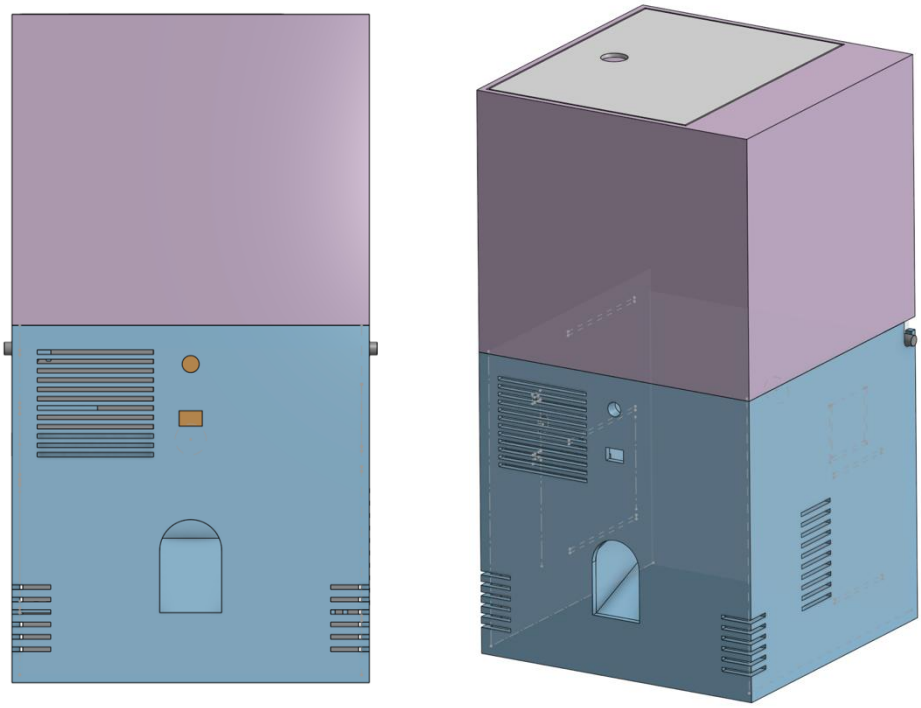

Figure 4: Prototype Illustration

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 [5]. 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 [6].

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 [7]. There is a long development time due to synthesis and debugging cycles, and the learning curve to design an FPGA is quite steep [7]. 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 [6]. 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+ [7]. 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.

Table 3: 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.

The comparison between an **MCU** like the ESP32-S3 and an FPGA highlights two fundamentally different approaches to embedded system design. FPGAs excel in applications requiring ultra-high performance, deterministic timing, and true parallel hardware execution. However, these advantages come at the cost of significantly higher development complexity, increased power consumption, the need for specialized HDL programming, and much greater hardware expense. In contrast, an MCU provides a balanced blend of processing capability, built-in wireless connectivity, low power usage,

and a highly accessible development environment suited for rapid prototyping and reliable long-term IoT deployment.

For a device that requires Wi-Fi/Bluetooth connectivity, real-time control, camera support, continuous operation, and a cost-effective hardware footprint, the MCU offers far more practical benefits. Its rich peripheral set, low operating power, and ease of debugging dramatically simplify development while still meeting all functional requirements. Because performance demands remain within the capabilities of a modern dual-core MCU, the added complexity and overhead of an FPGA provides no meaningful advantage for this application. Therefore, the ESP32-S3 MCU is the clear and efficient choice, delivering the best combination of capability, reliability, cost, and development efficiency for our system.

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 [8, 9]. 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 [9]. 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.

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 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.

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 make 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 [10]. 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 [10, 11]. 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 [12]. Furthermore, older boards have inefficient power consumption and typically do not have extended support ultra-low power processing [12]. 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 [12, 13]. 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 [10, 11]. 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 [13]. 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 [10, 11]. 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, since the Arduino lacks a camera interface, instability is increased [13]. For these reasons, we decided not to select the Arduino family of MCUs for our design.

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 [14]. 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 [14]. 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 [14]. 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 [15]. Lastly, these boards are also fairly large and require thermal management, meaning that it is not suitable for compact embedded enclosures.

So, 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.

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 [16]. 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 [17]. 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 [17].

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 [17]. 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 [17]. 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.

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 [16].

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.

Table 4: 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

Evaluating the STM32, Arduino platforms, Raspberry Pi, hybrid Pi-plus-MCU solutions, and ESP32 variants reveals significant differences in power efficiency, computational capability, development complexity, and cost. High-performance options like the Raspberry Pi or Pi-MCU hybrids provide strong processing capabilities but come with

substantial power demands, higher costs, and increased software complexity due to the need for OS management and multi-platform synchronization. Conversely, Arduino boards offer simplicity and low cost but lack the processing power, wireless connectivity, and real-time control needed for video streaming, cloud communication, and intelligent system behavior. More advanced solutions like STM32 or ESP32 paired with a small edge accelerator deliver strong performance but introduce extra development overhead, specialized SDK requirements, and more complex hardware integration.

The standard ESP32 strikes the ideal balance between performance, efficiency, and simplicity. Its 240 MHz dual-core processor, built-in Wi-Fi/BLE, reasonable power draw, abundant availability, and straightforward firmware ecosystem allow it to support streaming, device control, backend communication, and real-time responsiveness without additional chips or toolchains. It offers an extremely low cost while still meeting all functional requirements of the project. For these reasons, the ESP32 alone is the most practical, efficient, and developer-friendly platform, making it the final choice for our system's microcontroller.

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 [18]. 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.

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 [19]. 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 [19]. 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.

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 [20]. 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 [20]. 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 [20]. 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.

Table 5: 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	$9–$

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-N8R8. This lets us start programming without needing the PCB or other components.

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 [21]. 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.

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 [22]. 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 [23]. 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.

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 [24]. 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.

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 [25]. 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.

Table 6: 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

We selected **Wi-Fi** because it is the only option that delivers the high bandwidth, low latency, and strong reliability needed for data-intensive features like live video streaming, frequent device updates, and real-time interaction with the feeder. While Bluetooth, Zigbee, and LoRaWAN excel in low-power or long-range applications, they all suffer from severely limited data rates and cannot support our camera streaming, mobile app connectivity, and cloud integration needs. Wi-Fi also offers robust security (WPA2/WPA3) and seamless access to household networks, making it the most practical and user-friendly choice for a smart home device that requires continuous and fast communication.

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 physical implementation 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 into 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 [26]. 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.

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 [27]. 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.

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 [27]. 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.

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 an 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 [28]. 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.

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 [28]. 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 [29]. 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.

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 [30]. 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.

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 [31]. 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.

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 7: Types of Cameras

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°–120°	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

For the dog feeder system, we have three camera modules were evaluated which are OV2640, OV5640, and Realtek AMB82. The OV2640 offer low cost and low power consumption but suffers from limited resolution and weaker low-light performance, which may reduce accuracy when deleting food levels or pet movement. The Realtek AMB82 integrates Wifi and Bluetooth, making it suitable for IoT designs, but it has higher power consumption, larger size, and limited ecosystem support making it harder to integrate with ESP32 in prototype setup. The OV5640 provides the best balance: it offers 5MP resolution, wide field of view, strong low-light and WDR performance, and uses DVP/MIPI interface, which is compatible with many ESP32/ESP32-S3 development boards. Although slightly more expensive, it delivers sharper images and higher reliability, also make it ideal for motion detection. Therefore, **OV5640** is chosen as the final camera since it provides the best image quality and low-light capability while remaining reasonable in price and power consumption.

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:

Bipolar stepper motor

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 [32, 33, 34]. 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 [32].

DC geared motor

A DC geared motor, on the other hand, is more cost-effective and provides high torque output [35]. However, to achieve precision in food dispensing, it usually requires additional components like an encoder, clutch, or extensive calibration [35]. When picking a DC motor for a pet feeder, we focus on getting the right balance of voltage, current, and torque [36]. Too much power can break the food or make it noisy, so we prefer micro DC gear motors for their quiet and efficient performance [36]. 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.

RC servo

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 8: Comparing types of Motors

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 best choice for a dog feeder mechanism is the **NEMA-17 stepper motor** (42 × 38 mm, 1.7 A/phase) from the comparison table, Table 8. 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.

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

Table 9: Models of Motor Driver

Feature	A4988	DRV8825	L298N
Key Characteristics	8–35 V, up to 2.0 A/phase, microstepping up to 1/16	8.2–45 V, up to 2.2 A/phase, microstepping up to 1/32	5–35 V, ~2 A max, no microstepping
Advantages	Simple Low cost Widely available	High current capability (up to 2.2 A per phase) Fine microstepping (1/32) for smooth motion Built-in thermal and overcurrent protection High efficiency and compact design	Easy to use
Disadvantages	Lower microstepping resolution and slightly lower performance	Requires current limit adjustment Slightly more complex setup compared to basic drivers	High power loss Excessive heat Poor precision
Evaluation	Good alternative, lower resolution	High precision, efficient, best choice	Inefficient, outdated for precision control

After comparing all the motor driver options in Table 9, we decided to use the **DRV8825** because it best fits for a dog feeder [37, 38, 39]. Selecting a motor driver for a NEMA 17 stepper motor requires careful consideration of several parameters. First, the current rating of the driver must match or exceed the motor's rated current, which typically ranges from 1 A to 2 A per phase. The DRV8825 supports up to 2.2 A, making it suitable for this application. Second, the supply voltage must be compatible with both the motor and the driver; higher voltages can improve torque performance. Third, microstepping capability

is important for smooth and precise motion, particularly in applications such as controlled food dispensing. The DRV8825 provides up to 1/32 microstepping, which enhances accuracy. Finally, thermal protection and efficiency must be considered to ensure long-term reliability.

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 to 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 louder sound 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 [40, 41] .

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

Table 10: 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.	4Ω 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 research for the technical parameters and comparing, we have created a table 10 [42, 43, 44, 45, 46, 47]. 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: **Speaker - 3" Diameter - 4 Ohm 3 Watt** [48]

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 [49]. 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 11: 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).
Linearity / Sound Quality	Excellent	Good (reduced distortion)	Good (depends on filtering)
Distortion	Very low	Low	Low to moderate (switching noise)
Power Consumption	Very high	Moderate	Very low

Heat Dissipation	High	Medium	Low
Complexity	Simple design.	Moderate, requires a push-pull arrangement.	Higher complexity, uses Pulse-Width Modulation (PWM).
Typical Applications	High-fidelity audio systems	Embedded systems, small audio devices	Modern portable electronics
Suitability for Dog Feeder	Not suitable	Highly suitable	Suitable but complex

Based on our research, we made Table 11 [50, 51, 52, 53, 54, 55, 56]. 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, **Class AB amplification** is selected for the dog feeder system because it provides the best balance between performance, simplicity, and practicality compared to other amplifier classes. While Class A amplifiers offer excellent linearity, their very low efficiency and high heat dissipation make them unsuitable for low-power embedded applications. Class AB amplifiers combine the advantages of both Class A and Class B by minimizing distortion while maintaining moderate efficiency, resulting in clear and reliable audio output. Although Class D amplifiers achieve higher efficiency and power output, they require more complex circuitry, including switching components and output filtering, which increases design difficulty and potential noise issues. For a system that only requires simple audio alerts, such as a dog feeder, the added complexity of Class D is unnecessary. Therefore, Class AB is chosen because it offers sufficient sound quality, low cost, ease of implementation, and compatibility with microcontroller-based systems, making it the most suitable option for this application.

However, there are 3 types of Class AB amplifiers we need to concern ourselves with:

Table 12: Models of Class AB Amplifiers

Feature	LM386	NE5532	LM1875
Amplifier Type	Integrated power amplifier	Dual low-noise op-amp (needs external stage)	High-power audio amplifier

Output Power	~0.25–0.5 W	Low (requires external power stage)	Up to 20 W
Supply Voltage	4V – 12V	±5V to ±15V	±16V to ±30V
Circuit Complexity	Very low	High	High
External Components	Minimal	Many required	Many required
Size / Integration	Very compact	Medium	Large (with heatsink)
Power Consumption	Low	Moderate	High
Cost	Very low	Medium	High
Ease of Use	Very easy	Complex	Complex
Typical Applications	Embedded systems, small speakers	Audio preamplifiers	Hi-fi audio systems
Suitability for Dog Feeder	Highly suitable	Not suitable	Not suitable

In table 12, some research has been done to see which fits better for our project [57, 58, 59]. Among the evaluated Class AB amplifiers, the **LM386** is selected as the most appropriate choice for the dog feeder system due to its superior balance of simplicity, cost, and functionality. Compared to alternatives such as the NE5532, and LM1875, the LM386 requires significantly fewer external components, reducing both circuit complexity and implementation time. While higher-power amplifiers like the LM1875 provide greater output capability, they are unnecessary for low-power audio alerts and introduce additional design challenges such as heat management. Similarly, op-amp-based solutions like the NE5532 require external amplification stages, increasing system complexity. The LM386, on the other hand, is specifically designed for low-voltage, low-power applications and can directly interface with microcontrollers such as the ESP32. Its adjustable gain, compact size, and minimal power requirements make it highly suitable for embedded systems. Therefore, the LM386 is chosen as the optimal Class AB amplifier for this application.

In conclusion, we picked **LM386** amplifier. [57]

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 disadvantages 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 [60].
  - Long Lifespan: A typical LED can last for tens of thousands of hours, far outlasting traditional bulbs and reducing maintenance [61].
  - Durability: LEDs are solid-state components with no fragile filaments or glass enclosures, making them highly resistant to vibration and impact [62].
  - 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 [62].
  - Voltage Sensitivity: LEDs are sensitive to voltage fluctuations and require a stable power supply or current-limiting resistors to prevent damage [62].
  - 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), Through-Hole (THT), and High-Power LEDs:

Table 13: Types of LED

Feature	Surface Mount (SMD) LED	Through-Hole (THT) LED	High-Power / Panel / 16 mm Ring LED
Mounting	Mounted on PCB surface	Leads through PCB holes	Panel-mounted, external wiring
PCB impact	No holes; optimal routing flexibility	Requires holes; reduces routing area	Requires large front-panel cutout
Profile height	Low profile; compact footprint	Tall; protrudes above PCB	Bulky; designed for external visibility

Assembly suitability	Ideal for automated assembly; 0805 still hand-solderable	Good for hand-soldering	Requires mechanical fastening
Mechanical durability	Adequate when located behind a lens or diffuser	Strong if physically accessible	Extremely high; rugged front-panel component
Power consumption	Very low (2–5 mA typical)	Moderate	High; unnecessary for status indication
Preferred use case	Consumer electronics, compact enclosures	Simple hobby-level projects	Aesthetic status rings or industrial indicators

After researching information, we have a comparative Table 13 [63, 64]. 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. In addition, high-power LEDs are excluded due to excessive current consumption and mechanical burden.

Now, we look at the different models of LED.

Table 14: Models of LEDs

Feature	ROHM SML-211UTT86 (Red)	Kingbright KP-2012SGC (Green)	Lumex SML-LX0805UWC-TR (White)
Package size	0805 (2012)	0805 (2.0 × 1.25 × 1.1 mm)	0805 (2012)
Typical V_F @ 20 mA	~1.8 V	2.2–2.5 V	~3.2–3.5 V
Luminous intensity	~2.5 mcd	4–15 mcd	~150 mcd white
Viewing angle	130°	120°	Wide, clear lens (120° typ.)
MCU Compatibility	~3.2W into 4Ω	~3W ×2 into 4Ω	~2.5W into 4Ω
Approx unit price (low qty)	~0.10-0.41 USD, depending on supplier	~ 0.385- USD each	~0.82-1.7 USD each at major distributors

After doing research, we have the table 14, we decide to pick the **ROHM SML-211UTT86** (red, 0805 SMD indicator LED) because it strikes the best balance of cost, simplicity, and fit for purpose in the dog-feeder design [65, 66, 67]. The red emission is ideal for a fault or alert indicator which simplifies the user interface without needing multi-color complexity. It also has a low unit cost. Compared to higher-brightness white SMD LEDs (which cost significantly more), the SML-211UTT86 offers a compact, cost-effective, and manufacturable solution perfect for the smart feeder's board-level status lamp.

3.1.10 Programmer

For the dog feeder project, the programmer is the hardware interface used to upload code from a computer to the microcontroller and to monitor serial data during testing. Since the feeder must control timing, motor movement, sensors, and audio feedback, the programmer must be reliable, low-cost, easy to use, and compatible with 3.3 V or 5 V logic.

When considering the choice of a push button, there are some options we can comnpare:

Table 15: Types of Programmers

Hardware Technology	Pros	Cons
USB-to-UART programmer	Low cost, simple wiring, supports Arduino/ESP-style serial upload, useful for debugging	Needs correct TX/RX wiring and logic voltage
ISP programmer	Directly programs AVR chips without bootloader	More pins required; less convenient for normal serial debugging
JTAG/SWD programmer	Advanced debugging, breakpoints, memory inspection	More expensive and more complex
Built-in USB microcontroller	No external programmer needed	Requires microcontroller with native USB support

Table 15 will be easier to compare [68, 69]. We can see USB-to-UART is the best choice

Now, we look at the model for the programmer.

Table 16: Models of Programmers

Feature	CP2102	CH340G / CH340C	FT232R
Supply/ Logic Support	Commonly used with 3.3 V and 5 V UART boards; includes USB transceiver and integrated clock	Supports 5 V and 3.3 V operation	Supports USB-to-UART with baud rates up to 3 Mbaud

Advantages	Reliable, compact, no external crystal required, good driver support, supports USB-to-UART development	Very cheap, widely used on Arduino clone boards	Very reliable and professional, strong driver support
Disadvantages	Slightly more expensive than CH340	Driver installation can be less convenient; CH340G may require external crystal depending on version	Usually more expensive than CP2102 and CH340

Based on Table 16, the **CP2102** is a great choice for the dog-feeder project [70, 71, 68, 69]. The recommended programmer for the dog feeder is the CP2102 USB-to-UART module. I would choose CP2102 because it gives the best balance between cost, reliability, and ease of use. The CP2102 includes an integrated USB transceiver, integrated clock, programmable memory information, and an on-chip reset circuit, so it requires fewer external components than older designs. Compared with CH340, CP2102 is usually considered a cleaner option for a project report because it has strong manufacturer documentation and is commonly used in ESP. Compared with FT232R, CP2102 is usually more cost-effective while still being reliable.

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 17: 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 17 to see which sensor would be fitted for our project [72, 73].

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.

Now, we need to decide the sensor to check food level.

Table 18: Models of Food Storage Sensor

Feature	RCWL-1601	JSN-SR04T	A02YYUW
Supply Voltage & Logic	3.0–5.0 V, logic = supply	5 V supply (some variants 3.3 V logic tolerant)	3.3–5 V supply (UART output)
Range (Typical)	~2–400 cm	~25–600 cm	~6–750 cm
Key Features	HC-SR04 protocol but works at 3.3 V or 5 V	Waterproof transducer on cable	Waterproof ultrasonic with digital UART output

Pros	Direct ESP32 compatibility; no level shifting; low cost; compact	Very rugged; moisture/dust resistant; long range	Waterproof; noise-filtered digital output; good accuracy
Cons	Not waterproof; dust may degrade accuracy over time	More expensive; mounting more complex; cable + puck footprint	More expensive; UART parsing required; slower update rate
Suitability for Dog Feeder	Best choice for indoor dog feeder; ideal electrical compatibility	Excellent for outdoor or washable feeders; overkill for indoor use	Good for industrial feeders; more complex than needed

After doing research and make a table 18, we select the **RCWL-1601** over the other ultrasonic modules because it delivers the best balance of electrical compatibility, simplicity, and performance for an indoor dog-feeder system [74, 75, 76]. Unlike the JSN-SR04T, it operates natively at 3.3 V, allowing direct connection to the ESP32-S3 without level shifters, extra regulators, or risk to the microcontroller's pins. While the JSN-SR04T and A02YYUW offer waterproofing and industrial durability, those features add cost, complexity, and mounting challenges that are unnecessary for a sealed indoor hopper. The RCWL-1601 achieves reliable distance measurement, low noise, stable operation, and an extremely low price point makes it the most efficient and good solution for a smart feeder where 3.3V logic integration and simple PCB design matter most.

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 [77, 78]. 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 [77, 78].

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 [79, 80]. 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.

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 [81, 82]. 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.

mmWave

Millimeter-wave (mmWave) sensors operate by using Frequency-Modulated Continuous Wave (FMCW) radar to detect objects and measure distance by analyzing the Doppler shift of reflected electromagnetic waves [83, 84]. They can function in any lighting condition and are unaffected by fur color and communicate through UART or SPI [83, 84]. It can potentially be sensitive enough to detect breathing motion.

Table 19: 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 Can 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 measuring principal 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 20: 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 Software

3.2.1 Mobile Application Stack

3.2.1.1 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 [85]. 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 [86]. 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.

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 [87]. When it comes to authentication and security, Supabase has built-in JWT/OAuth support, making it easy to manage multiple devices and users securely [87]. 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.

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 [88]. 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 [89]. 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.

Table 21: 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.
Maintenance	Minimal maintenance (serverless).	Low maintenance; open source with optional self-hosting.	Higher maintenance— updates, security patches, backups managed manually.

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.

3.2.1.2 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 [90]. 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 [90]. 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.

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 [91]. 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 [91]. 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, our group members have little to no 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.

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 [92]. Its custom rendering engine, Skia, ensures a consistent UI across all platforms [92]. 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.

Table 22: 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

Because Flutter has seamless support for the backend, Firebase, it is an excellent choice for the frontend application. The other options are not as suitable due to strictly having only a mobile application, complexity, or a lack of knowledge in the area. In addition, previous coursework gives an advantage to the learning curve with foundational knowledge that will reduce overall implementation time.

3.2.2 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 [93].

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 [94]. For example, dispensing the food and changing the LED light indicating the task was successful.

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 [95]. It is much slower in performance and execution for embedded tasks compared to C/C++ , meaning significant runtime overhead [96].

Table 23: 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 precision	Manual memory management. Limited abstraction. Longer development cycles
C/C++ with FreeRTOS	Run components concurrently.	Real-time scheduling ensures deterministic behavior. Enables parallel task management.	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.

3.2.3 Detection and Classification 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 [97]. 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.

- **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 [98, 99]. 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.

- **Object Detection and Localization**

  The model identifies a region in the image that has a dog by using a bounding box and confidence score [100]. 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 [101]. 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.

- **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 [102]. 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.

- **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" [103]. 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 $0.75 > 0.5 \rightarrow$ label = "sitting" $\rightarrow$ 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 [104]. The conversion would be done through a framework called ONNX [105], which can convert any deep-learning framework into another framework.

Model Training

The development of the Sit & Chow detection and posture classification models requires a comprehensive training pipeline to ensure high accuracy and generalization across real-world conditions. While the inference controls how the model performs, the training process determines the quality of the model's features for reliable detection and classifications.

- **Data Collection:**
  Training begins by gathering images and video frames either from online datasets or directly captured from the Sit & Chow camera module. The images have real conditions such as varied lighting, backgrounds, shadows, dog positions, and breeds. The data includes images of various sizes and distances since the image

captured from the detection will be cropped. In addition, images with no dog will be intentionally included so that the model learns to ignore irrelevant backgrounds. If the images were captured, they need to be manually labeled. For the detection model, the bounding boxes must be annotated around the dog. For the posture classifier, each image is labeled "0" for "sitting" and "1" for "not sitting." Only images where the dog is fully sitting will be labeled 0, so images where the dog is transitioning from one pose to sitting will be labeled 1.

- **Data Preparation:**
  The dataset will be split into an 80% testing set and 20% testing. Class balance needs to be monitored so that the classifier does not become biased towards predicting "not sitting."
- **Training Procedure**
  Training is performed in frameworks like PyTorch and TensorFlow, which can support GPU acceleration and automatic differentiation. The workflow follows the standard deep-learning optimization cycle:
  - **Forward Pass:**
    The input image passes through the detection or classification network to generate a bounding box or posture prediction.
  - **Loss Calculation:**
    The model compares its prediction with the correct label, and the loss function measures how far off the prediction is. For detection, loss combines localization error and classification error. For posture classification, binary cross-entropy calculates the loss between predicted labels and true labels.
  - **Backpropagation:**
    The ML framework calculates how much each weight, a value that controls how strongly an input influences the output, contributed to the error. Then, gradients are computed with respect to each weight to determine how the errors influence the parameter updates.
  - **Optimizer:**
    The optimizer, such as Adam or gradient descent, updates a model's weights using a learning rate. The training continues for multiple cycles until the loss plateaus.

  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.
- **Hyperparameter Tuning**
  Several hyperparameter are tuned to achieve an optimal performance:
  - Learning rate
  - Batch size
  - Number of epochs
  - Confidence threshold
  - Input resolution
- **Evaluation and Validation**

Detection Metrics:
  - mAP50
  - mAP50-95
  - Precision and recall
  - Localization accuracy

Classification Metrics:
  - Accuracy
  - Precision/Recall
  - F1-score
  - Confuson matrix

- **Evaluation and Validation**
  To reduce overfitting, the following techniques are used:
  - Augmentation diversity
  - Dropout layers
  - Weight decay regularization
  - Model checkpointing

Training From Scratch Vs. Fine-Tuning Pretrained Models

There are two primary approaches to developing a model for a specific task: training from scratch and fine-tuning [106]. 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 and 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, including dogs, with high accuracy saves the intensive computation of training a new model with potentially lower accuracy. In addition, YOLOv11 is trained with the COCO dataset, where the images are already annotated saving time on annotating images manually.

In Figure 5 below, performance metrics show YOLOv11 reaching an mAP50-95 up to 55% of the time for the multi-class dataset COCO and is faster compared to other YOLO models' performance [2].

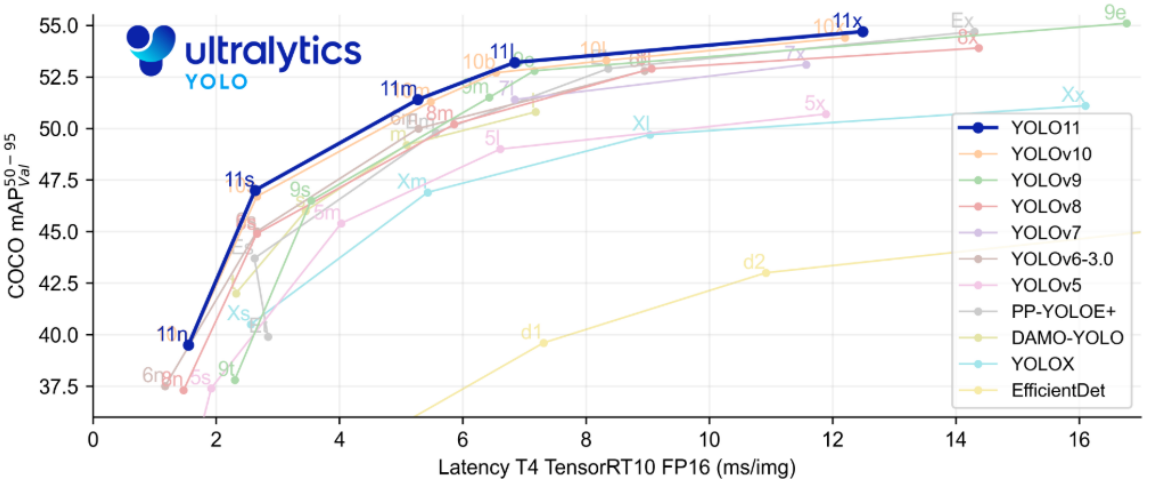

Figure 5: Ultralytics YOLO Performance Metrics

For the classification model, since it is a binary classifier, training from scratch ensures the model may be better than fine-tuning a pre-trained model for Sit & Chow. The required task is highly specific, distinguishing "sitting" and "not sitting," so the classifier becomes highly specific to Sit & Chow. Training from scratch allows for full control of the architecture size, depth, and activation functions and reduces the risk of transfer bias, where a pretrained model learns irrelevant features from its original dataset, making it much harder to reach high classification accuracy even with fine-tuning. Nevertheless, there are not many resources available for a pretrained model, but there are many use cases with YOLO models as seen in Chapter 2.2.

Table 24: 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 [107], 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 [107].
- 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 [108].
    - Runs offline so there is 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 [109].
- 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 [110], and action time.

Accuracy & Model Capacity

- MCU
  - Model size must be compressed with a smaller input size (e.g., 96-160 px) [111].
  - Supports binary classification (e.g., sitting vs. not sitting).
  - Lower accuracy and more sensitive to noise [111] (e.g., light, texture).
  - Only runs on CPU [111]
- Cloud
  - Can run high-level detection models (e.g., YOLO) and larger input sizes [112].
  - Higher accuracy and recall [112].
  - Possible GPU use instead of CPU [112].

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 preserve 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 [113].
  - Must be securely stored (e.g., encrypted) [114].

Table 25: 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.

3.3 Power Management

Power is essential to keep all the components in a product running. The electrical system defines the boundaries of performance, reliability, and safety for any embedded design. For a dog feeder, we need to consider uninterrupted functionality, and the safety of both the user and the pet. To see which power source is appropriate to use, we will look at the pros and cons as well as how to use them.

USB Power Banks

A USB power bank is a portable energy storage devide containing Li-ion or Li0poly cells with built-in charge control, protection cicuitry, and regulated 5V USB output.

USB power banks have become ubiquitous portable power sources due to their convenience and safety features. A typical bank contains multiple Li-ion pouch cells, a protection board, and a boost-regulation stage that delivers a clean 5 V output through USB-A or USB-C ports [115]. Using one is straightforward: simply plugs the device into the USB output and ensures that the downstream electronics regulate the voltage as needed. However, because they are optimized for phone charging, most power banks include

automatic load detection and may shut off if the device draws less than a minimal current threshold (often 50–100 mA) [115]. This behavior can cause intermittent power loss in ultra-low-power designs. Their voltage output is also fixed to 5V unless additional DC-DC converters are introduced.

Bench Power Supplies

A bench power supply is a precision laboratory instrument capable of delivering adjustable voltage and current with real-time monitoring and protection mechanisms.

Bench power supplies represent the precision end of the power-delivery spectrum. These instruments provide adjustable voltage and current limits with real-time monitoring and built-in protection mechanisms [116]. They are essential tools for prototyping, component characterization, and debugging hardware faults. Because bench supplies are not compact, portable, or cost-efficient, they are unsuitable for integration into final products. Their purpose is developmental rather than operational.

Wall Adapters (AC-DC adapters)

Wall adapters convert household mains AC (120/240 V) into low-voltage DC outputs such as 5 V, 9 V, or 12 V. They are widely used in consumer electronics.

AC–DC wall adapters offer an ideal primary power source for most indoor embedded systems. These adapters transform 120/240 V AC mains into isolated, safe DC voltages such as 5V, 9V, or 12V [117]. They are widely available, inexpensive, and reliable enough for continuous operation. In consumer products, they are often paired with a DC barrel jack to deliver power to an internal regulation stage. Their limitations center around dependency on mains power. During outages, the system will fail unless equipped with a backup solution [117]. Additionally, inexpensive adapters may introduce switching noise or ripple that can interfere with analog circuits.

Battery Packs

Battery packs combine multiple cells into a series or parallel configuration to produce a desired voltage and capacity.

Battery packs composed of battery cells provide an alternative, low-voltage, self-contained power source [118]. Packs can be easily assembled in series or parallel to meet voltage and capacity requirements. They are free from auto-shutoff behavior, making them more consistent than USB power banks for low-current devices. However, the voltage of primary cells decreases with discharge, requiring voltage regulation for most electronics [118]. Packs also add bulk and weight, and their capacity is lower relative to Li-ion technologies.

Switching Power Supplies (SMPS)

Switching power supplies use high frequency switching elements and inductors to convert voltages with high efficiency.

Switching power supplies form the backbone of modern embedded voltage regulation. Buck converters, boost converters, and buck-boost topologies use high-frequency switching to achieve high efficiency, low heat, and compact design footprints [119]. These

are especially common when stepping down a 12-V wall adapter output to 5 V or 3.3 V for microcontrollers. Their primary drawback is electromagnetic noise, which requires careful PCB layout, grounding techniques, and appropriate filtering [119]. Sensitive analog systems may require additional L-C filtering or shielding.

Solar Panels

Solar panels convert sunlight into usable DC electricity using photovoltaic cells. Output varies based on irradiance, angle, and temperature

Solar panels provide renewable energy for outdoor or remote IoT systems. Their output varies strongly with light intensity, temperature, and orientation, making them unsuitable for direct powering of loads [120]. Instead, a typical configuration uses a solar panel connected to a charge controller (preferably MPPT), which then charges a battery. The load draws power from the battery, ensuring stable operation. Solar energy enables long-term autonomous deployments, but it increases system complexity and demands additional space.

To see which one will be good to use in our project, we compare them.

Table 26: Types of Power Supply

Power Supply	Advantages	Disadvantages
USB Power Banks	Portable, lightweight, rechargeable. Built-in protection: overcurrent, short-circuit, thermal. Very clean 5 V output, often low ripple. Widely available at low cost.	Auto-shutoff under low-load conditions (<50–100 mA). Cannot naturally output 9–12 V without boosters. Capacity degrades with age and temperature. Many units cannot simultaneously charge and discharge.
Bench Power Supplies	Highly accurate voltage regulation. Adjustable current limiting prevents component damage. Ideal for debugging, prototyping, and electrical characterization.	Expensive compared to consumer power sources. Bulky and not portable. Not suitable as a final embedded power solution.
Wall Adapters (AC-DC adapters)	Reliable, continuous 24/7 power. Electrically isolated for safety.	Requires access to a wall outlet. System loses power during outages unless battery backup is added.

	Available in many voltage/current ratings. Perfect for stationary appliances.	Cheap adapters may cause electromagnetic noise.
Battery Packs	No auto-shutoff issues (unlike USB banks). Flexible voltage design (1.2 V–12 V+). Inexpensive and safe. Excellent for backup power.	Heavy and bulky. Voltage decreases as cells discharge. Need periodic replacement or charging.
Switching Power Supplies (SMPS)	Very efficient (>85–95%). Minimal heat generation. Supports high input voltage ranges. Available as ICs or off-the-shelf modules.	Switching noise and EMI. Layout must follow strict rules. Requires additional components (inductor, diode, capacitors).
Solar Panels	Renewable energy source. Ideal for remote or outdoor project. Reduces dependency on AC mains.	Unusable indoors or at night. Requires charge controller. Needs a battery to stabilize voltage. Larger physical footprint.

Based on the table 26, we can see the different uses of them. First, USB power bank may be convenient and easy to use. However, their power output does not meet the requirements for this project since some components need 12V power to operate. With bench power supply, its network points are quite good, but the price is too expensive and not necessary for this project. Next, wall adapter is a stable choice to maintain the product with enough power to provide continuous and smooth operation. Moreover, battery pack seems to be a good choice, but if the main power source is only used for them, we will need a lot of backup battery cells. It will be very bulky and cannot ensure continuous power when the battery cells run out of energy. In addition, switching power supply is not a good choice either because it needs to be carefully built, and requires a lot of accompanying components. Finally, solar panels should not be used because our project is an indoor product. They almost only use electricity from outside.

After considering, we decided to use a **wall adapter** to convert AC power to DC for use.

3.3.1 Wall Adapter

Rather than build an off-line AC-DC converter on the PCB (which would require high-voltage layout, creepage/clearance, and regulatory testing), We assume the dog feeder will use a low-voltage external adapter (a wall plug-in or small desktop brick). This keeps high voltage outside the enclosure and lets the PCB focus on low-voltage DC distribution (12 V bus, 5 V and 3.3 V regulators, backup battery).

Linear vs. Switching Plug-In Adapters

Older wall warts often use a linear transformer plus rectifier, while modern adapters are almost all switch-mode (SMPS).

Table 27: Types of Wall Adapter

Feature	Linear Plug-In Adapter	Switching (SMPS) Plug-In Adapter
Efficiency	Low (often 40–60 %)	High (upwards of ~80–90 %)
Size & Weight	Bulky, heavy transformer	Compact, lightweight
No-load Power	Relatively high	Very low (meets modern Level VI limits)
Output Regulation	Often poor; varies with load/line	Tight regulation over load and input range
Heat Dissipation	Runs warm at moderate power	Cooler for same output power
EMI	Very low EMI	Higher EMI, but certified designs pass compliance
Cost	Simple construction, sometimes lower	Mass-produced SMPS now often competitive on price
Availability at 12 V/3 A	Less common	Very common
Suitability for Dog Feeder	Acceptable but inefficient	**Preferred**: efficient, compact, widely available

According to Triad magnetics, LEDsupply, Mouser, we can see the different between two types of adapters in table 27 [121, 122, 123]. For 12V adapter, a linear design would be bulky and hot. A switching adapter is small and efficient. For a 24/7 plugged-in dog feeder, efficiency and low no-load power matter. Therefore, a **switching plug-in adapter** is our choice

Unregulated vs. Regulated DC

Some old 12 V adapters are unregulated: under light load they can output >16 V, and only approach 12 V at rated current. Modern switch-mode adapters provide regulated 12 V over most of the load range.

Unregulated 12 V could overstress downstream buck converters and the motor driver during light load.

For this design, we assume **regulated 12 V DC** only.

Fixed-Voltage Wall Adapter vs. USB-C PD vs. Desktop Brick

There are some types of chargers which we need to consider:

Table 28: Types of Chargers

Feature	12V Wall-Mount Adapter	USB-C PD Charger + PD Sink	Desktop Brick
Output Type	Fixed 12 V DC	Negotiable (5–20 V) via PD protocol	Fixed 12 V DC
Complexity on PCB	Very low (just barrel jack & filtering)	High: PD sink IC, CC negotiation, etc.	Low (similar to wall-mount)
Connector	Barrel (e.g., 2.1 mm)	USB-C	Barrel (2.1–2.5 mm)
Availability at 12V/ 3A	Very common	Common, but PD sink design adds work	Common GST series bricks
Cost & BOM Impact	Low total cost	Higher total (charger + PD sink BOM)	Slightly higher than wall wart
User Familiarity	Very familiar	Also familiar, but expects USB devices	Common for laptops/routers
Mechanical Integration	Adapter hangs from outlet	Charger + cable; feeder still needs jack	Brick sits on floor or shelf

Based on the comparison in the table 28, we can see all options will be good [124, 125, 126]. However, USB-C PD charger + PD sink option is overkill unless we need PD flexibility. The desktop brick is possible, but the wall-mount is simpler. For this project, we prefer the **12V fixed-voltage wall adapter** because it gives us exactly what we want (12v DC at a few amps) without having to implement USB-C PD negotiation or deal with non-standard chargers.

Now, we look at concrete COTS parts that meet the 12V, around 3A requirement and are realistically available from major distributors.

CUI Inc. SMI36-12-V-Px (Wall-Mount, 36 W)

CUI's **SMI36 series** are 36 W, DOE Level VI, single-output wall-mount AC-DC adapters [127]. The 12 V variant (e.g., SMI36-12-V-P5 or SMI36-12-V-P6) provides **12 V at 3 A**, accepts 90–264 VAC, and offers 2.1 mm or 2.5 mm barrel plugs with interchangeable AC blades [127].

Advantages:

- Exactly matches 12 V/3 A requirement.
- High efficiency, Level VI compliant (low standby power).
- Interchangeable blades are handy if the product ever leaves North America.
- Barrel connector options (2.1 mm or 2.5 mm) match standard jacks.

Disadvantages:

- Slightly higher cost than very generic, no-name adapters.
- Interchangeable blades are mechanically more complex than fixed US plug (but also more flexible).

Mean Well GS36U12-P1J (Wall-Mount, 36 W)

Mean Well's **GS36U12-P1J** is a 36W, 12V, 3A AC-DC switching wall adapter [128]. It uses a 2.1 mm x 5.5 mm center-positive barrel plug and supports global 85–264 VAC input [128].

Advantages:

- 12 V/3 A output, perfect for the dog-feeder budget.
- Mean Well has a strong reputation for reliability and safety.
- Compact design, efficiency comparable to modern green supplies.

Disadvantages:

- Often slightly higher price than no-name brands.
- Availability can vary by region and distributor.

Facmogu 12V 3A Adapter

Facmogu 12V 3A Adapter is a 36W, a wide range 100-240V AC to DC 12V 3A, offers 2.1 mm or 2.5 mm barrel plugs [129].

Advantages:

- Matches the dog-feeder requirement (12 V, 3 A, 36 W).
- Works with 100–240 V worldwide.
- Cheaper than CUI/Mean Well industrial supplies.
- Barrel connector options (2.1 mm or 2.5 mm) match standard jacks.
- Listing claims OVP/OCP/OTP protection.

Disadvantages:

- Minimum headroom: 3 A is barely enough for motor + electronics peak loads.
- Less detailed safety/efficiency documentation than CUI/Mean Well.

- Listing warns against high-power loads: Indicates limited robustness
- Now, we compare them.

Table 29: Types of Adapters

Feature	Facmogu 12V 3A Adapter	Mean Well GS36U12-P1J (36 W)	CUI SMI36-12-V-Px (36 W)
Output Voltage / Current	12 V / 3 A	12 V / 3 A	12 V / 3 A
Power Rating	36 W	36 W	36 W
Headroom for Motor Peaks	Minimal (just 3 A)	Minimal	Minimal
Barrel Connector Fit	Flexible (two barrel sizes)	Specified barrel size	Specified barrel size, known connector
Cost / Availability	Likely lower cost	Slightly higher cost but trade-off in reliability	Slightly higher cost but trade-off in reliability

Looking at the comparison information in Table 29, all three models have quite similar information. However, for **Facmogu 12V 3A adapter**, it meets the basic voltage/current requirement (12 V/3 A) and thus could function in the system. It might be significantly cheaper than premium brands and thus better for a student project budget. The flexible connector is also convenient if there is uncertainty about barrel size.

3.3.2 Power Jack

A DC power jack (barrel jack) is a cylindrical, polarized connector that accepts a matching DC plug, typically with an inner "pin" and outer sleeve. Typical adapters use diameters like 2.0 mm or 2.1 mm inner (ID), and 5.5 mm outer (OD).

Standard open-frame PCB barrel jacks

Standard open-frame PCB barrel jacks are available in various sizes, with the most common being 5.5mm outer diameter (OD) with either a 2.1mm or 2.5mm inner diameter (ID) center pin [130]. They are primarily used for low-voltage DC power input, typically ranging from 5V to 24V DC [131].

Sealed or locking barrel jacks

Sealed barrel jacks are a type of DC power connector that incorporate special design features, such as rubber gaskets or O-rings, to create a barrier against environmental contaminants like moisture, dust, and chemicals [132]. They are designed for applications in harsh environments where the integrity of the electrical connection must be protected from the elements and often carry an Ingress Protection (IP) [132].

Locking barrel jacks are a variation of the standard DC power connector that feature a mechanical mechanism (such as a threaded collar or a bayonet-style twist-lock tab system) to prevent accidental disconnection [133]. The plug and jack are designed to physically interlock, providing a secure, high-retention connection that resists accidental tugging, vibration, or movement that would easily disengage a standard friction-fit barrel jack [133]. They are used in applications where an uninterrupted power supply is critical, such as medical devices or portable equipment that may be subject to physical stress.

Non-barrel alternatives

Non-barrel alternatives are types of electrical power connectors that deviate from the standard cylindrical, center-pin design of a barrel jack. These alternatives use various mechanical configurations to provide power input, often offering specific advantages such as higher current ratings, more secure connections, standardized interfaces, or environmental sealing [130].

For this dog feeder, the power jack must:

- Interface with a 12 V, ~3 A wall adapter.
- Provide a reliable, low-resistance connection under motor load.
- Mount securely on the edge of the main PCB, aligning with an enclosure cutout.
- Handle thousands of insertion cycles over the product's life.

Table 30: Types of Barrel jacks

Feature	Standard open-frame PCB barrel jacks	Sealed or locking barrel jacks	Non-barrel alternatives
Pros	Cheap, widely available; easy to mate with 12 V adapters. Good EDA models and footprints; 2–3.5 A ratings common.	Locking mechanism prevents accidental disconnect. High current rating (~5 A); robust build; long life (5000+ cycles).	Simple, robust clamping of bare wires. No special plug format. Can handle high currents.
Cons	Can be unplugged accidentally. Not sealed. Current rating must be checked vs motor load.	More expensive. Special mating plug. Slightly more mechanical complexity and height.	Less user-friendly (requires screwdriver). Easier to mis-wire. No quick-connect feel.

Fit for Dog Feeder	Very good baseline choice	Great for mission-critical devices Arguably overkill for student dog feeder	OK for internal wiring. Not ideal for "user-facing" power input

For a consumer-style dog feeder that uses a standard 12 V adapter, a **PCB barrel jack** is the most natural technology: users already expect to plug in a barrel plug, and both 12 V adapters and barrel jacks are commodity components. Sealed/locking jacks are attractive but increase cost and require matching cables

After knowing which kind of barrel jack we use, we also need to pick the model for it.

CUI / Same Sky PJ-102A

The PJ-102A is an unshielded, 2-conductor jack designed to accept a standard 5.5mm OD, 2.0mm ID DC power plug [134]. Its right-angle orientation and through-hole pins provide a physically robust connection to a printed circuit board (PCB) [134].

Kycon KLDX-0202-AC

The Kycon KLDX-0202-AC is a specific model of a right-angle, through-hole DC power barrel jack designed for PCB mounting in applications requiring a reliable power connection with a standard 5.5mm outer diameter and a 2.0mm inner diameter center pin [135].

Switchcraft BKZ Series Locking Jack

The Switchcraft BKZ series is a family of right-angle PCB-mount locking DC barrel jacks [136]. Unlike standard jacks, BKZ jack mate with a matching plug that locks into the jack via wings and slots, preventing accidental disconnection. They are rated for up to 5 A at 24 V and at least 5000 insertion cycles

Now, we compare them to see which fits better for our project.

Table 31: Types of Power jacks

Feature	CUI / Same Sky PJ-102A	Kycon KLDX-0202-AC	Switchcraft BKZ Series Locking Jack
ID / OD	2.0 mm / ~5.5 mm barrel family	2.0 mm ID / 5.5 mm OD	2.0 mm or 2.1 mm options, DC barrel family
Voltage / Current rating	~16 V DC @ 2.5 A	~24 V DC @ 3.5 A (variant dependent)	24 V @ 5 A

Mounting style	Right-angle, through-hole PCB	Right-angle, through-hole PCB with kinked pins	Right-angle PCB; through-hole, SMT, hybrid options
Special features	Simple open-frame jack	Some variants include switch contact	Locking bayonet mechanism, high-current design
Mechanical robustness	Good for standard duty	High (kinked pins, higher rating)	Very high; designed for critical equipment
Cost	Low	Medium	High

Based on Digikey, Switchcraft, we make Table 31 to see the difference between them [134, 135, 136]. After comparing all options, we intentionally choose the **PJ-102A** as the final power jack for this dog-feeder project. Electrically, it provides the right balance of capability and practicality: the feeder's average current at 12V is typically under 2A, with short 3A peaks during stepper acceleration, and the PJ-102A's 16V at 2.5A rating is sufficient once realistic duty cycle, derating, and conservative motor tuning are accounted for. Mechanically, its right-angle through-hole footprint is well-supported across ECAD libraries (including Fusion 360making schematic and PCB design smooth and predictable, while the through-hole pins safely transfer insertion forces into the board instead of stressing solder joints. It also fits into the mass-market 2.0–2.1 mm / 5.5 mm barrel ecosystem, meaning it pairs cleanly with nearly all common 12 V adapters. Alternatives such as the Kycon KLDX-0202-AC or Switchcraft BKZ series offer higher current capability or locking mechanisms, but they introduce extra cost and complexity that a student-level dog feeder does not need. In the end, the PJ-102A may not be the flashiest connector, but it delivers reliable power input without inflating the BOM or complicating the design.

3.3.3 Voltage Regulators

A voltage regulator is an essential electronic component designed to automatically maintain a constant output voltage, regardless of fluctuations in the input power source or changes in the electrical load the circuit demands. Its core function is to stabilize power delivery, acting as a crucial intermediary between the raw power supply and the sensitive electronic components of a system.

In a dog feeder project, the voltage regulator plays a critical role in ensuring reliable and safe operation. Power sources such as battery packs exhibit varying voltage levels, and wall adapters may have slight line noise or variances. A regulator steps this input power down to the precise, stable voltage levels required by components like the main microcontroller (MCU), various sensors, and wireless communication modules (typically 3.3V or 5V).

The need for a voltage regulator stems from the potential for damage and operational errors if components receive unstable power. Without regulation, a dropping battery voltage can cause the MCU to reset unexpectedly, corrupt settings, or misinterpret sensor data, leading to incorrect feeding times or jammed motors. Furthermore, the high current demands and

electrical noise generated by the motors when they operate can interfere with or even damage sensitive digital logic. The regulator isolates the control circuitry from these power fluctuations, guaranteeing the consistent and safe operation of the automated dog feeder.

Linear Regulators

Linear regulators are a simple type of voltage regulator that produce a stable, lower output voltage by dissipating excess electrical energy as heat [137]. They function by using an internal transistor as a variable resistor to maintain a constant output level, effectively burning off any surplus input voltage [137]. While simple to implement, inexpensive, and providing a very clean, low-noise power output, their main drawback is low efficiency [137]. The significant power loss as heat makes them best suited for low-power applications or situations where the input and output voltages are close together.

Buck Regulators

- A Non-Synchronous Buck Regulators is a type of switching voltage converter that efficiently steps a higher DC input voltage down to a stable, lower DC output voltage [138, 139]. Unlike linear regulators, it uses a rapid switching action with an inductor and capacitor to achieve much higher efficiency [138]. The key feature is the use of a simple rectifier diode to manage current flow during the switching cycle [140]. While this design is cost-effective and simpler than more advanced converters, the power lost as heat across the diode means it is less efficient than a synchronous buck converter, making it suitable for applications where moderate efficiency and low cost are prioritized.
- High-Current Simple Switcher Buck Regulator is a family of highly integrated step-down switching regulator ICs from Texas Instruments designed for efficient power conversion up to 5A or more. The Simple Switcher design philosophy integrates the power switch, oscillator, and compensation into one chip, requiring minimal external components and simplifying the design process. These regulators achieve efficiencies exceeding 90% and often feature a wide input voltage range, making them a robust and easy-to-implement solution for high-power applications where both efficiency and design simplicity are necessary [141].

Off-the-shelf Buck Modules

Off-the-shelf buck modules are pre-assembled, ready-to-use power modules that integrate all necessary components of a step-down switching regulator onto a single PCB. They offer a simple solution, requiring minimal design effort beyond connecting the input and output wires. These modules provide a highly efficient, compact, and cost-effective alternative to designing a custom power circuit or using inefficient linear regulators, making them ideal for rapid prototyping or projects where design simplicity and proven reliability are prioritized [142].

Table 32: Types of Regulators

Feature	Linear Regulators	Non-Synchronous	High-Current Simple	Off-the-shelf Buck Modules

		Buck Regulators	Switcher Buck Regulators	
Typical Efficiency (12→5 V)	~42%.	~70–85 %.	~85–92 %.	Similar to IC used.
Current Capability	Up to a few A (with huge heatsink).	Up to ~3A.	Up to 5A.	Often 2–3A typical.
Thermal Behavior	Enormous heat: (12–5)·I, up to 35W at 5A.	Moderate heat, manageable at 2–3A.	Several watts of loss at 5A, heat sinkable.	Depends on module quality.
PCB Complexity	Very simple.	Moderate.	Moderate.	Very easy.

Based on the comparation in Table 36, a **buck regulator** is the good choice. It keeps heat manageable, delivers the current headroom we want, and fits nicely into existing simple switcher-style design language.

Now, we need to pick the model part for regulator. Because the chances of success are very low when using TPS and BQ models, we will focus on choosing LM series for regulator.

There are three candidates for LM series.

Table 33: Models of Regulators

Feature	LM2596	LM2576	LM22678
Topology	Non-synchronous buck.	Non-synchronous buck.	Non-synchronous buck.
Vin range	4.5–40 V.	8–40 V.	4.5–42 V.
Vout option	Fixed 5V available.	Fixed 5V available.	Fixed 5V available.
Iout max	3A.	3A.	5A.
Typical efficiency (12→5 V)	~75–85% (design-dependent).	~85–92% (design-dependent).	~85–92% (design-dependent).
Layout difficulty	Low–moderate.	Moderate (comfortable for 2-layer).	Moderate–high (more EMI care).
Fit for 5V at 3A dog feeder	Undersized.	Excellent, recommended.	Excellent, but more layout-sensitive.

Based on 3 models in Texas Instruments, we can see the information in Table 37 [143, 144, 145]. **LM2576** is the best choice for our design since it's not overkill and easy to use.

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 animal's 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 [146].

UL 94 summarizes the flammability of the different plastic materials that are in our design [147]. 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 [147].

ANSI/FM 4910 elaborates on the cleanroom materials and the test protocol for flammability [148]. 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 [148]. 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.

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 [149]. 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 [149]. 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.

ANSI Z535 is a series of different safety signs with colors and labels being emphasized [150]. 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 [150]. 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.

ANSI UL 60730-1 summarizes the electrical controls within the household when they are applied automatically [151]. 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 [151]. 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.

4.1.3 Mechanical safety

ISO 12100 is a standard for risk assessment of different machinery and other hazards that could potentially arise [152]. 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 [152]. 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.

ASTM B117 is a standard for the evaluation of how resistant a part is to corrosion [153]. 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.

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 [154]. 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 [154]. In relation to our feeder, we will 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.

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 [155]. 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 [155]. 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.

ISO/IEC 29119-1:2022 is a global standard for software testing processes, documentation, and techniques [156]. It defines structured phases such as test planning, design, execution, and closure to ensure that software components meet their intended requirements [156]. 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.

4.1.6 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 [157]. 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 [157]. Following these guidelines ensures consistent signal range, throughput, and reliability for remote feeder control.

4.1.7 Cloud Security

ISO/IEC 27017 provides additional guidelines for information security in cloud computing environments, extending ISO 27002 with cloud-specific controls [158]. 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 [158]. 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 [159, 160]. 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.

ISO/IEC TR 24028:2020 provides guidelines for evaluating the robustness and trustworthiness of AI models [161] [162]. It focuses on adversarial resistance, interpretability, and model reliability [162] [161]. 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.

ISO/IEC 30170:2012 standardizes Python's syntax and behavior, ensuring that software developed in Python is portable and consistent across platforms [163]. This is relevant for the backend model server running the posture detection model in Python [163]. Adhering to this standard ensures code interoperability, predictable behavior during inference, and compatibility with other standardized Python environments, helping maintain a stable backend system [163].

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 [164]. 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. The owner will be given an alert through our mobile app that communicates that the storage bin is running low to create 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 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 cannot 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 will 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 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 [28]. The OV5640 also increases the reliability of our feeder from the extended peripherals that it has. It can be easily integrated with the MCU, which will allow for connection and communication between the two to be easy and reliable [28, 20]. It also has an extensive environment for learning how to use and operate the camera and the MCU, allowing us to focus more on how we want the camera to operate and "see" instead of working on simple tasks to initialize the camera. Overall, reliability of the camera to MCU connection and the reliability of the feeder as a whole will be improved.

Since our feeder operates without the owner having to be present during feeding times, another constraint that we have is making sure that the dog receives food and the dog is safe 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. Based on ISO 12100:2010, 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 [152]. 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.

According to Texas Instruments, based on IEC/UL 60730-1/60335-1 standard, 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 [149]. 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

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 is a real advantage when building a project that might run on limited power or batteries. It only needs a single digital signal from the microcontroller, which would be helpful when we arere still prototyping and do not 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 we noticed is that it doesn't mention the sound enclosure or real-world placement much (like the way the speaker is mounted affects the clarity and volume).

The plus side of the answers is that they tell our team where to start or what types of speakers we need to investigate (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 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.

5.5 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

This schematic is a very simple LED power indicator circuit connected to a 12 V supply. It is used to show visually when the system (such as your dog feeder project) is powered on.

The circuit begins with a DC input jack on the left side. The jack provides two main connections: +12 V and ground (GND). When a power supply is plugged in, the 12 V line becomes the main positive rail for the circuit, while the ground line serves as the common reference. These two lines are also routed to connectors on the right side of the schematic, making the 12 V and GND available to other parts of the system.

From the 12 V line, a branch goes through a series resistor R1 and then into LED1 before returning to ground. This creates a current path: 12 V → R1 → LED → GND. When power is applied, current flows through this path, causing the LED to light up. The LED acts as a visual indicator that voltage is present on the 12 V rail.

The resistor R1 is essential because it limits the current flowing through the LED. Without it, the LED would draw too much current from the 12 V supply and could burn out. The value of R1 is chosen based on the LED's forward voltage and desired current. For example, assuming a typical LED forward voltage of about 2 V and a desired current of around 10–20 mA, a resistor in the range of 470 Ω to 1 kΩ would be appropriate for a 12 V supply.

After entering the board, the 12 V supply is split into three separate branches to support different subsystems. One branch is routed to the 3.3 V voltage regulator, which steps the voltage down to power low-voltage digital components such as microcontrollers or sensors. A second branch is directed to the 5 V voltage regulator, which provides power for components that require a higher logic level or intermediate voltage. The third branch supplies power directly to the motor, which typically requires higher voltage and current than logic circuitry. This distribution approach ensures that each part of the system receives the appropriate voltage level while sharing a common power source.

In summary, this circuit is a straightforward power indicator and distribution node. The DC jack brings in 12 V power, the LED and resistor provide a visual confirmation that power is present, and the connectors distribute that power to other components in the project.

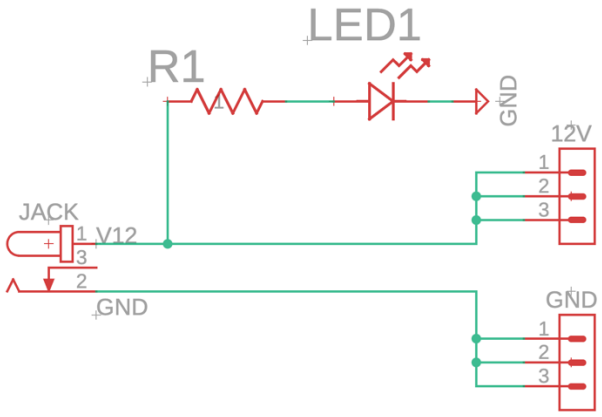

Figure 6: Power Jack board Schematic

6.1.2 Voltage Regulators

The circuit shown in the schematic is a 12 V to 3.3 V step-down (buck) switching regulator built using the LM2576HV-ADJ integrated circuit. This device, according to the Texas Instruments datasheet, is a monolithic regulator capable of delivering up to 3 A of load current while operating at a fixed switching frequency of approximately 52 kHz. The “HV” version allows operation at higher input voltages, making it suitable for a 12 V supply. Unlike linear regulators, the LM2576 achieves high efficiency by rapidly switching the input voltage and filtering it through external components, which significantly reduces power dissipation.

To construct the circuit, the input stage must first be established. The 12 V supply is connected to pin 1 (VIN) of the LM2576, while pin 3 is connected to ground. A bulk input capacitor (CIN), typically around 100 µF, is placed between VIN and ground as close to the IC as possible. This capacitor stabilizes the input voltage and reduces noise caused by the switching action. Proper placement of this capacitor is critical, as emphasized in the datasheet, because it minimizes voltage spikes and improves overall regulator stability.

The switching and energy storage stage consists of the internal switch (pin 2, OUTPUT), the inductor (L1), and the Schottky diode (CR1). The output pin of the LM2576 connects directly to one side of the inductor and to the cathode of the diode. The anode of the diode is connected to ground. During operation, the regulator rapidly switches the output pin on and off, allowing current to build up in the inductor and then flow through the diode when the switch is off. The inductor, typically around 100 µH with a current rating above the expected load, stores energy and smooths the current, while the Schottky diode provides a low-loss path for current during the off-cycle. The datasheet recommends selecting a diode with a current rating at least 1.2 times the load current and a reverse voltage rating higher than the input voltage.

The output filtering stage is formed by the inductor and the output capacitor (COUT). The capacitor is connected from the output node (after the inductor) to ground and is usually in the range of 680 µF to 1000 µF. This capacitor reduces voltage ripple and ensures a stable DC output. The combination of L1 and COUT acts as a low-pass filter, converting the

pulsed waveform from the switching regulator into a steady 3.3 V output suitable for powering electronic components.

Voltage regulation is achieved through the feedback network connected to pin 4 (FB). The LM2576 adjustable version uses an internal reference voltage of 1.23 V, and the output voltage is set using a resistor divider composed of R1 and R2. R1 is connected from the feedback pin to ground, while R2 is connected from the output to the feedback pin. The relationship between these resistors and the output voltage is given by $V_{OUT}=1.23(1+R2/R1)$. Using 1% tolerance resistors is recommended to maintain accurate voltage regulation. The feedback trace should be kept short and away from noisy switching nodes to avoid instability.

The enable control (ON/OFF pin), which is pin 5, determines whether the regulator is active. In this circuit, it is connected directly to ground, which keeps the regulator enabled at all times. The datasheet specifies that this pin must not be left floating, as doing so can lead to unpredictable operation. Pulling the pin high would disable the regulator if shutdown functionality were required.

Finally, additional components such as the LED and its series resistor (R3) provide a simple output indicator, showing when the 3.3 V rail is active. When assembling the circuit, careful attention must be given to PCB layout, as switching regulators are sensitive to noise and parasitic inductance. The datasheet highlights that high-current loops—particularly those involving the input capacitor, diode, and switching pin—should be kept as short and compact as possible. Ground connections should be solid and low impedance, and sensitive nodes like the feedback pin should be routed away from switching paths. Following these layout practices ensures stable operation, reduced electromagnetic interference, and optimal performance of the regulator.

Since we use LM2576-ADJ voltage regulators, we just need to change values of R1 and R2 to get the expected output as we want.

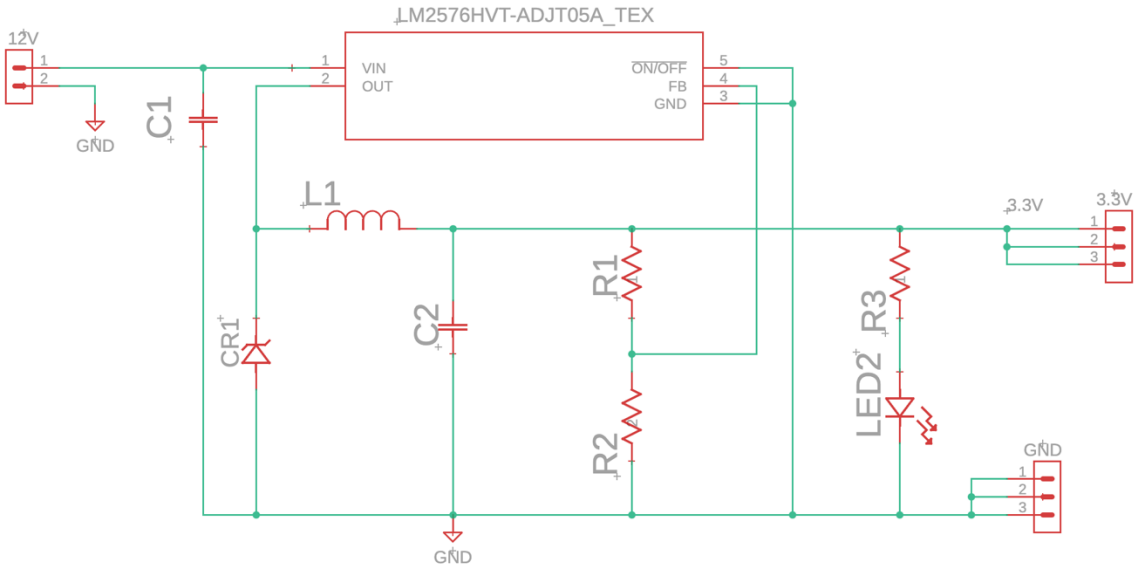

Figure 7: 3.3V Voltage Regulator Schematic

The same LM2576-ADJ regulator can be used to generate a 5 V output from a 12 V input by keeping the overall circuit topology identical to the 3.3 V design and only modifying the feedback network. The device operates as a buck (step-down) switching regulator, meaning it converts a higher DC input voltage into a lower regulated output by rapidly switching and filtering the energy through an inductor and capacitor. Because of this, the connections for the input stage, switching components, and output filter remain unchanged when moving from 3.3 V to 5 V.

To build the 5 V version, begin by connecting the 12 V input supply to the VIN pin (pin 1) of the LM2576 and connecting pin 3 to ground. A 100 µF input capacitor should be placed between VIN and ground as close to the IC as possible to stabilize the input and reduce switching noise. The ON/OFF pin (pin 5) is tied directly to ground so that the regulator is always enabled. This ensures continuous operation without requiring an external control signal.

Next, construct the switching and energy transfer section. The output pin (pin 2) connects to one side of the inductor and to the cathode of the Schottky diode. The diode's anode is connected to ground. A 100 µH inductor, rated for at least the expected load current (typically ≥3 A), is used to store and transfer energy during the switching cycles. The diode must be a fast Schottky type, such as the 1N5822 or MBR360, to minimize losses and handle the current during the off-cycle of the switch.

The output stage consists of the inductor and the output capacitor. The other side of the inductor becomes the 5 V output node, and a 680 µF to 1000 µF capacitor is connected from this node to ground. This capacitor smooths the voltage and reduces ripple, ensuring a stable DC output suitable for powering digital circuits. For a 5 V output, the capacitor should have a voltage rating of at least 10 V, though using a 16 V rating improves reliability.

The key difference between the 3.3 V and 5 V configurations lies in the feedback resistor divider connected to the FB pin (pin 4). The LM2576-ADJ regulates its output by maintaining a fixed internal reference voltage of 1.23 V at the feedback pin. The output voltage is therefore determined by the ratio of two resistors according to:

$$V_{OUT}=1.23(1+(R2/R1))$$

To achieve a 5 V output, a typical and recommended selection is R1 = 3.6 kΩ (from FB to ground) and R2 =11 kΩ (from output to FB). Using standard 1% resistor values such as 11 kΩ or 3.6 kΩ will produce an output very close to 5 V. This resistor divider senses the output voltage and feeds it back to the regulator, allowing it to adjust the switching duty cycle and maintain a stable output.

Overall, converting the design from 3.3 V to 5 V is straightforward: the same LM2576-ADJ circuit and components are used, and only the feedback resistor values are changed. Maintaining proper PCB layout is still critical—especially keeping the high-current switching loop (input capacitor, diode, and regulator) short and compact, and routing the feedback line carefully to avoid noise. Following these practices ensures efficient, stable, and reliable operation of the 5 V regulator.

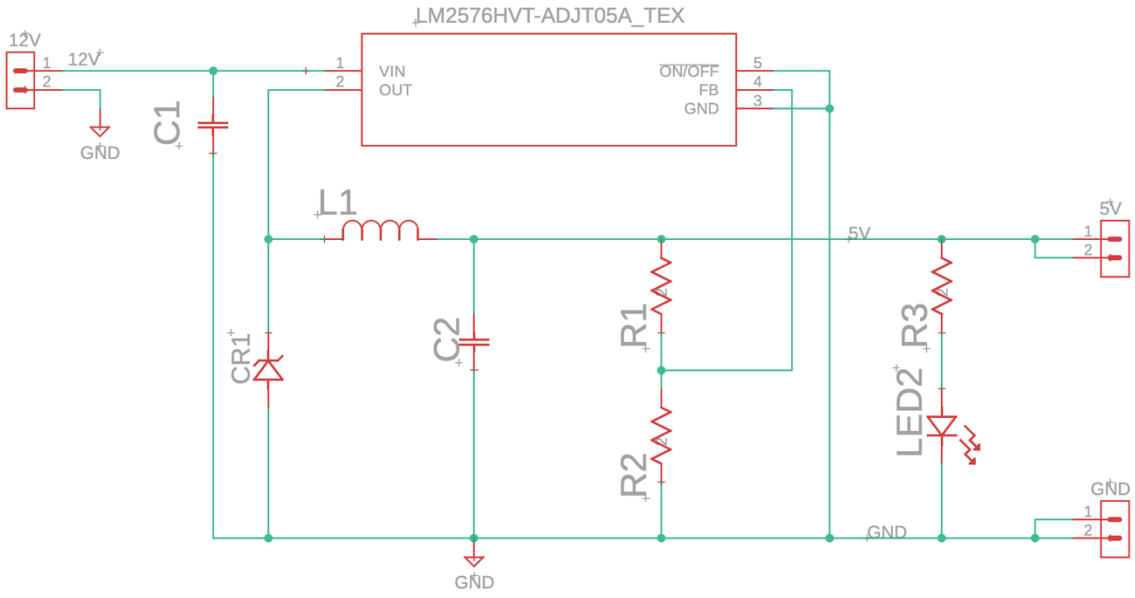

Figure 8: 5V Voltage Regulator Schematic

6.2 Programmer

The schematic is a CP2102 USB-to-UART programmer circuit. It is used to convert the computer's USB signal into serial UART signals, so a microcontroller in the dog feeder project can be programmed or debugged through TXD, RXD, GND. According to the Silicon Labs datasheet, the CP2102 is a USB 2.0 full-speed bridge device with an internal

USB transceiver, oscillator, EEPROM, and UART interface, so only a small number of external parts are needed.

To build the circuit, first connect the USB connector to the CP2102. The USB VBUS 5 V line goes to the power input side of the circuit, while GND connects to the common ground. The USB data lines connect directly to the CP2102 USB pins: D+ from the USB connector goes to D+, and D− goes to D−. These two traces should be routed close together and kept short because they carry the USB communication signal. The datasheet shows that the CP2102 is designed to connect directly to the USB bus, so no external USB controller is required.

The power section is made using the CP2102 internal regulator. In the schematic, the USB 5 V line is connected to REGIN, which supplies the internal voltage regulator. The regulator output, VDD, provides the 3.3 V supply used by the CP2102. A capacitor should be placed from VDD to ground to stabilize this voltage, and another capacitor should be placed near REGIN to reduce noise from the USB supply. These capacitors should be physically close to the CP2102 pins because they prevent voltage dips and improve reliable USB operation.

The UART side of the circuit connects the CP2102 to the external programming header. The CP2102 TXD pin should connect to the target microcontroller's RXD pin, and the CP2102 RXD pin should connect to the target microcontroller's TXD pin. This crossing is required because one device's transmitter must connect to the other device's receiver. The header should also include GND, because both the programmer and the target board must share the same reference voltage. If the target board uses 3.3 V logic, the CP2102 UART pins can communicate directly.

The schematic also includes handshake/control pins such as DTR, RTS, CTS, DSR, DCD, and RI. These pins are optional for basic serial programming, but they can be useful if the microcontroller needs automatic reset or bootloader control. For a simple dog feeder programmer, the most important signals are usually TXD, RXD, GND. If automatic reset is needed, DTR can be routed through a small capacitor to the microcontroller reset pin. However, in this case, we don't need DTR

The output header at the bottom of the schematic provides the signals that will connect to the dog feeder controller board. The TXD and RXD lines are the serial communication lines, GND is the shared ground, and the voltage pins can be used to power or reference the target circuit. Care must be taken not to connect 5 V to a microcontroller that only accepts 3.3 V. For the safest design, the CP2102 programmer should use 3.3 V UART logic when connecting to common microcontrollers such as ESP32 or other 3.3 V devices.

When making the PCB, place the CP2102 close to the USB connector. Keep the USB D+ and D− traces short, parallel, and away from noisy power traces. Place the decoupling capacitors close to the VDD and REGIN pins. The ground connections should be solid and low resistance. After assembly, install the Silicon Labs CP210x VCP driver if needed,

and the device should appear on the computer as a virtual COM port for programming and serial monitoring.

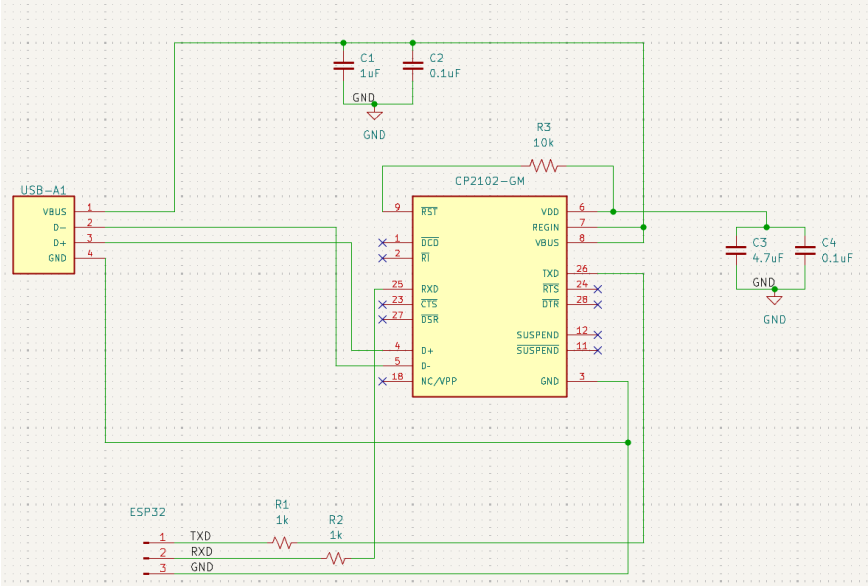

Figure 9: CP2102 Programmer Schematic

6.3 Amplifier

The circuit in schematic is a small LM386 audio amplifier used to drive a speaker from a low-power audio signal, such as a microcontroller GPIO audio output. The LM386 is designed for low-voltage audio applications and needs only a few external parts. According to the TI datasheet, it can operate from about 4 V to 12 V for most versions, has low quiescent current, and its default voltage gain is 20 when pins 1 and 8 are left open.

To make the circuit, connect the 5 V supply from the input header to pin 6 (VS) of the LM386, and connect the circuit ground to pin 4 (GND). Place the 0.1 µF capacitor C1 from the supply line to ground close to the LM386. This capacitor helps filter power-supply noise and prevents unstable amplifier behavior. The datasheet recommends keeping the supply well regulated and placing a capacitor close to the device.

The audio signal from the input header goes into the amplifier input. In this schematic, the signal line passes through R1, a 10 kΩ resistor, and then connects to the LM386 input. The LM386 has two input pins: pin 3 is the non-inverting input and pin 2 is the inverting input. For this single-ended amplifier design, one input receives the audio signal and the other input is connected to ground. This lets the LM386 amplify the incoming sound signal using ground as the reference.

Pins 1 and 8 control the gain. In your schematic, they are not connected together with a capacitor, so the amplifier uses its default gain of 20, or about 26 dB. This is good for a simple speaker output because it keeps the circuit stable and reduces noise. If more volume were needed, a 10 µF capacitor could be placed between pins 1 and 8 to increase the gain up to 200, but that may also increase distortion and noise.

The amplified output comes from pin 5 (VOUT). Because the LM386 output is biased internally at about half of the supply voltage, the speaker cannot be connected directly to pin 5. Instead, the schematic uses C3, a 250 µF electrolytic capacitor, in series between pin 5 and the speaker output. This capacitor blocks DC voltage and allows only the AC audio signal to reach the speaker. The positive side of the capacitor should face the LM386 output, and the other side goes to the speaker connector.

The circuit also includes C2 and R2 connected from the output to ground. This is a Zobel or snubber network, commonly used with audio amplifiers to improve stability when driving a speaker. In your schematic, C2 is 0.05 µF and R2 is 10 Ω, which matches the typical LM386 application style shown in the datasheet. This network helps reduce high-frequency oscillation and keeps the amplifier stable with the speaker load.

Finally, connect the speaker between the output connector and ground. The LM386 is intended for small speakers, commonly 4 Ω to 32 Ω, so an 8 Ω speaker is a good choice for this circuit. For PCB layout, keep the LM386 close to the speaker output capacitor, place the power capacitor near pin 6, and use a shared ground connection to reduce hum and noise.

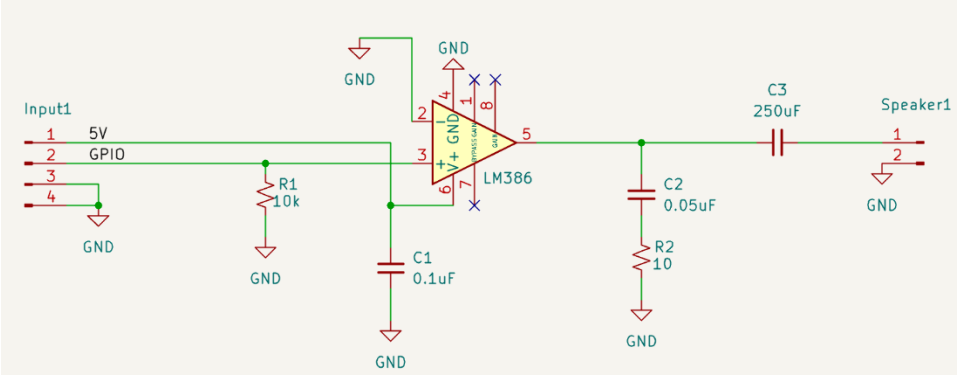

Figure 10: LM386 Amplifier Schematic

6.4 Motor Driver

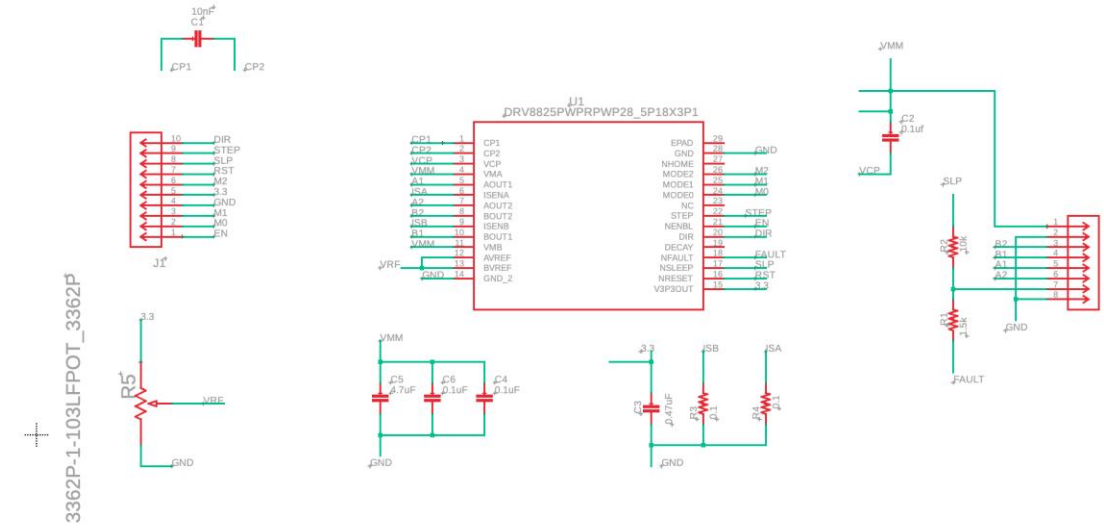

Figure 11: DRV8825 Motor Driver Schematic

The DRV8825 was the motor driver we chose to create as a separate PCB that would connect to the MCU PCB using two 10 pin jst xh pin headers. Routing from the mcu to the DRV8825 chip were the DIR, STEP, SLP, RST, M0, M1, M2, EN, 3.3V, and ground. Ultimately, we only routed 7 of the pins from the mcu to the DRV8825 chip since we were able to reroute the M0, M1, and M2 pins. Both M0 and M1 were routed directly to 3.3V using a soldered wire and M2 was routed to ground using the same method. This allowed us to free up specific pins that could then be used for other peripherals if needed. The DRV8825 contains various resistors and capacitors along with a potentiometer that allowed us to raise or lower the amount of current that would run through the motor.

Functionally, the motor driver will be used to operate the motor that controls the dispensing of the dog's food when the feeder has recognized that the dog has successfully sat down. Which we placed along the left wall in a holding spot that secured the PCB and the wires. Since the header connection for the motor driver was located on the left side of the pcb as well it allowed for a secure connection that didn't have to run through the entire feeder to connect.

6.5 MCU

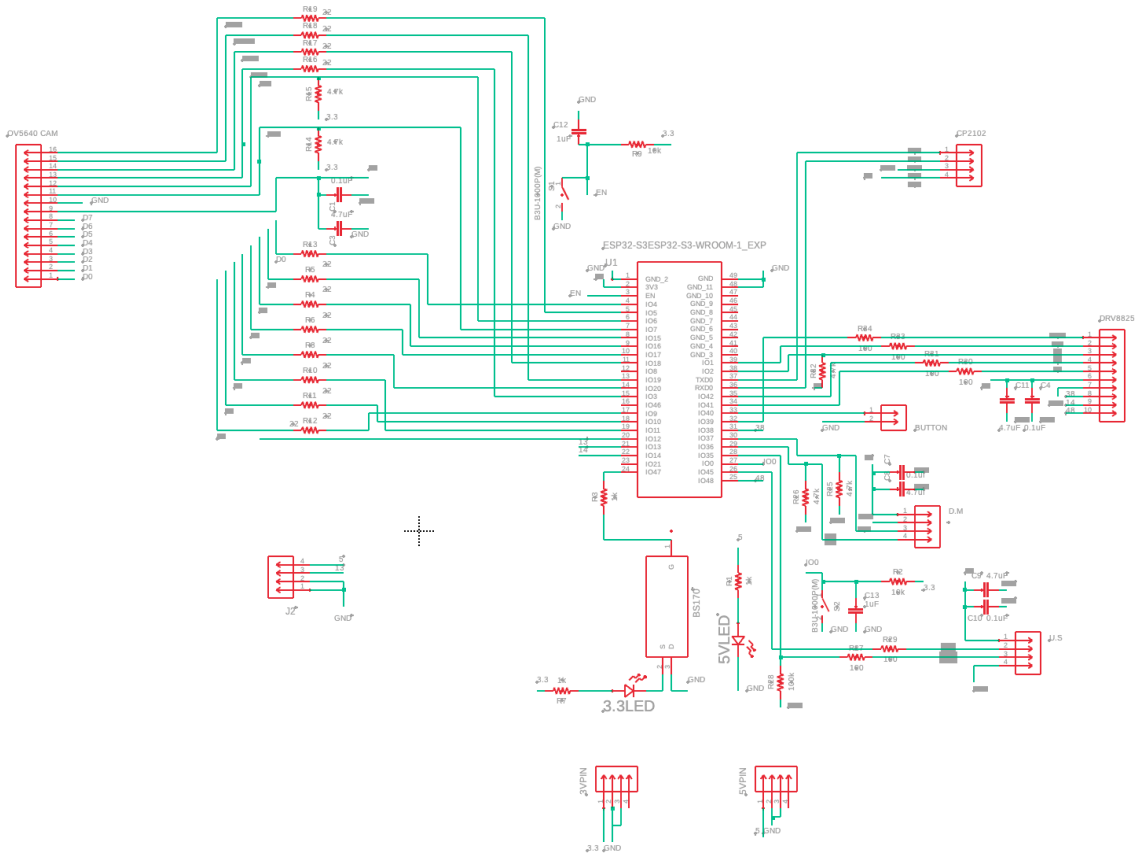

Figure 12: ESP32-S3 PCB Schematic

The MCU is the ESP32-S3 which will be used to run and integrate all the peripherals that we are going to use for the dog feeder. The PCB for the ESP32-S3, shown in Figure 12, details all the connections from the main MCU to other peripherals and other PCB boards. The first connection from the MCU is a 16-pin male header that is designated for the connections of our OV5640 camera. Once connected, the camera will then be placed on the front wall of our pet feeder so that it can run the tasks we have designated for it. The next connection to our MCU is a amplifier where the amplifier will then be connected to a speaker. In our dog feeder this will allow the user to talk through the phone to the dog on the app. We then have a programmer header which will allow for communication with the MCU to allow for programming the embedded software and machine learning algorithms. The fourth connection to the MCU is a 10-pin motor driver header that will then connect to the motor driver and motor schematic. The purpose of the motor in our design is to physically dispense the designated amount of food for the food when the command has been given. Where the motor will start to turn, allowing food to slide into the dog's bowl for feeding. We also have a 6-pin male header for the depth module connection. The depth module will help to ensure that no false positives are being taken when it is time to feed the dog. The last male header that we have connected to the MCU is a 4-pin header for the ultrasonic sensor. The ultrasonic sensor will realize when the food storage container is

getting low where the user will get a notification that the bin is low and needs to be refilled. In the layout, there are two LED's that we are going to be using to test that the input voltage from the regulators is working for both the 3.3v and 5v regulators. We then have two female headers connection to a switch that is designated for the connection of the regulator PCB. This will allow for the connection of the regulators and the input voltage to be able to operate and run our peripherals. Finally, we have a general ground that has locations at pins GND, GND11, and GND2. With the input voltage of 3.3v being connected to the 3v3 pin. This also includes a 10kΩ resistor and a 0.1uF capacitor for noise control and reliability.

6.5.1 Camera

The OV5640 camera that we have picked out contributes to a significant role in the functionality of our project. For our PCB design we used a 16-pin male header that will connect to the ESP32-S3 on our physical board. We will then connect our camera to the header pins, allowing us to position the camera wherever we see fit. For the specific positioning of each node, we placed the 8-bit data bus on all even pins (2-16), which are connected in series with a 22Ω resistor, to ensure the integrity of the system along with the safety of the MCU. The 22Ω resistors connected in series with our 8-bit data bus also help to limit the current that is going into the MCU ensuring that the chip is protected from spikes. Along with this they also help to reduce ringing and overshoots that occur.

We then used pin 1 to connect our input voltage from the 3.3V regulator. We used 2 decoupling capacitors that are connected to our 3.3V input in order to reduce the noise coming into the system. It also helps to keep the current transitions stable throughout the whole system and ensures that the images we are receiving from the camera are stable and don't get corrupted.

For the rest of the odd pins 3-15 we used pin 3 as ground, pin 5 for our SDA node, pin 7 for SCL, pin 9 for XCLK, pin 11 for PCLK, pin 13 for VSYNC, and pin 15 for HREF. For our pins 3 and 5, which are the nodes designated for our SDA and SCL we used a 4.7kΩ pull up resistor that is connected to our 3.3V input in order prevent input problems throughout the system which could result in noise, false triggers, and a change in the voltage level logic. For nodes 7-15 which are designated for XCLK, PCLK, VSYNC, and HREF we also used a 22kΩ resistor that is connected in series. Each pin is connected to the MCU for the same benefit as the data bit pins, which is to reduce noise, have a stable current running into the MCU, and overall increase the stableness and reliability of the system.

When planning the specific location that the camera header will be placed on our PCB board, we ultimately decided to locate the header near one of the edges. The main reason we decided this was to ensure that the camera could be placed in the correct position so that it is able to perform the tasks that we require of it, which are being able to identify the dog on the camera and identify when the dog has sat down for feeding. Having the header for the camera on the front edge will allow us to place the camera on the wall of the dog feeder and will keep the PCB board at a safe distance away so that no risks could present themselves to the board or the dogs safety. For example, if water were to accidentally get in it could fry our electrical components and possibly hurt the dog.

In Figure 13 below, it shows a close-up of the camera interface in the PCB layout. The figure highlights the digital signals with pull-up resistors and local decoupling capacitors. The 22kΩ resistors reduce high-speed signal ringing to ensure reliable communication between the camera and the MCU.

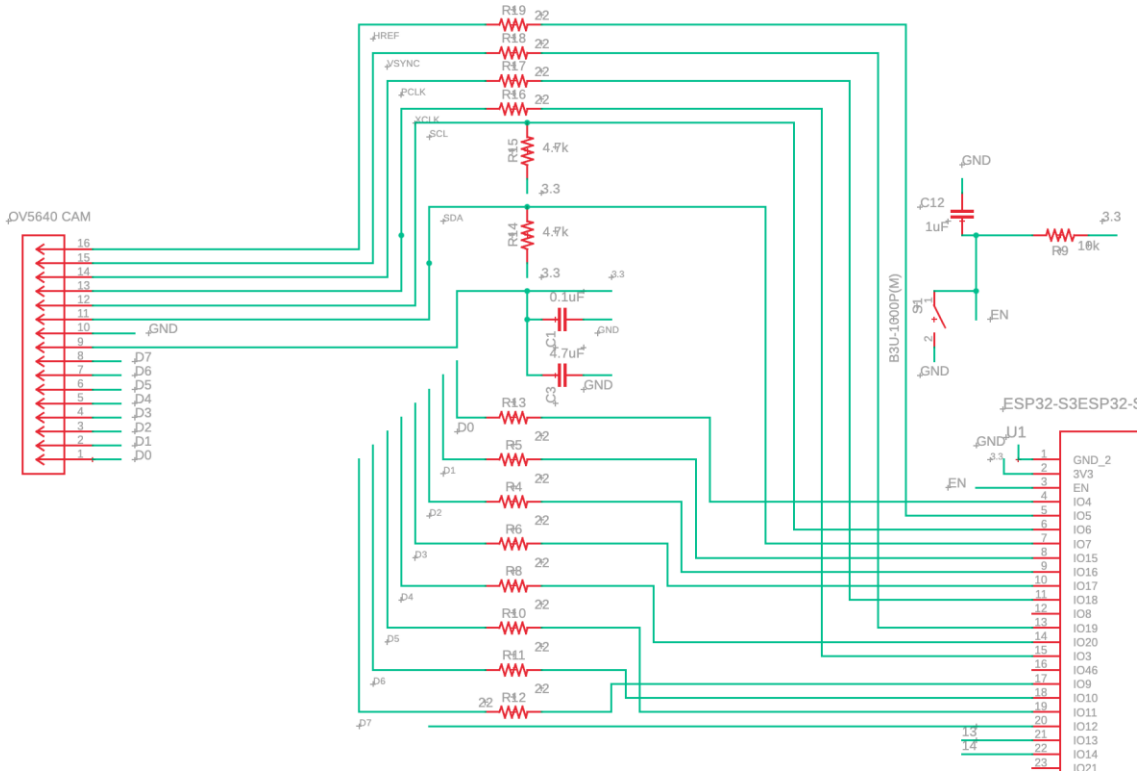

Figure 13: OV5640 Camera Schematic Close-Up

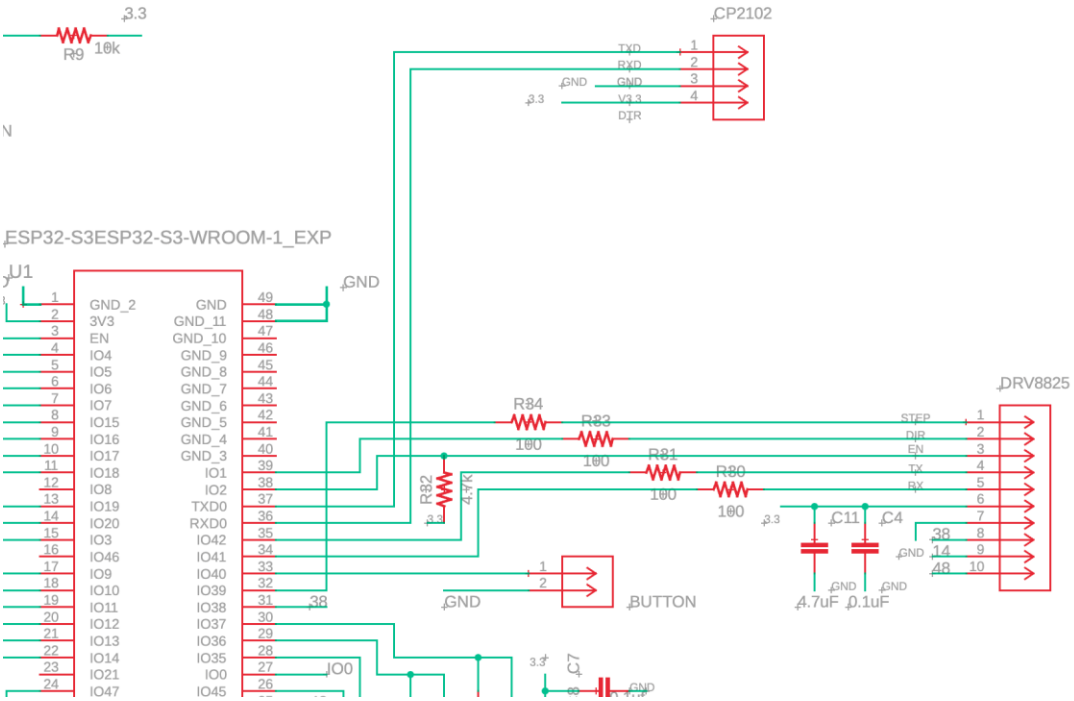

Figure 14: Programmer, Motor Close-Up

6.5.2 Depth Module

The next peripheral that is included in our design is TMF8701 depth module. In our dog feeder, the depth module helps to improve the overall accuracy of realizing when the dog has approached the camera and sits down. Without the depth module, there could be inaccuracies in feeding the dog, and false positives could occur from the dog being too far away from the camera.

The depth module that we used, TMF8701, required four different pin connections for the peripheral to be running. We used pin 1 to input the voltage source to the depth module of 3.3V. Connected in parallel to the input voltage are two decoupling capacitors of 0.1uF and 4.7uF which help to smooth the system and prevent spikes from occurring and harming the MCU or depth module. We used pin two to connect the depth module to our ground. Pins three and four were the pins that we used to connect to the ESP32-S3 to be able to communicate with the MCU, where pin 3 was our SDA pin and pin 4 was our SCL pin. For both SCL and SDA connections, we used a pull up resistor of 4.7kΩ to connect them to the MCU. Without a pull up resistor the buses wouldn't realize if they were high or low and it would result in the depth module not being realized. Finally, we connected the SDA for our depth module to IO37, and we connected SCL to IO36 as seen in Figure 16.

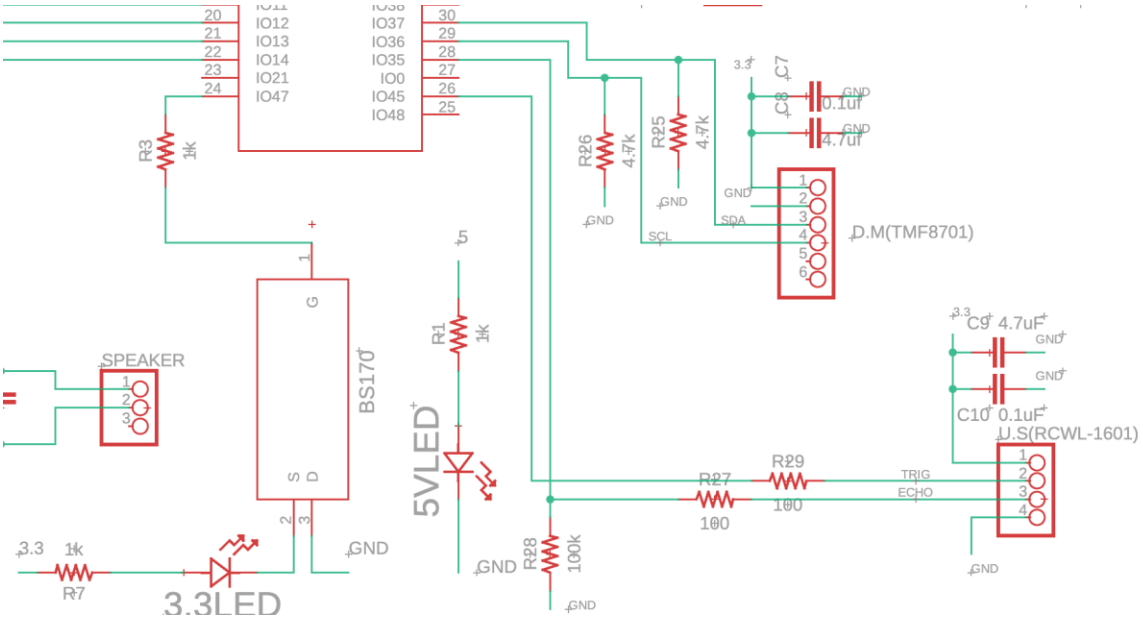

Figure 15:Depth Module, Ultra Sonic Sensor, 3.3 & 5V LEDs

6.5.3 Ultrasonic Sensor

The Sit & Chow dog feeder will have the ability to remotely operate, done by the mobile application, and the user needs to be notified when the food storage is low. An ultrasonic sensor located next to the storage container will be used to detect when the storage container is getting low. Once the sensor has detected that the storage container for the food is low, it will then send a message to the app and notify the user that the storage container is low and needs to be filled.

For the design of the ultrasonic sensor. There are two pins connected to our ESP32-S3, the first being TRIG and the second being ECHO. TRIG is connected in series with a 100Ω resistor to limit the noise and safeguard against any type of spikes in voltage that could occur. Where ECHO is also connected in series with a 100Ω resistor with the addition of a pull-down resistor of 100kΩ to ensure that the ultrasonic sensor doesn't have any type of false triggers, that would tell the system that the feeder was empty. The functions of each of these pins consist of TRIG creating a pulse when the sensor has been told it to, and ECHO will tell the length of the pulse that was taken.

6.5.4 Power Input

The input of power throughout the PCB contributes to a significant role in the overall reliability of the PCB and the dog feeder. Without a clean and stable power supply, we would be unable to functionally use any of the peripherals that we need to successfully develop the dog feeder. For our dog feeder, we will be using a 12V jack as the main power supply function. We will then have a backup power supply with the intent of eliminating the risks that arise from losing power. The backup power supply that we will be using is a backup battery. The main power supply will then be regulated by two regulators with voltage levels of 3.3 volts and 5 volts. The schematic shown in figure 10 will then connect to the main PCB using two headers from both the regulator PCB and the main PCB. This

will then allow us to regulate the voltage level throughout the main PCB in order to supply power for the different peripherals that we are going to use.

In Figure 16, displays the 3.3 and 5V power selection and output switch circuit. It illustrates the selectable power-rail where 3.3V and 5V are routed to the same power line. A single-pole switch controls if the power line passes through to the output.

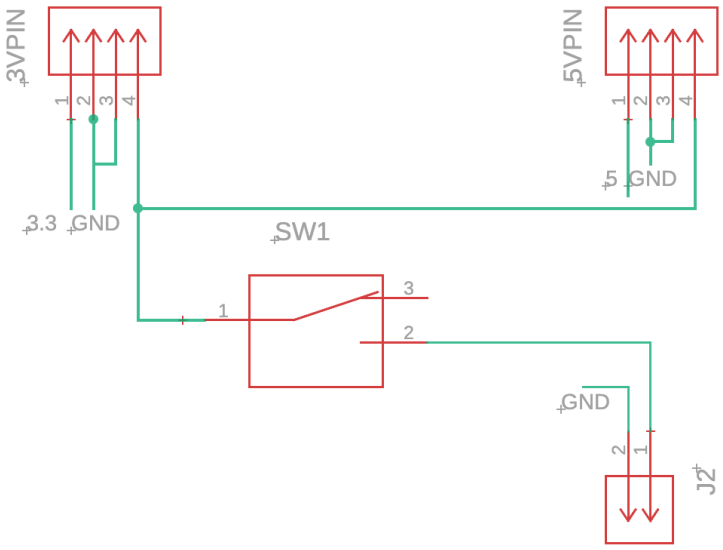

Figure 16: Regulator Headers and Power Input Schematic

Chapter 7 – Software Design

The software design of Sit & Chow is organized into three integrated parts: the embedded software, the mobile application, and the detection model. The embedded software on the microcontroller manages all hardware operations. This includes the stepper motor and motor driver, reading sensors, managing power states, and handling Wi-Fi communications to the backend server. The mobile application serves as the user interface for the embedded system. Through the app, users can configure feeding schedules and portion sizes. The app communicates with a Firebase Realtime Database (RTDB) that stores user settings, logs, and alerts, and with a Cloud Run backend server that handles authentication token management and push notifications. The detection and classification pipeline resides in the backend, where camera frames are streamed from the ESP32 to determine whether the dog is sitting. These three areas form the end-to-end architecture for the Sit & Chow smart feeding system.

7.1 MCU

The MCU is responsible for handling all peripherals of the device. This includes communicating with the camera, the speaker and amplifier, the stepper motor and motor driver, the manual override button, the ultrasonic sensor, and the time-of-flight (ToF) presence sensor. It also handles all communication to and from the mobile application via Firebase Realtime Database (RTDB).

On power-up, the ESP32-S3 follows a fixed initialization sequence. First, it connects to Wi-Fi using BLE provisioning — the user pairs the device through the mobile app over Bluetooth to supply network credentials, which are then stored in non-volatile storage (NVS) for future boots. Once Wi-Fi is established, the device syncs its clock via SNTP and initializes the Firebase client. Authentication with Firebase is handled through short-lived ID tokens obtained from a Cloud Run /esp-token endpoint. These tokens are automatically refreshed every 55 minutes before they expire. The device then fetches its owner UID from the RTDB, reports its online status, and starts all background tasks. All communication between the ESP32, the mobile app, and the backend uses Firebase RTDB as a central hub. The ESP32 communicates with RTDB over HTTPS using its REST API. To avoid repeated TLS handshake overhead, the firmware maintains three persistent HTTP client handles: one for general short Firebase REST calls (GET, PATCH, PUT, POST, DELETE), one dedicated to audio intercom polling, and one for camera-related polling. This strategy prevents heap fragmentation and eliminates the latency of re-establishing TLS sessions on every request. The ESP32 polls the devices/{deviceId}/commands/pending node in RTDB every 5 seconds for feed-now commands sent from the app. A separate heartbeat task writes the device's online status to RTDB every 30 seconds so the app can display connectivity state. The schedule runner is a FreeRTOS background task that wakes every minute, reads the user's schedules from userSchedules/{ownerUid} in RTDB, and checks whether any schedule matches the current local time. When a match is found, the feed callback is triggered. The primary feed workflow proceeds as follows:

1. Beep — the speaker emits an audible tone to alert the dog that feeding time has started.
2. Pet approach detection — the ToF sensor (TMF8801) waits for the dog to come within a configured distance threshold. If no presence is detected within the timeout window, the feed is aborted to avoid wasting food.
3. Sitting detection — once the dog is detected, the camera activates and begins streaming JPEG frames over a WebSocket connection to the Cloud Run /ingest endpoint. The backend runs the machine learning algorithm on incoming frames and writes a posture result back to RTDB. The ESP32 polls this result node; if a sitting position is confirmed within the timeout, the feed proceeds immediately and is logged as a success. If no sit is detected within the timeout, the food is dispensed anyway so the dog is not left hungry, and the event is logged as a failure.
4. Dispense — the ESP32 sends the appropriate number of steps to the DRV8825 stepper motor driver, translating the configured portion size in grams into the corresponding number of motor turns.
5. Capacity check — after dispensing, the HC-SR04 ultrasonic sensor measures the food level in the hopper. If food is detected as low, a notification payload is written to RTDB, which the Cloud Run backend picks up and forwards to the user as an FCM push notification.
6. Log event — the feed result (timestamp, portion size, sitting detection outcome, source) is written to devices/{deviceId}/logs in RTDB for display in the app's feed history.

The diagram below (Figure 17) shows the workflow of the entire scheduled feed process.

Figure 17: Scheduled Feed Workflow

After each scheduled feed, the ESP32 performs a capacity check using the ultrasonic sensor. If the hopper is not low, the sensor is powered down and normal operation resumes. If the food level is below the threshold, the ESP32 writes a notification event to RTDB. The Cloud Run backend monitors this RTDB node and, upon detecting a new notification, uses the Firebase Admin SDK to send an FCM push notification to all registered device tokens for the owner. The workflow is shown below in Figure 19.

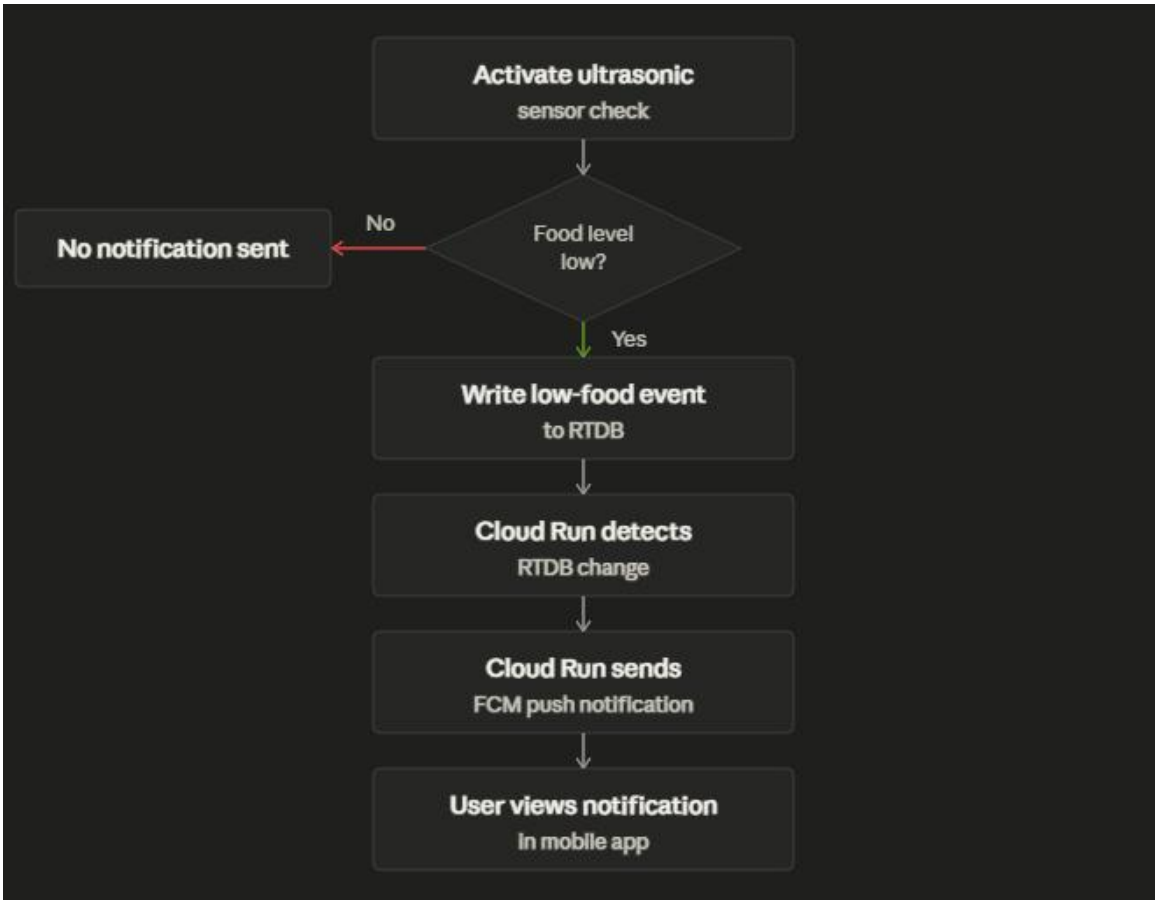

Figure 18: Notification Workflow

The ESP32 continuously captures camera frames and converts them to JPEG. When streaming is enabled, frames are sent as binary WebSocket messages to the Cloud Run /ingest endpoint. The Cloud Run server buffers the latest frame in memory and serves it to any connected Flutter client that requests the live feed. On first connect, the ESP32 sends a JSON hello message containing its device ID so the server can associate the stream with the correct device. The mobile app displays the stream as a continuously refreshing MJPEG feed. This WebSocket relay architecture avoids the need for the device to have a publicly

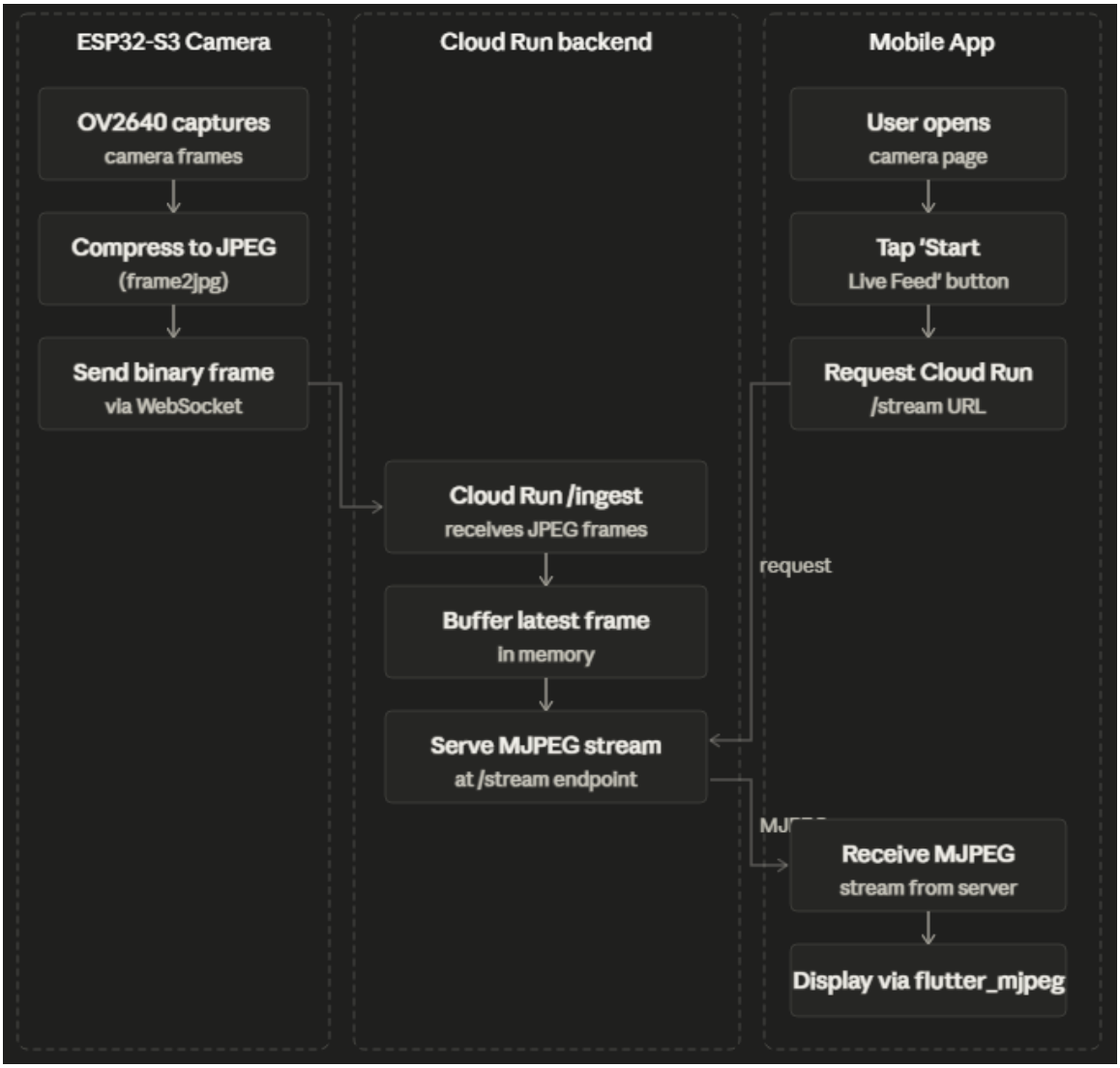

Figure 19: Live Stream Workflow

There is also microphone functionality that allows the user to speak to the dog through the feeder's speaker. When the user activates the microphone on the camera live page, the Flutter app begins recording audio in rolling chunks. Before transmission, the WAV header is stripped from each chunk, leaving raw 16-bit signed PCM at 16 kHz mono. This raw PCM data is base64-encoded and written directly to the devices/{deviceId}/audio node in RTDB, along with a chunkIndex that increments with every new chunk and a Unix timestamp.

The ESP32 runs a background audio intercom task that polls this RTDB node approximately once per second. When it detects a new chunkIndex value, it reads the payload, base64-decodes the PCM data using mbedTLS, and passes the resulting sample buffer to the speaker playback driver. The PCM is played through the LM386 amplifier and speaker in real time. When the user toggles the microphone off, the Flutter app deletes the audio node from RTDB, and the intercom task detects the missing node and stops playback. Because audio payloads can reach 100+ KB per chunk, the intercom client uses a PSRAM-backed buffer to avoid stack overflow and silent data truncation. The workflow is shown below in Figure 20.

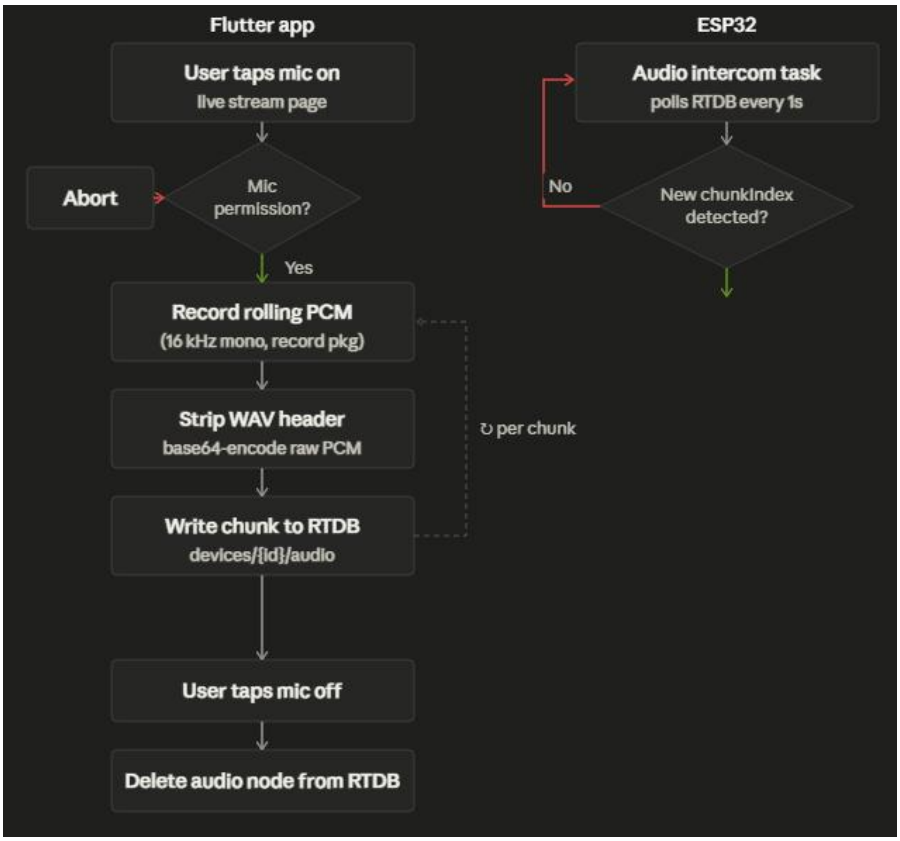

Figure 20: Microphone Workflow

7.2 Mobile Application

The mobile application is tasked with hosting the user interface so that the user can properly operate the dog feeder. It must meet several requirements. First, it must be user-friendly, with clear labels and intuitive navigation. Second, it must handle all necessary data input and output, including schedules, devices, feeding logs, and user account information. Third, it must communicate bidirectionally with both the ESP32 (through RTDB) and the Cloud Run backend.

The mobile application is built with Flutter and uses Firebase as its backend. A Firebase Realtime Database stores all persistent data, kept intentionally shallow to minimize Firebase read costs and latency. The database stores users, schedules, devices, and logs. The app also registers for FCM push notifications through the Cloud Run backend to receive alerts for feeding events and low food warnings.

The first page is the landing page where users must log in or register, as seen in Figure 21. Authentication is handled by Firebase Authentication. When a user registers, they provide their full name, email address, and password. This creates a Firebase Auth account that is used to authenticate all subsequent RTDB reads and writes, ensuring that a user can only access their own devices and schedules.

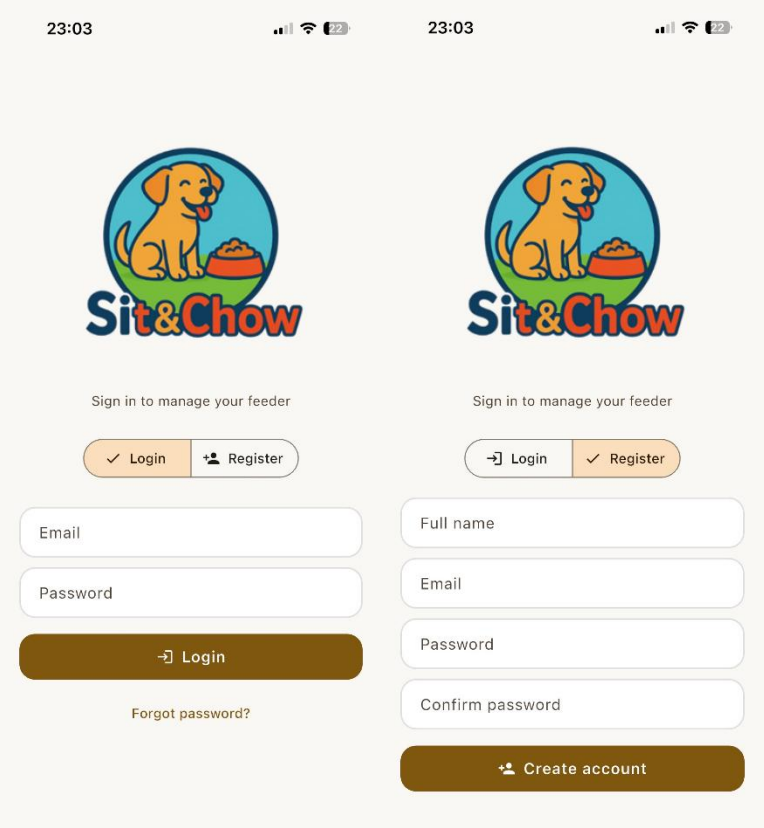

Figure 21: Login and Register Page

After logging in, the user lands on the schedule page. The app has a bottom navigation bar containing four tabs: Schedule, Devices, Camera, and Settings. The top app bar displays the current tab name and, on the schedule page only, a history icon that opens the feed log.

The schedule page begins with a Quick Feed option, allowing the user to select a paired device and immediately dispense a specified amount of food. This triggers a feed-now command written to devices/{deviceId}/commands/pending in RTDB, which the ESP32 picks up on its next 5-second poll cycle. The quick feed bypasses the sitting detection algorithm entirely.

Below the quick feed option is the schedules collection, displaying all configured feeding times. Each schedule entry shows the time, days of the week, portion size, and assigned device. Schedules can be edited to change the time, portion size, active days, or assigned device, and can be deleted. An Add Schedule button at the bottom lets the user create new entries by setting the feed time, portion size, day pattern, and target device. All schedule data is written to userSchedules/{ownerUid} in RTDB, where the ESP32's schedule runner reads it.

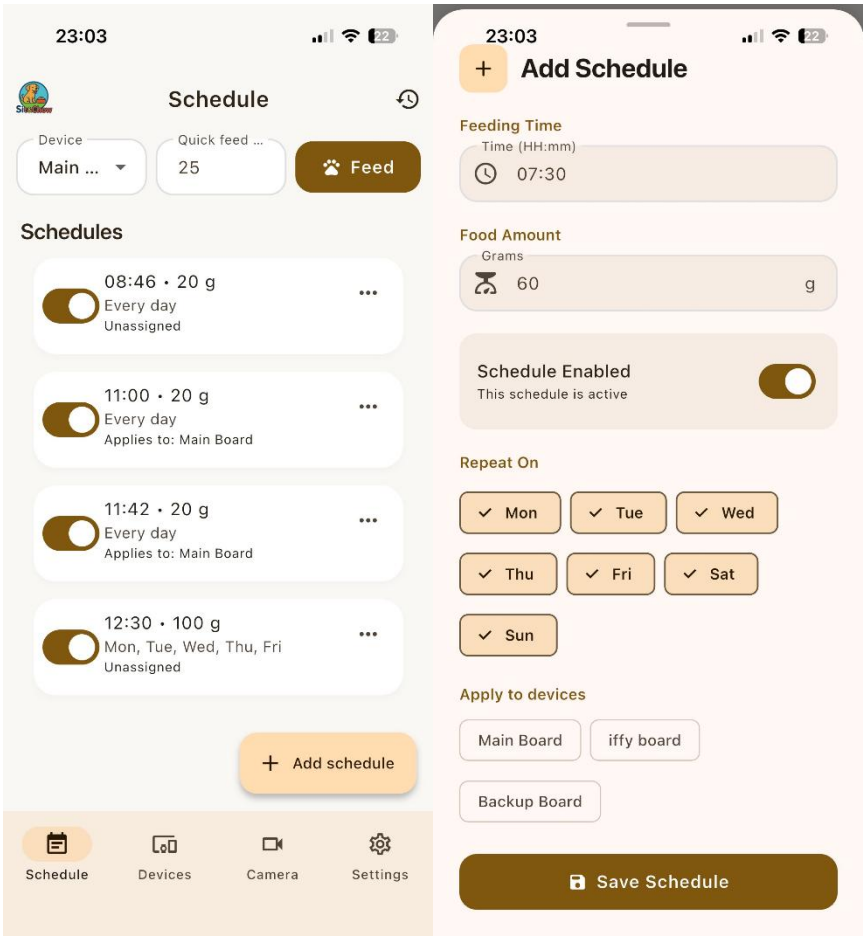

Figure 22: Schedule Page and Add Schedule Widget

The history icon on the schedule page opens the feed log, as seen in Figure 24. This log displays all past feeding events, each showing the timestamp, portion size, trigger source (scheduled or manual), and the outcome of the sitting detection attempt. Successful detections are marked as successes; feeds that timed out without detecting a sit are marked as failures. This gives the user visibility into the dog's training progress over time.

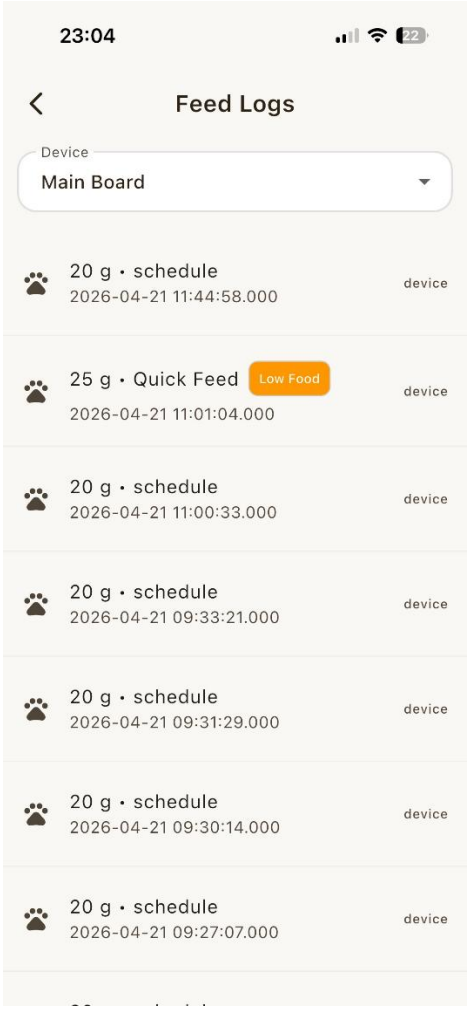

Figure 23: Feed Log Page

The device page, shown in Figure 25, is where the user manages paired feeders. Existing devices are listed with their assigned name and online status, which is read from the devices/{deviceId}/status node in RTDB updated by the ESP32 heartbeat. Users can edit a device's display name or delete a device entirely.

To add a new device, the user taps the Link Device button. The app initiates BLE provisioning, scanning for nearby ESP32 devices broadcasting the provisioning service. Once found, the app transmits the home Wi-Fi credentials to the ESP32 over BLE, and the device connects to the network and registers itself in RTDB. The app then associates the new device with the user's account.

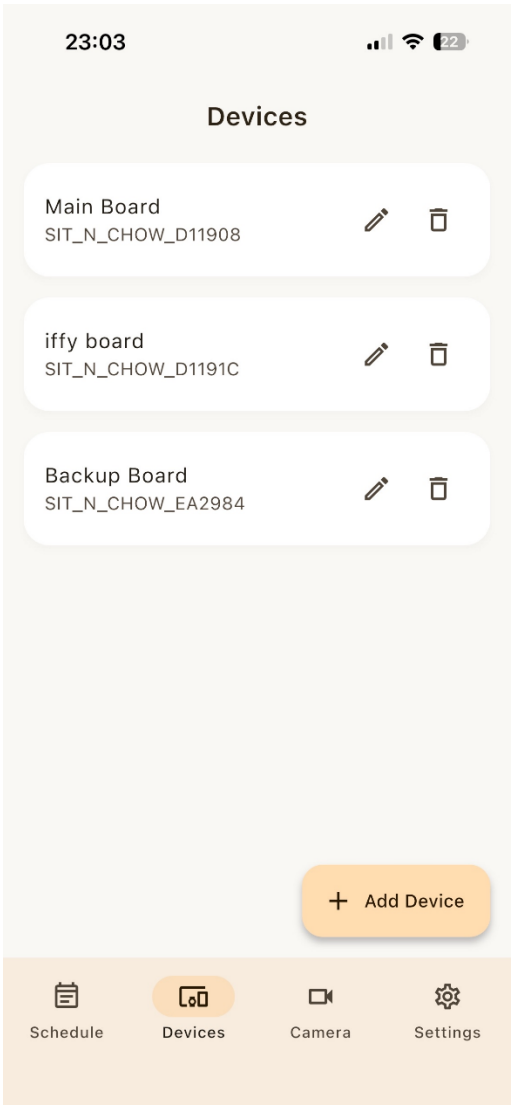

Figure 24: Device Page

The camera page is shown in Figure 26. When the user navigates to this tab, a button prompts them to start the live feed. This deliberate click prevents the feed from loading automatically every time the tab is opened. Once the feed is started, the app connects to the Cloud Run streaming endpoint and displays the MJPEG stream using the flutter_mjpeg package. The stream is sourced from the Cloud Run server, which relays frames received from the ESP32's WebSocket camera ingest connection.

Overlaid on the live feed is a pulsing microphone button. When tapped, the app starts recording audio using the record package. Each audio chunk is processed by stripping its WAV header to extract raw PCM, base64-encoding it, and writing it to the devices/{deviceId}/audio node in RTDB. This process repeats in a rolling loop while the mic is active. Tapping the button again stops recording and deletes the audio node from RTDB, signaling the ESP32 to stop playback.

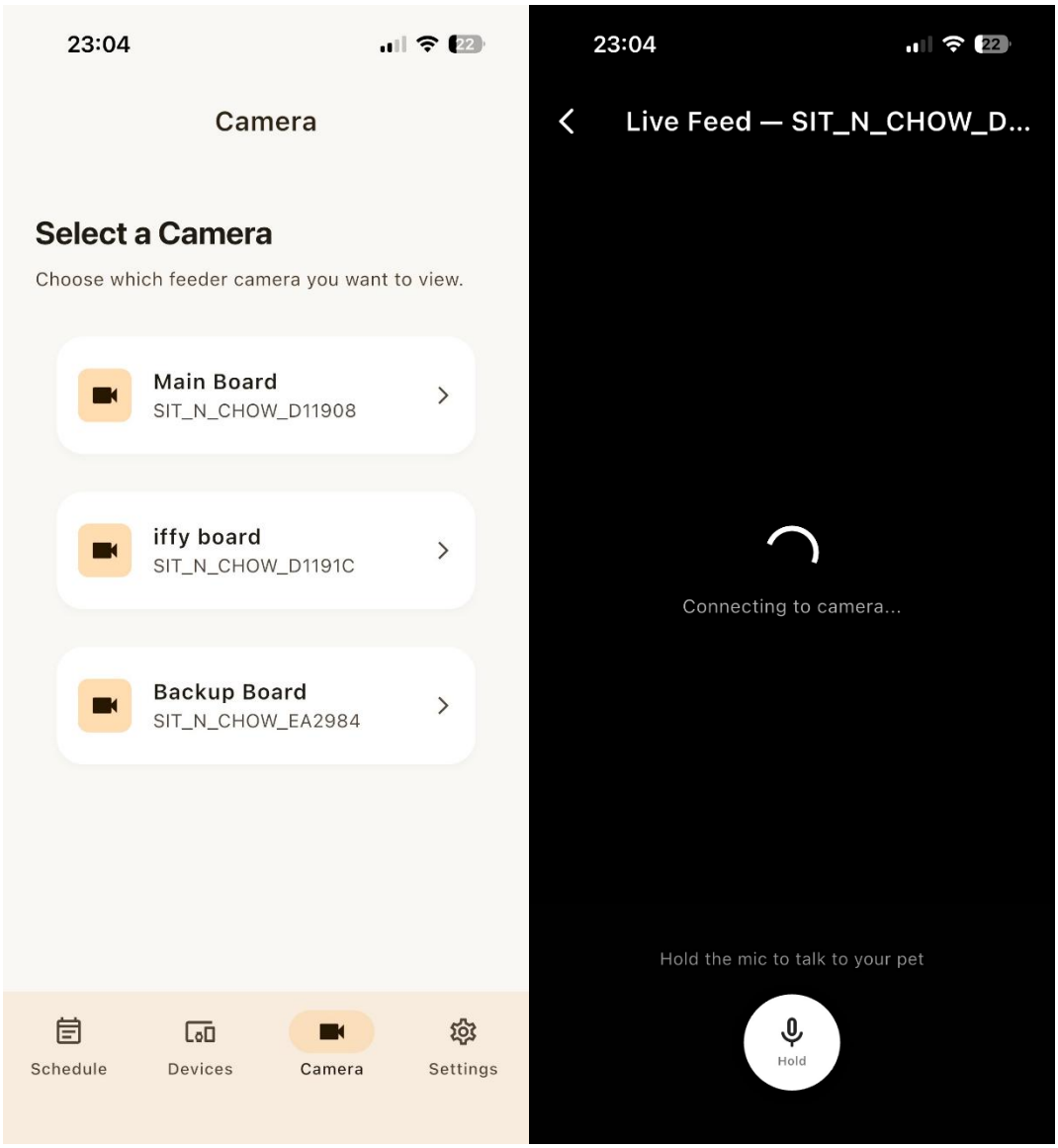

Figure 25: Camera and Live Camera Page

The settings page is shown in Figure 26. Here the user can view their name and email address and edit their display name. Options are available to verify the email address, reset the password, and toggle push notifications. Notifications are delivered via FCM and alert the user when a scheduled feed occurs or when the food hopper runs low. The page also provides a logout button that clears the local auth session and returns the user to the landing page.

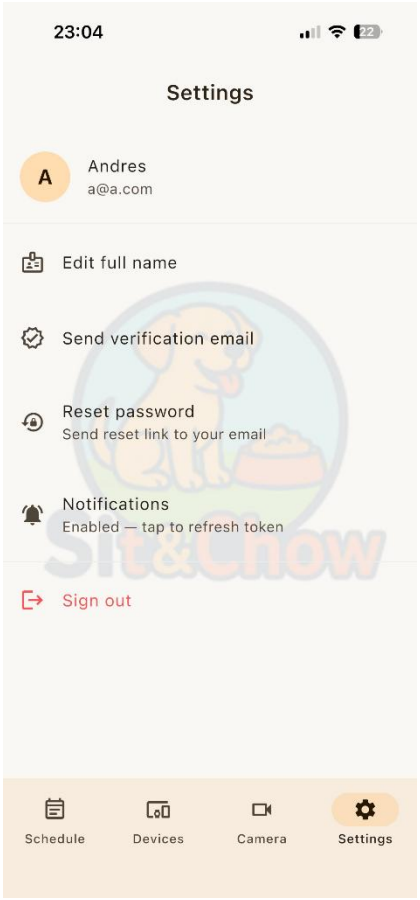

Figure 26: Settings Page

7.3 Machine Learning Algorithms

The detection and classification models are 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. The system utilizes computer vision to "see" that a dog is in view, and that the dog is "sitting" before dispensing food. The pipeline integrates the camera for inference, Time-of-Flight (ToF) sensor to make sure the dog is in range, and embedded logic for the dispensing mechanism.

The machine learning subsystem integrates:

- ToF sensor to verify dog is physically within range for inference.
- Finetune YOLOv11 detector to identify and localize the dog in the frame.
- A binary classifier to determine if the dog is "sitting" or "not sitting."
- Ensure that the "sit" position is held for at least 2 seconds.
- Lockout logic to prevent dispensing back-to-back.
- Event outcome publishes to the backend to publish in the mobile application.

7.3.1 Depth Sensor: Range Estimation

The pipeline begins with the ToF sensor to continuously estimate the distance between the Sit & Chow dog feeder. This feature helps prevent false activations by ensuring that the dog is within about 1 foot of the dog feeder before the detection and classification inference. This condition is reflected in the first diamond in Figure 28. This first step in criteria is required, or the system does not proceed to the dispensing logic.

7.3.2 YOLOv11 and Posture Classification

The detection and classification model operates in five main stages:

- Getting the image
- Preprocessing the image
- Displaying and cropping the bounding box image
- Classifying the image
- Event publishing.

Detection Model

YOLOv11 is a pretrained model that will be finetuned with a new set of dog images to detect only dogs. Regression loss is optimized with stochastic gradient descent or Adam with a learning rate of 0.1, 0.01, or 0.001. Performance will be monitored using metrics such as the mean average precision (mAP50 and mAP50-95). The metric for mAP50 indicate that the predicted bounding box overlaps at least 50% of the actual bounding box within pre-made images. For mAP50-95, the output is the average of different thresholds (0.50, 0.55, 0.60, …, 0.95). These metrics show how accurately the model found the dog and how consistent the model found it.

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 image inside the box. Each frame is passed into a YOLOv11 model that is a lightweight object detector that is finetuned to specifically detect dogs. The detector outputs:

- A bounding box around the dog.
- Confidence scores
- Class predictions

In the event that no bounding box is identified, the system will:

- Dispense food after a 3 minute timer finishes.
- Logs the outcome to display on the mobile application.

Posture Classification: "Sitting" vs. "Not Sitting"

The binary classification algorithm will be trained with a large dataset of various image sizes of dogs in different poses that will be deployed on Firebase. The model is trained with binary cross-entropy loss, where the target labels are “sitting” and “not sitting.” After the bounding box is identified, the frame is cropped to classify the dog’s posture. The new cropped image is then passed into a convolution neural network that is a binary classifier to distinguish between “sitting” and “not sitting.”

Before the food dispenses, the classification prediction must be stable:

- The dog must remain “sitting” or “not sitting” for at least 5 seconds.
- Confidence score must exceed a threshold of 0.80.

If the output fluctuates, the systems waits until the dog is detected in one of the positions, or it is no longer the scheduled feeding time.

7.3.3 Integration

Once the dog is in range, the camera activates and runs the machine learning pipeline. It does this through two services on Google Cloud Run. One service, the WebSocket, holds the machine learning algorithms and models, and the other streams the camera frames that calls the machine learning service on each frame. The WebSocket has two timers: a three-minute timer for the entire scheduled feeding event, and a five-second timer for the sitting occurrence. There is a timer when the dog is seen sitting so that the dog can properly show and learn the behavior needed for the feeder to dispense food. If the dog fails to sit within the three-minute feeding window, then the dispense logic will still proceed. Whether or not the dog is seen sitting or not sitting, the outcome is logged and seen in the mobile app. This is done by sending the final result to the Firebase database.

Here is the entire process:

- Dog is in range, and the camera activates.
- WebSocket streams the camera frames and calls the machine learning prediction on each frame.
- Dog is seen sitting for 5 seconds, and the feeder dispenses food.
- Dog fails to sit within 3 minutes of the scheduled feeding time, and the feeder dispenses food.
- Either outcome is logged into the mobile app’s logs.

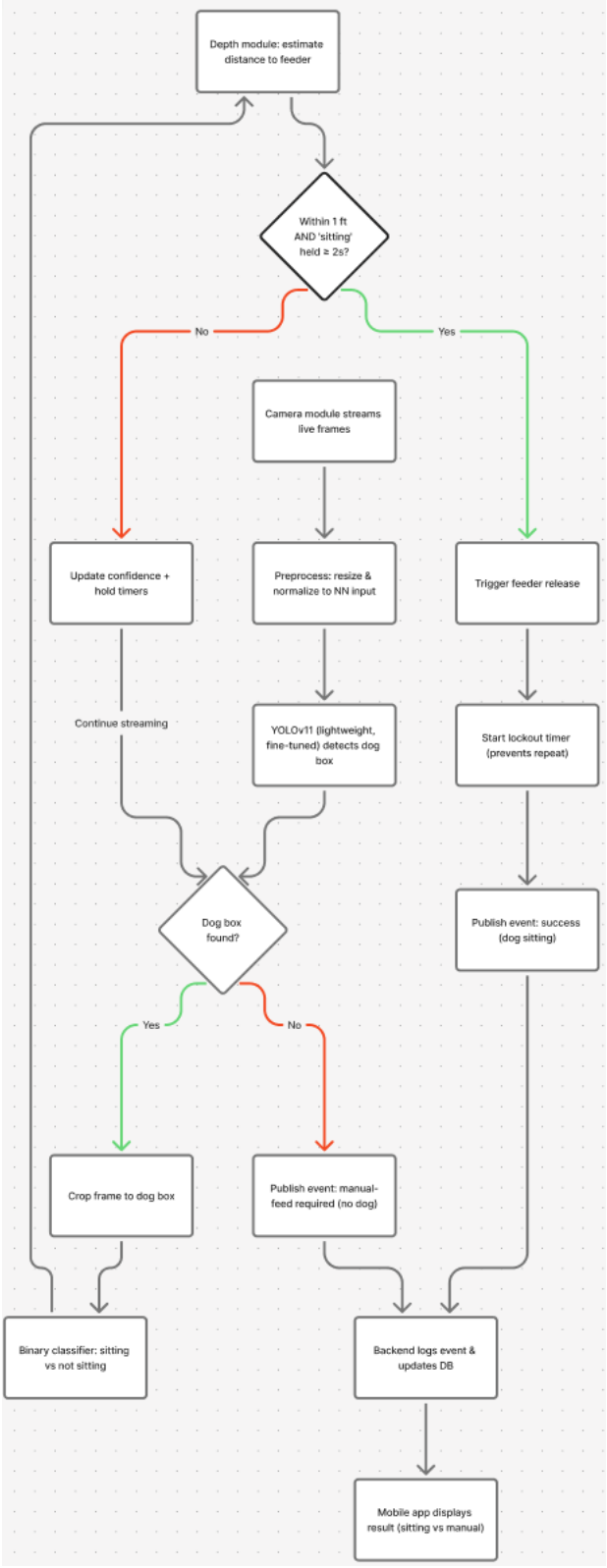

Figure 27: Detection and Classification Flowchart

Chapter 8 – System Fabrication/Prototype Construction

In this dog feeder project, we learn that designing a system is more than just choosing components, it's about making everything truly fit together. That's where PCB design and 3D modeling in Fusion 360 become incredibly important. When we work on the PCB, we focus on the electrical world: footprints, traces, ground pours, mounting holes, test-points, current ratings, and part orientation. Every line on the board matters, and every component placement affects performance. But when we switch to the 3D model in Fusion 360, and KiCad, we can visualize how the PCB sits inside the enclosure, whether the connectors are aligned properly, if the wiring has space to bend, and even how the lid or screws might interfere with components.

Fusion 360 & KiCad lets us link the PCB layout directly to a realistic 3D model. Then, the circuit isn't just a schematic, it becomes a real physical object that we can rotate, measure, and check for clearances. We can see if there are some problems with our design.

8.1 Power Supply Unit

First, we utilize power supplied by a wall adapter. This power is then connected to the PCB via a power jack and split into three 12V branches. The 12V supply is directed to the 3.3V voltage regulator, the 5V voltage regulator, and the motor. Additionally, to facilitate easy verification of whether the circuit board is operational, we have incorporated an LED indicator.

The PCB layout is a two-layer design with the top layer used mainly for power routing and the bottom layer acting as a solid ground plane. The board size is about 49.5 mm by 45.4 mm, and components are arranged to keep the layout compact and efficient. The DC power jack is placed at the edge for easy access, with the input voltage routed inward using a wide trace to handle higher current safely, while the ground is directly connected to the ground plane.

Power routing uses thick copper traces and large copper areas to reduce resistance and improve heat dissipation. High-current paths are kept short and avoid sharp angles to minimize losses and noise. The voltage regulator and its supporting components are placed close together to reduce the switching loop area, which improves efficiency and stability.

The bottom layer is a continuous ground plane that provides a low impedance return path, reduces noise, and improves thermal performance. Connectors are placed away from the power section, with thinner traces used for signals to prevent interference. Overall, the layout follows good design practices by separating power and signal paths, using wide traces for current flow, and maintaining a solid ground reference.

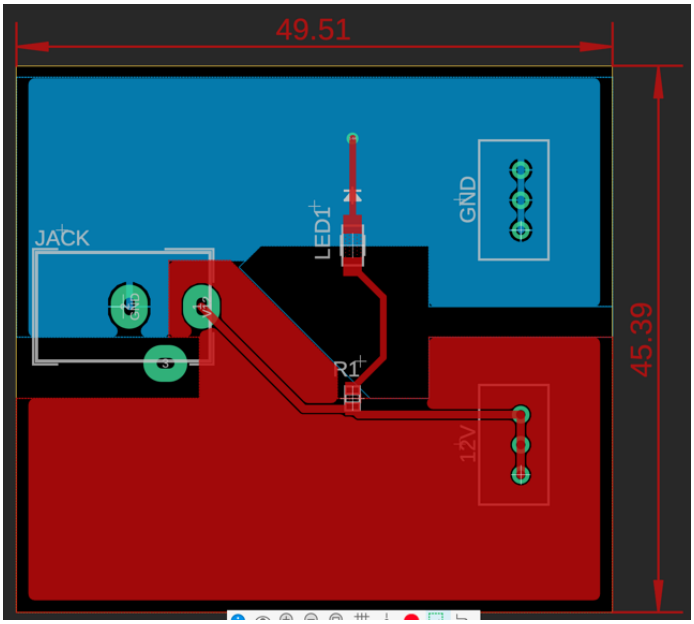

Figure 28: Power Jack board PCB Layout

And this is power jack board for 12V in 3D model.

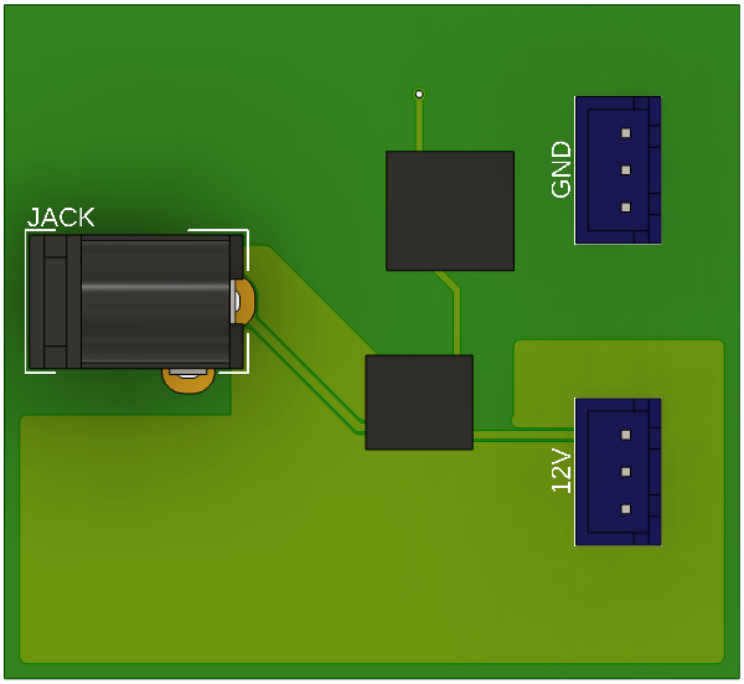

Figure 29: Power Jack board 3D Model

Based on the LM2576 datasheet, the PCB layout should keep the switching regulator section compact because the input capacitor, catch diode, inductor, and output capacitor strongly affect performance. The LM2576 is a 3 A step-down switching regulator, so wide copper traces and large copper areas should be used for the input and output power paths to reduce voltage drop and heat.

In the layout shown, the red layer is used for the main power routing, while the blue copper area acts as the ground plane. The input capacitor should be placed close to the LM2576 input pin and ground pin. This reduces noise caused by fast switching current. The catch diode should also be placed close to the regulator and connected with short, wide traces because it carries switching current when the internal switch turns off.

The inductor should be placed near the LM2576 output pin, but the copper area connected to the switching node should not be made larger than necessary. A large switching-node copper area can increase electrical noise and EMI. The output capacitor should be placed close to the inductor and output connector to smooth the regulated voltage before it leaves the board.

The ground connections of the input capacitor, catch diode, output capacitor, and LM2576 should return to a solid ground plane with short paths. This reduces ground loops and improves stability. In this design, vias should be used to connect top-layer ground points to the bottom ground plane. Overall, the layout should use short, wide traces for high-current paths, keep the regulator components close together, separate noisy switching traces from signal traces, and maintain a strong ground plane for reliable operation.

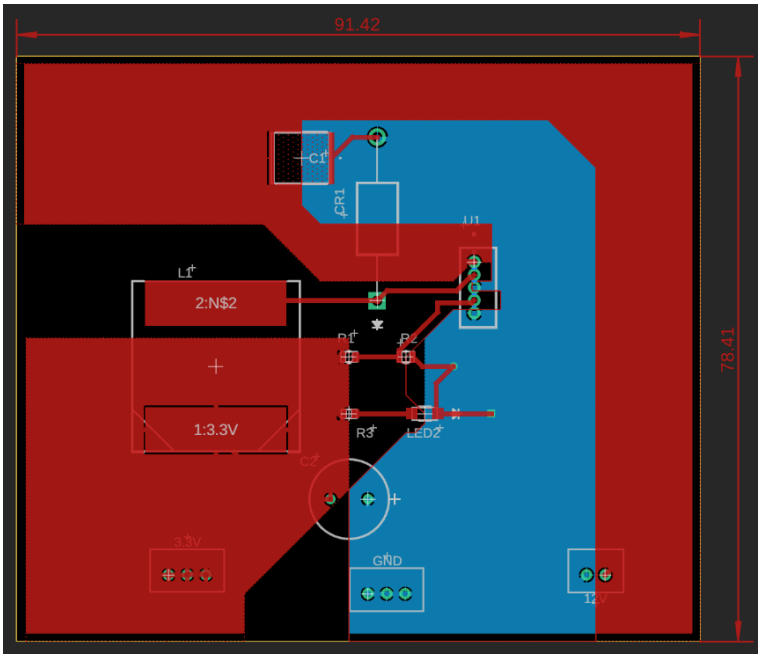

Figure 30: 3.3V Regulator PCB Layout

Similar to 3.3V, 5V regulator uses LM2576-ADJ, we can use the same layout as 3.3V.

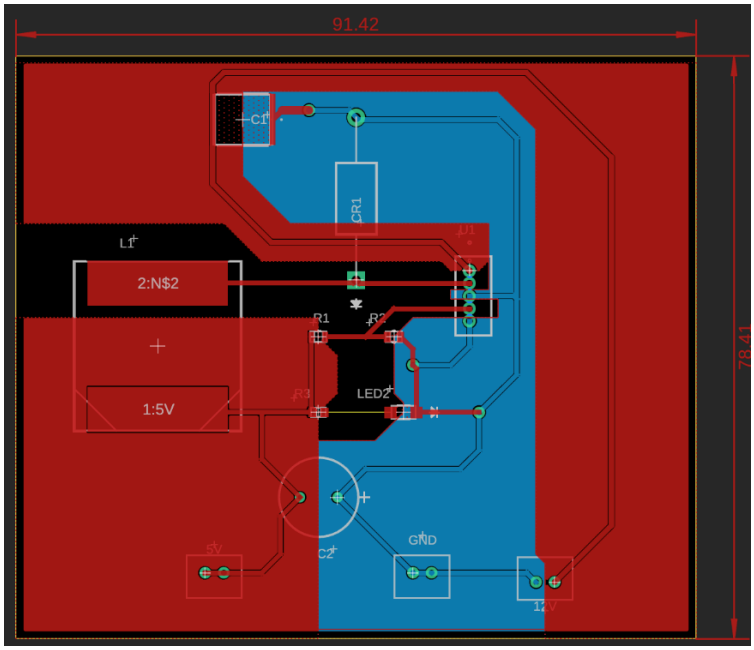

Figure 31: 5V Regulator PCB Layout

Since we use the same model for both 3.3V and 5V, the layouts for both of them are the same.

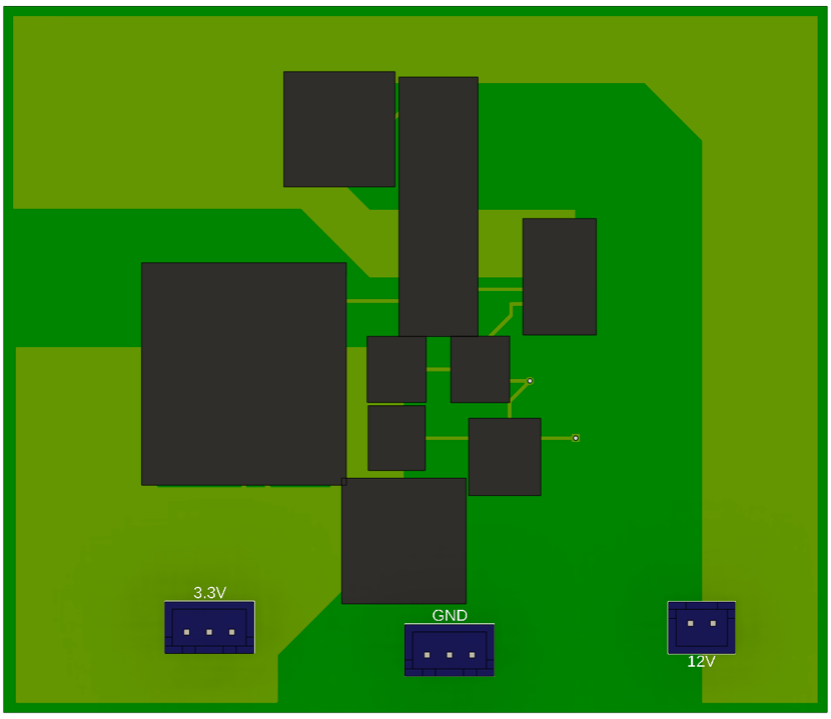

Figure 32: Voltage Regulators 3D Model

8.2 CP2102 Programmer

Based on the CP2102 datasheet and the PCB layout shown, the design focuses on proper USB signal routing, stable power delivery, and compact component placement. The board is relatively small (about 37 mm by 20 mm), so components are placed tightly around the CP2102 to minimize trace lengths and improve signal integrity. The CP2102 IC is positioned near the center, with surrounding passive components such as decoupling capacitors (C1, C2) placed as close as possible to its power pins, which is critical for stable operation according to the datasheet.

The USB connector (USB-A1) is placed at the edge of the board for easy external connection. The differential data lines (D+ and D−) are routed directly from the USB connector to the CP2102 with short, parallel traces. These traces should be kept equal in length and routed together to maintain proper impedance and reduce noise, as recommended in the datasheet. Sharp bends and vias on these lines should be minimized to preserve signal quality.

Power from the USB (5V) is routed to the CP2102 and supporting circuitry using short and reasonably wide traces. Decoupling capacitors are placed very close to the VDD pins to filter noise and stabilize the voltage. A ground plane (visible in red/filled copper) is used to provide a low-impedance return path, which is essential for both power stability and USB signal integrity. All ground pins and components connect to this plane using short paths and vias where necessary.

The UART interface pins from the CP2102 are routed to header pins (labeled near ESP32), allowing communication with an external microcontroller. These signal traces are thinner and routed away from the USB differential pair to avoid interference. The layout keeps digital signals organized and avoids crossing over noisy power areas.

Overall, the PCB layout follows the CP2102 datasheet guidelines by keeping USB traces short and matched, placing decoupling capacitors close to the IC, using a solid ground plane, and organizing components compactly. This ensures reliable USB-to-serial communication and stable performance when interfacing with devices like an ESP32.

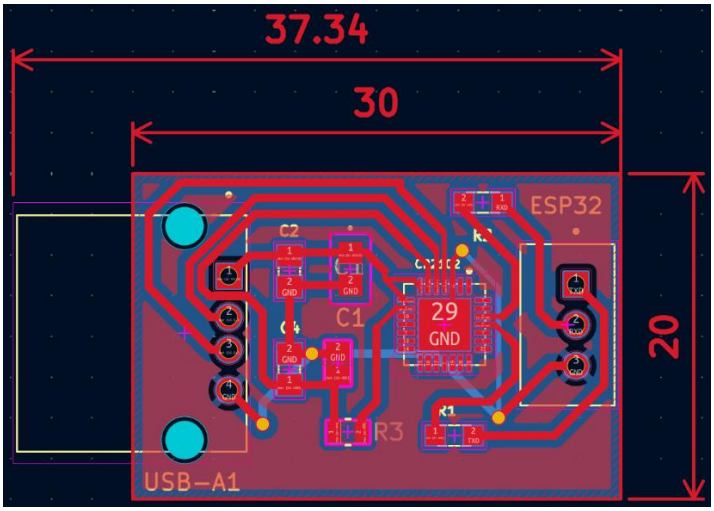

Figure 33: CP2102 Programmer PCB Layout

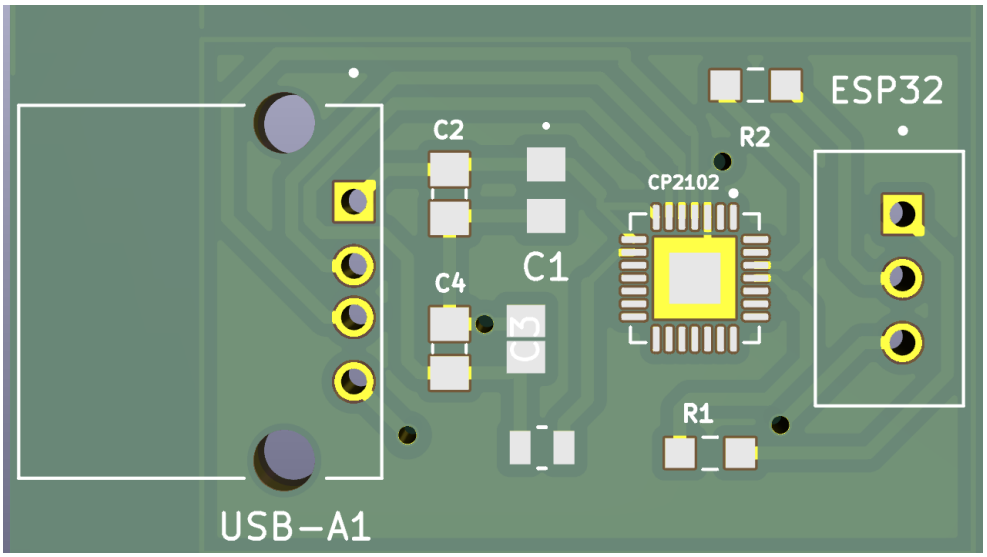

Figure 34: CP2102 Programmer 3D Model

8.3 LM386 Amplifier

Based on the LM386 datasheet and the PCB image, the layout should keep the audio amplifier section compact and organized. The LM386 is placed near the center so the input, output, power, and ground traces can be routed with short paths. The board size is about 30 mm by 30 mm, which is suitable for a small audio amplifier module.

The input connector should be placed close to the LM386 input pins to reduce noise pickup. The input signal traces should be kept short and separated from the speaker output traces because the output carries a larger amplified signal. The gain components connected between pins 1 and 8 should also be placed close to the IC, since the datasheet explains that these pins control the gain from 20 up to 200 when a capacitor is added.

The power pin of the LM386 should be connected with a short trace, and a bypass or decoupling capacitor should be placed close to the supply and ground pins. This helps stabilize the voltage and reduce noise. The ground should be made as a large copper area, as shown in the layout, to provide a low-resistance return path for the circuit.

The speaker output trace from the LM386 should be wider than the input signal traces because it carries more current. The large capacitor, labeled C3 in the image, should be placed close to the output path because it acts as the output coupling capacitor, blocking DC while allowing the audio signal to pass to the speaker. Overall, the layout should separate input and output signals, use short traces around the LM386, place capacitors close to the IC, and maintain a solid ground plane for stable audio performance.

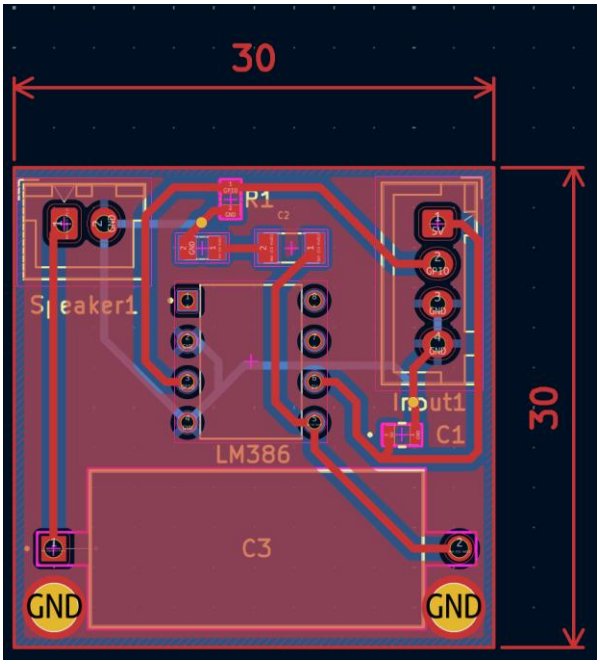

Figure 35: LM386 Amplifier PCB Layout

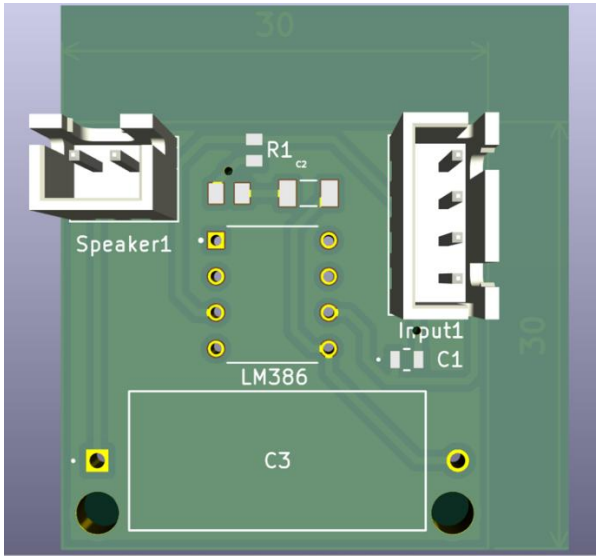

Figure 36: LM386 Amplifier 3D Model

8.4 Motor Driver

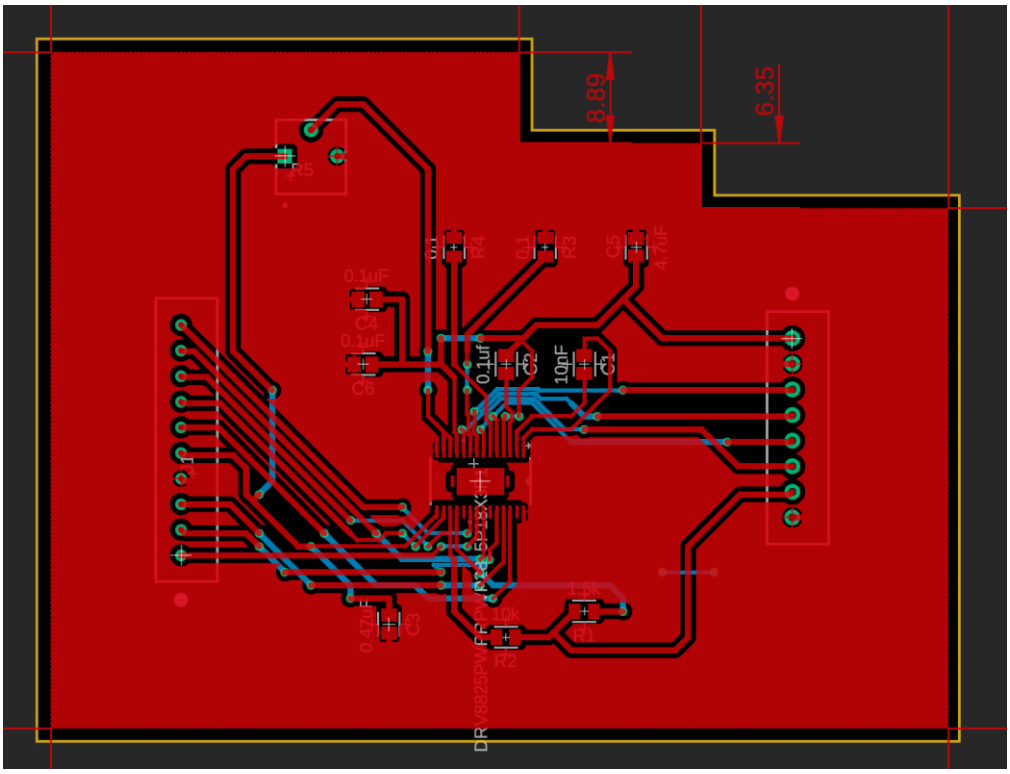

Figure 37: DRV8825 PCB Layout

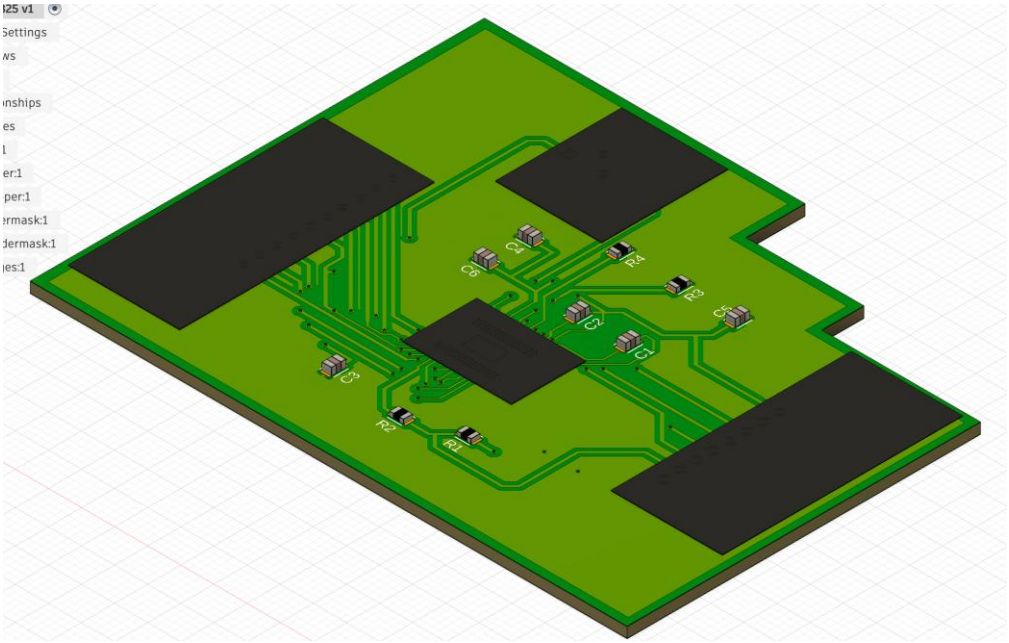

Figure 38: DRV8825 3D Layout

8.5 Main PCB Layout

The main PCB for our MCU plays a vital role in the functionality of our dog feeder. The main PCB will be in charge of holding all of the peripherals or the peripheral headers in order to run and test them. The peripherals that will connected to the main PCB are the camera, amplifier, speaker, regulators (3.3V and 5V), outlet jack, motor driver, ultrasonic sensor, depth module, and programmer.

When designing the main PCB, organization played an important role in placement of each component and peripheral. Peripherals or peripheral headers are needed to be placed in specific spots to ensure that when it is connected to the PCB, they were able to execute the task that we have required for them.

The OV5640 camera and the depth module need to be able to be mounted to the front wall of the dog feeder to ensure that they were able to perform the tasks that we had for them. Tasking includes making sure the dog is in range, being able to identify the dog, and when the dog has sat down. In addition, the user is able to join into the live feed from the dog feeder. Since they are required to do this, it was vital to put these peripherals in a spot that we could easily connect together and mount them on the wall. On the PCB, we placed both the camera header and the header for the depth module towards the front of the board, so that when attaching these peripherals to the PCB, there were no problems that could occur from them being too far away from the front of getting tangled with other peripherals.

The speaker also needed to follow relatively the same rules as the camera header and the depth module. The dog needs to be able to hear from the camera when the user is talking and needs to be able to hear when it is time to be fed. On the PCB, we have the speaker

and the amplifier toward the front right of the board for this reason. Allowing us to have easy access to mounting the speaker on the front of the dog feeder to ensure that the dog can hear from the speaker. Placing the header for the speaker towards the front also allows for each connection to the speaker and connecting it to the wall. If we were to have placed the speaker header in the middle of the board, connecting it to the wall could create problems with functionality and organization.

The motor driver header is placed on the right side of the PCB for these same reasons. We plan to place the physical motor on the right side of the dog feeder which would help to keep the motor at a safe distance from the dog's bowl. By placing the header closer to the motor and the motor driver, it allows for an easier connection between the two of them. Having to connect both the header and the motor to each other when the header is placed on the opposite side of the PCB could create problems with the functionality of the motor and running the connection through other connections.

The ultrasonic sensor is placed at the back right of the PCB based on the location the food storage container will be placed. The sensor we are using will have the purpose of detecting when the container is low on food. When connecting the ultrasonic sensor to the PCB header, we must be able to attach it to the wall of the storage container for it to serve its purpose. The button placement and the outlet jack placements have the same concept. The button will allow food dispensing when the user is at home. Therefore, the user needs to be able to access the button which will be mounted into the side left wall of the dog feeder. Placing the header close to the location of the button helps with organization and functionality of this peripheral. The outlet jack also needs to have access to the wall in order to gain power to the whole system, so placing the header close to the wall solves any problems related to this that could occur.

For the placement of all the components that integrate each peripheral, we emphasized placing them in spots where they could carry out their task while also allowing for easier implementation when the time comes. Having capacitors, resistors, or inductors that are extremely close in length on the physical board could create a problem when we go to glue them. For example, when placing the resistors for the OV5640 camera, we placed them slightly staggered to each other. This would create a bit of space between them so that when glueing the board, there was no problem with bumping into pieces that we had already placed. We made sure to use pin tracing that was 0.02 inches or more. For most of the tracing connections, we use 0.024 inches, and others used 0.020 inches. For the grounding, we used a polygon to connect all of the top layer to ground. This benefited the PCB by ensuring that everything is connected to ground, and also saved time when designing the PCB tracing.

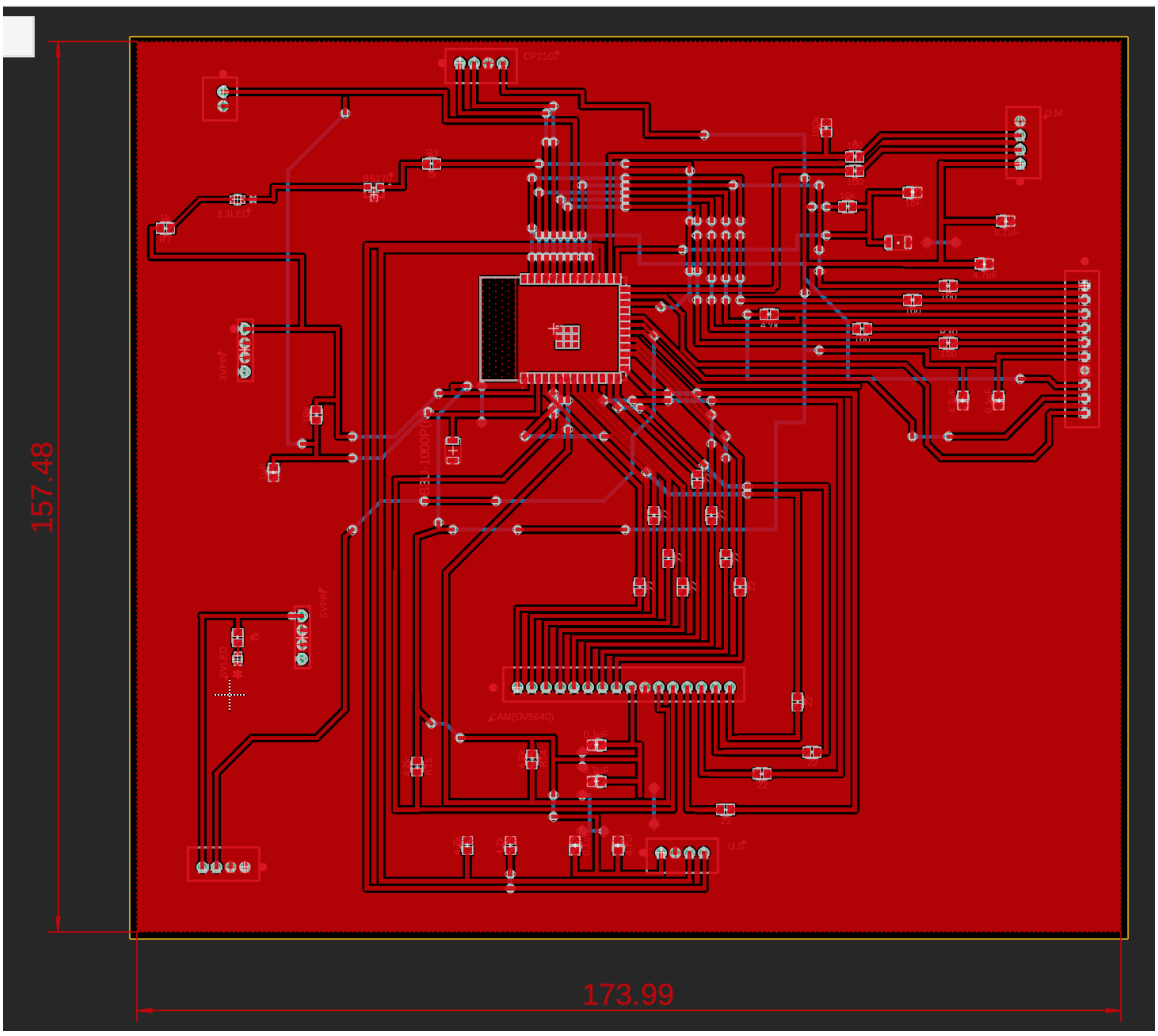

Figure 39: PCB Layout (With Ground Polygon)

Above, Figure 33 shows the main PCB layout of Sit & Chow, showing the component placement, routing, and grounding strategy. The design highlights the dedicated ground plane beneath the ESP32-S3, pin connections, and grounding to minimize overall electrical noise. Below, Figure 34, shows the main PCB layout without grounding, showing a clear view of the routing and tracing between the MCU, peripherals, and power system.

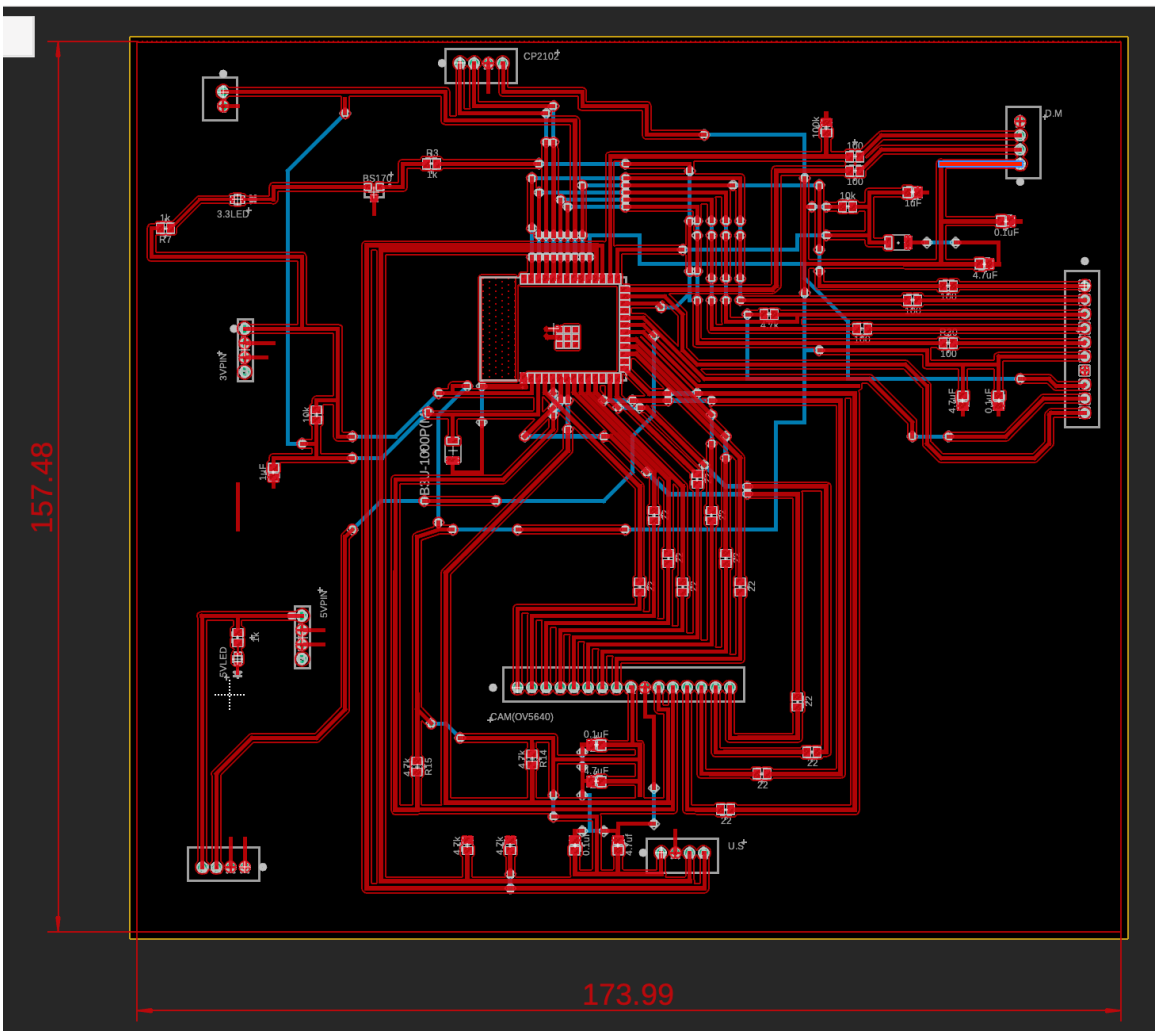

Figure 40: PCB Layout (Ground Polygon Hidden)

Figure 35 below shows the 3D model representation of the main PCB layout for Sit & Chow. It shows the component placement, routing, and connector locations. This visualization highlights the spatial layout of the ESP32-S3, motor driver, sensor headers, power regulation, and other components to provide a clear representation of how the final board will appear assembled.

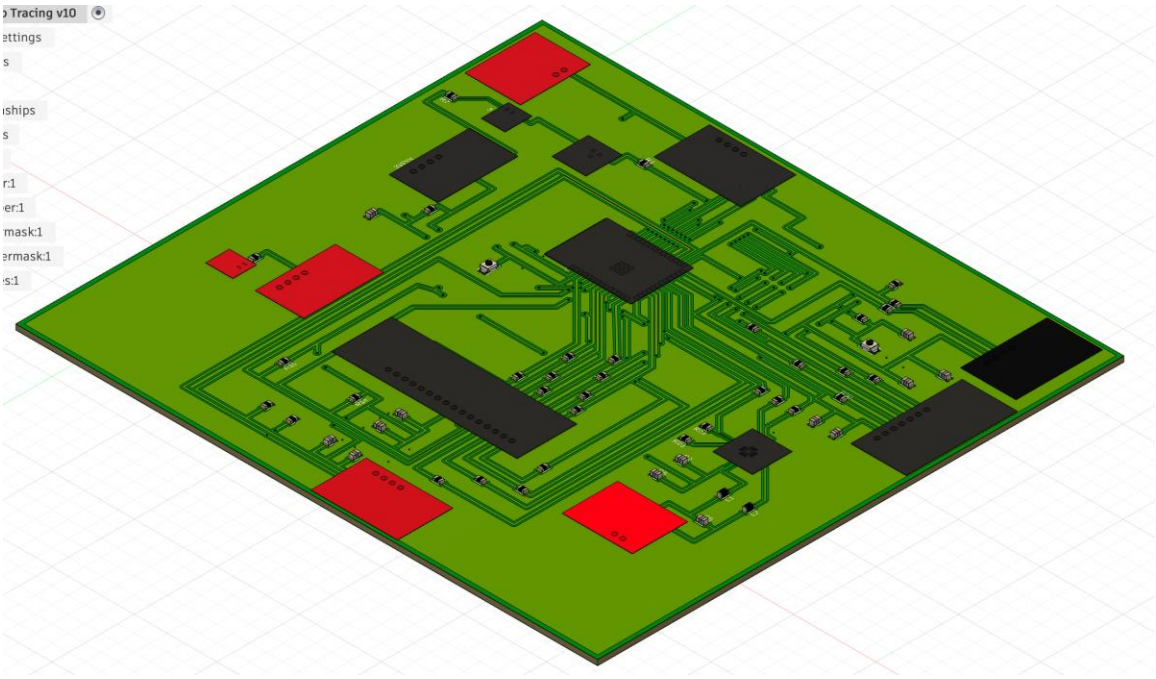

Figure 41: 3D View of PCB

8.6 Case 3D Model

We use OnShape to make 3D Dog Feeder Case Model. After adjusting multiple times, we finally decided how to make it.

This is front view with air flow, camera, ToF sensor, and speaker connected. Also, the small door will be a place to drop the food out.

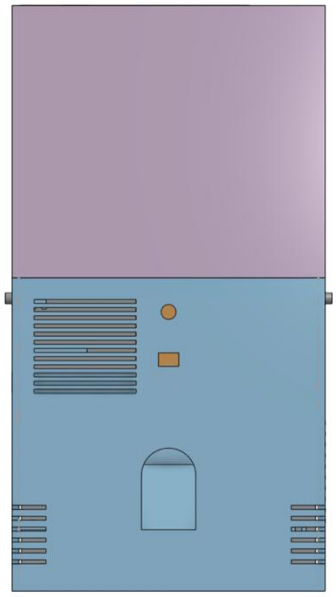

Figure 42: Front View of Dog Feeder Case

For the inside, we plan to have some spaces for PCB boards to slide in. Also the motor will be on top with the gear. Moreover, the food storage will be easily moving to fit with the motor gear.

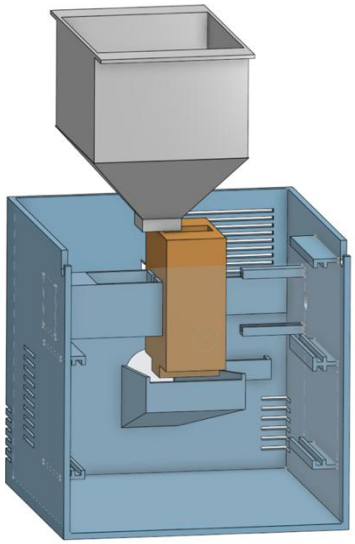

Figure 43: Inside View of Dog Feeder Case

Finally, we built the top with small door to open and remove the food storage in case the user wants to clean food storage.

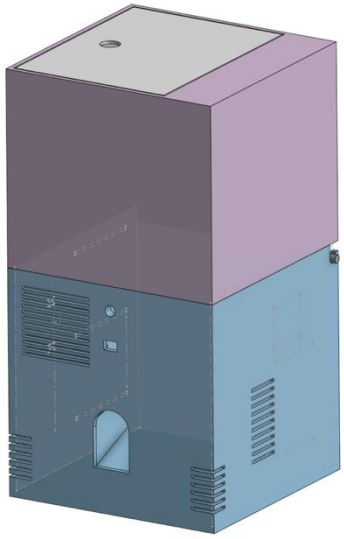

Figure 44: Final View from Outside of Dog Feeder Case

Chapter 9 – System Testing and Evaluation

The picture below (Figure 36) shows most of the parts we ordered with the regulators, motor driver, speaker, and the LEDs missing. The chip on the top right is the TMF8701 ToF sensor, to the left of that is the TS02-66-70-BK-160-LCR-D button, then the RCWL-1601 ultrasonic sensor. Then you can see the TMC2209 motor driver next to the Nema 17 Stepper Motor and the 12V 3A AC to DC Adapter power supply with the CUI PJ-102A power jacks next to them. In the bottom middle of the image is the OV5640 camera and to the right of that is the MCU dev board with an ESP32-S3-N8R8.

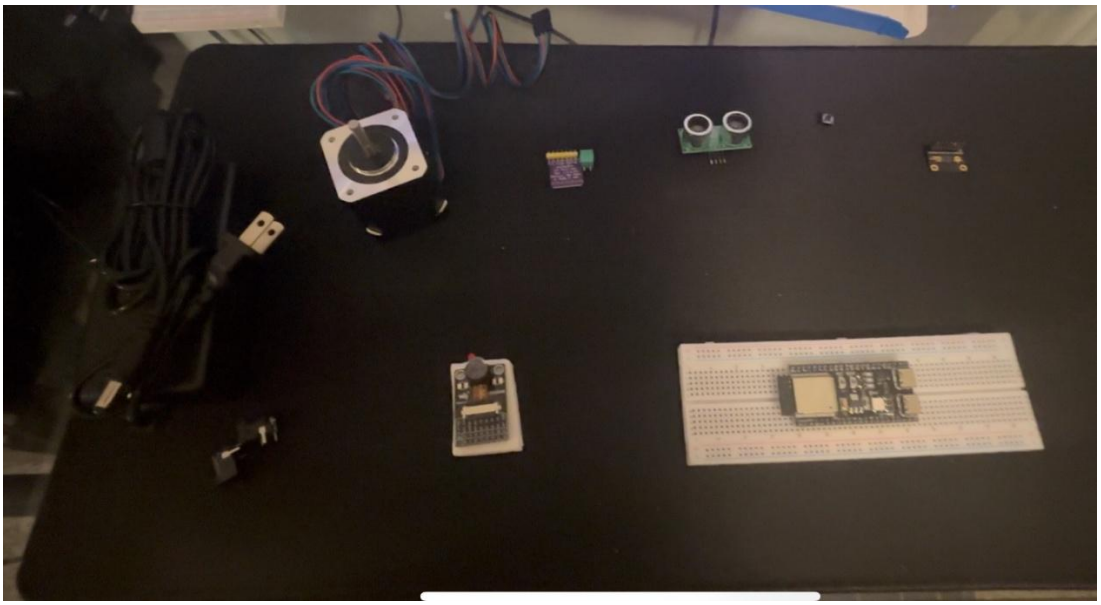

Figure 45: Parts and Peripherals

9.1 Performance Metrics

Here are some performance metrics to ensure Sit & Chow meets minimum reliability standards before the final integration.

MCU and Processing Performance

- Average inference time for the detection and classification model under 200ms per frame.
- Streaming and Wi-Fi broadcasting remained stabe with <40% CPU load

Sensor Accuracy and Stability

- ToF Sensor: Measured error ±1 – 2cm within a 0-60cm range, which was verified in section 9.2.2.
- Ultrasonic Sensor: Stable within ±5 – 10mm for a short distance since it detects how low the food supply is.
- Camera: Achieves stable 15 – 30FPS under typical indoor lighting.

Power and Thermal Stability

- 3.3V and 5V regulators maintained stable output under a moderate load.
- No overheating occurred during MCU operation.

9.2 Hardware Testing

9.2.1 ESP32-S3

In order to test whether or not the ESP32-S3 we purchased was functional, we needed to connect it to the computer through a UCB-C cable. Once connected, we were able to see that the red and green LEDs were turned on. We verified the functionality of the reset and boot buttons on the ESP32 by pressing both of them. We also tested the voltages of the GPIO pins through a digital multimeter.

9.2.2 ToF Sensor

Testing the functionality of the ToF sensor's hardware also required testing the software aspect. There are no LEDs or other peripherals that indicate if it is powered or not. The ToF sensor did not come with pins soldered to them. So, in addition to not having any indication of power, we also had to trust that the soldering connections we made were sufficient. In order to test whether or not it was connected, we did not need to make the sensor work, we simply needed to test whether or not an I2C connection was made through the GPIO pins. We had to connect the sensor's VCC to 3.3V, GND to GND, GPIO4 to SCL, and GPIO5 to SDA. Once connected, we verified we had an I2C connection on the MCU and saw that the ToF sensor was functional.

9.2.3 OV5640 Camera

Similarly to the ToF sensor, in order to verify that the camera's hardware was functional, we needed to use software as there are no peripherals on the camera itself that indicate functionality. The camera has many pins that need to be connected to. These include VCC, GND, SIOC, SIOD, VSYNC, HREF, PCLK, XCLK, D9-D2, RST, and PWDN. We did not need to connect RST or PWDN, but we connected VCC to 3.3V, GND to GND, and the rest to GPIO pins. Since there were so many pins to be connected, it was likely that a faulty connection would occur. So, we needed to triple check our connections before testing. After checking, we indeed found that there were some incorrect connections and adjusted them accordingly. Once we verified our connections, it was time to move on to the software. After testing the software we were able to verify that the camera was in fact working.

9.2.4 Ultrasonic Sensor

To test the hardware functionality of the ultrasonic sensor, we needed to connect it to the proper pins on the MCU. We connected VCC to 3.3V, GND to GND, TRIG to GPIO5, and ECHO to GPIO18. To test whether or not it was properly connected, we needed to also test the software. The connection was verified once we observed the proper behavior coming from the ultrasonic sensor in the software.

9.2.5 Amplifier and Speaker

In order to test the amplifier and speaker was working properly, we attached the amplifier to its respective GPIO pins so that the audio data could be properly streamed and also attached 3.3V and GND. After that, there were two phases of audio testing. The first was testing the amplifier and speaker with a simple tone and the second was playing an audio recording. We also tested the amplifier with an oscilloscope to make sure that a sin wave was being properly generated.

9.2.6 Motor and Motor Driver

In order to test the motor and motor driver, we connected the motor driver to the esp32 with 3.3V and GND and then connected several GPIOs to configure the motor. We connected a STEP, DIR, EN, SLP, and RST pin to the esp32 and also hardwired M0, M1, and M2 to either high and low. The STEP, DIR, and EN pins are used to enable, set the amount of turns, and set the direction of the motor. The SLP and RST pins must be connected and set to high so as to keep the motor on. The M0, M1, and M2 pins are used to configure microstepping, which allows for finer control of the motor. After connecting these pins we also had to make sure that the motor driver was properly connected to the motor via its A1, A2, B1, and B2 pins. If these were connected improperly, the motor would not move. We also had to have a 12V supply and a common GND sent to the motor or else not enough power would be generated to move the motor. After setting up all of these connections, we tested the motor with the software and it was functional.

9.2.7 Voltage Regulators

The 3.3V and 5V regulators will be attached to a 12V power supply. Using a multimeter, the input pin is measured to ensure that the regulators are operating within its rated input range. Once the input is confirmed, the output voltage of each regulator is measured with no load attached. Since the regulators are 3.3V and 5V, the output should read around the same values.

Each regulator will also be tested under a load to verify that they can supply current to other hardware components. A test load is applied with either one of the hardware components or a high-wattage resistor. With the load, the regulator’s output voltage will be measured to confirm the connection is stable.

9.2.8 Full PCB and Peripherals

In order to test the full PCB, we first made sure that we could talk to the ESP32 via a CP2102 programmer. We first needed to connect the 12V power supply to our regulators and then connect those regulators to the 3.3V and 5V headers on the main board. This powered everything connected to the PCB. After that, we needed to set the ESP to boot mode every time in order to flash the program, but after testing the programmer and making sure it was functional, we saw the ESP on the board was working properly. After this, to make sure each peripheral was working, we connected each individually and tested the previously generated demo code. Once each peripheral was verified to be functional, we started to add multiple at a time and made sure that the main program was working with each of them properly. After running into some road blocks, some adjustments with both the hardware and the software needed to be made. We had to change some GPIO pins on the ToF sensor and Ultrasonic sensor to be the M0, M1, and M2 pins on the motor driver since they were connected to pins only meant for PSRAM. We also had to keep the wires connected to the camera extremely short since the data being transmitted was extremely sensitive and would corrupt if the wires were too long or too bent. After making these adjustments all parts of the project were tested and confirmed to be working.

9.3 Software Testing

9.3.1 ESP32-S3

In order to test the software functionality of the ESP32, we used VS Code with the ESP IDF plugin. We did not want to use Arduino as our IDE because of its simplicity and lack of control. Using the ESP IDF allows us to have more control and customization over our programs. We chose to use an example from the ESP IDF example library given to us. We chose the blink program. To set up the example project, we first needed to choose our specific ESP32 for setup. We then went to the ESP IDF command prompt, went to our project directory, and then ran idf.py build. This builds the project to run properly in the desired directory. Once we ran this, we then built our project in VS Code itself, flashed it onto the ESP32-S3, and then monitored its performance. As long as there were no mistakes during the setup process, this let us run the program on our ESP32-S3 and verify its functionality.

9.3.2 ToF Sensor

To test the functionality of the ToF sensor, I needed to see how the manufacturer wants me to use the sensor. So, I looked at the data sheet to get the pin layout and what should be connected. I saw that this device uses I2C to communicate with other devices. After finding that out, I used an example from the ESP IDF to set up I2C connections. At this point I did not have the sensor soldered so no connection was detected but once soldered I used the sensors standard I2C address, and a connection was made. In order to get the sensor working as it should, I needed to find an example of how to use it. The manufacturers themselves have an example in Arduino code. Unfortunately, I did not want to use Arduino code, so I needed to convert all of it into code for the ESP IDF. This did not work at first, like one would expect, but after many bug fixes I saw that it was working. The only issue was the sensor was not calibrated properly. In the same github repository that the examples were on, there is also code to get the calibration for the sensor. Once I converted that code and used it, I got the proper calibration metrics for the sensor. Once calibrated, the sensor was detecting properly within the expected range of 0-60 cm.

9.3.3 OV5640 Camera

The OV5640 is a camera that has great compatibility with the ESP32-S3. There are already protocols to set up and communicate with the camera with minimal hassle. To start this process, I originally looked that the examples within the ESP IDF. Unfortunately, the examples that are there do not work with the ESP32-S3, except for one that requires an additional LCD, which we do not have. So, I sought to look in other areas for other examples. I found a set of examples of using a camera with the ESP32-S3 in a GitHub repository by ESP themselves. What this example does is create a Wi-Fi signal that is broadcast to other devices, sets up a website, and then continuously feeds the video stream from the camera to that website. I then brought one of the examples into my local system and set it up for my own devices. I also needed to bring in extra header files that were not included in the example. Once I did that, I configured the pins in the header files so that I could connect the camera through the desired GPIO pins. I then connected the OV5640 to the ESP32-S3 and verified the connection. Then, I connected my computer to the Wi-Fi signal broadcasted by the ESP32, went to the website at http://192.168.4.1/stream, and then observed the video stream from the camera.

9.3.4 Ultrasonic Sensor

To test the functionality of the ultrasonic sensor, we created a program that detected the distance between the sensor and an object in front of it. We began by setting the TRIG and ECHO pins to their respective GPIO pins and then creating a global variable for the distance traveled at the speed of sound for every microsecond. Then, we setup the GPIO pins for the TRIG and ECHO pins, setting the TRIG pin as an output and the ECHO pin as an input. After that, we begin the loop that starts by sending out a trigger pulse. We set a timeout variable to make sure we don't wait for the echo forever and wait for the echo to be received and track the time it takes to receive it. We then use that time and the previously declared speed of sound variable to calculate the distance through this formula: distance_cm = (duration_us * SOUND_SPEED_CM_PER_US) / 2.0f. We then print the distance to the monitor. After testing this with the ultrasonic sensor, we were able to observe that it was working properly and displaying the distances accurately.

9.3.5 Amplifier and Speaker

To test the amplifier and speaker, a dedicated audio playback driver was developed using the ESP-IDF LEDC peripheral. The driver (audio_player.c) initializes a LEDC timer and channel at a 20 kHz PWM carrier frequency with 8-bit resolution, configured on the speaker GPIO. Audio samples are stored as a pre-converted int16_t array in a header file (audio_data.h) and played back at 16 kHz. The test program (app_main) initializes the player with audio_player_init(13, 20000, 16000), then calls audio_player_start() passing the sample buffer and count. Playback runs on a dedicated FreeRTOS task pinned to core 1 to avoid interference with Wi-Fi and Bluetooth on core 0. A busy-wait loop timed with esp_timer_get_time() ensures precise per-sample pacing at the correct 16 kHz rate. The main task polls audio_player_is_playing() every 500 ms and waits for the task to complete. Successful execution was confirmed by audible playback of the test audio clip through the LM386 amplifier and speaker at the expected quality and duration.

9.3.6 Motor and Motor Driver

To test the stepper motor and DRV8825 driver, a standalone firmware program was written using ESP-IDF GPIO and timing primitives. The test configures five control pins — STEP, DIR, EN, SLEEP, and RESET — as outputs and brings SLEEP and RESET high to enable the driver. A trapezoidal speed profile function (stepper_move_ramp) was implemented to ramp the step period from a slow start (4× the target period) down to the cruise speed and back, preventing missed steps due to abrupt velocity changes. Step pulses are generated with a 3 µs high time followed by the remaining period delay, using ets_delay_us for precise timing. The motor was commanded to execute one clockwise revolution at 20 RPM using stepper_rotate(1, 20, true), which translates the revolution and RPM parameters into the correct number of microsteps and target step period. The motor enable pin is asserted before motion and de-asserted after to cut holding current while idle. Correct rotation direction and smooth acceleration were verified visually on the bench.

9.3.8 Detection and Classification Algorithm

Testing the detection and classification pipeline includes:

Running diverse video clips of different breeds, sizes, lighting, distance, and orientations.

- Comparing predicted bounding boxes to expected bounding boxes to compute detection accuracy.
- Evaluating classification accuracy by comparing predicted posture labels to true labels.
- Testing video clips with a dog moving between poses to verify a hold-time of at least 2 seconds.

In Senior Design 2, the classifier was evaluated with accuracy, precision, recall, and confusion matrix metrics. The detection and classification models were tested using a couple of different dog plushies and toys to simulate a real-life environment. In the end, due to camera troubles, the five-second timer was removed since the YOLO model had trouble detecting on the new camera stream.

9.3.9 Full Framework

The full Sit-N-Chow system was tested as an integrated pipeline spanning the ESP32-S3 firmware, Firebase Realtime Database, and the Flutter mobile app. Testing was performed by exercising each system path end-to-end and confirming correct behavior at every stage.

The command dispatch path was verified by issuing a "Feed Now" command from the Flutter app's home page, which writes a dispense action and gram amount to commands/{deviceId}/pending in Firebase RTDB. The ESP32's polling task detects the new node, acknowledges it by deleting the pending entry, and executes the full feed workflow: the TOF sensor checks for pet presence, the stepper motor dispenses the requested gram amount, the ultrasonic sensor reads the post-dispense food level, and a feed event is logged back to Firebase. If the food level falls below the configured threshold, an additional low-food flag is written to devices/{deviceId}/inventory/lowFood and a separate FCM push notification is dispatched through the Cloud Run backend.

The scheduled feeding path was tested by creating schedules in the Flutter app and confirming that the ESP32's schedule_runner task, which wakes every minute to compare the current time against stored schedules in userSchedules/{uid}, correctly fires the feed workflow at the configured time and day without manual intervention.

The audio intercom was tested by opening the camera live page in the Flutter app and toggling the microphone on. The app records rolling PCM chunks, strips WAV headers, base64-encodes the raw PCM, and writes each chunk sequentially to devices/{deviceId}/audio in RTDB with an incrementing chunkIndex. The ESP32's audio_intercom task polls this node each second, detects a new chunk via the changed index, base64-decodes the payload using mbedTLS into a PSRAM-backed buffer, and plays the audio through the speaker. Successful end-to-end voice playback was confirmed with audible output matching the recorded speech. Toggling the mic off in the app deletes the audio node, and the ESP32 task correctly detects the absence and halts playback.

9.4 Senior Design II Testing and Validation Plan

In Senior Design II, the focus is full system integration, reliability testing, and optimizing hardware and software performance. This phase will include completing untested subsystems, optimizing the detection and classification pipeline, and validating feeding consistency in order to ensure seamless coordination between the embedded hardware, inference processing, and mobile application.

Senior Design II Tasks

Motor Driver Integration

- Implement UART communication to configure stepping, motor current, and step resolution.
- Validate stepper motor rotation, torque, and responsiveness

Full Power Delivery System

- Integrate regulators, buck converters, and 12V power supply together.

- Test voltage stability during simultaneous camera streaming, Wi-Fi communication, sensor polling, and motor activation.
- Measure thermal output to ensure the power system does not exceed operating temperatures.

Dispensing Mechanism Testing

- Assemble hopper and slotted wheel driven by the Nema17.
- Conduct repeating tests to measure how consistently the system dispenses food portions.
- Identify potential failures such as jamming, slippage, or under/over-dispensing.

Integration Testing

- Validate interoperability between sensors, motor, camera, inference and MCU controls.
- Conduct full-cycle tests where the dog sits, the system detects, classification confirms, motor dispenses, and logs are recorded.

Environmental and Stress Testing

- Test performance in varied lighting conditions, distances, and orientations.
- Runs multiple tests of the inference waiting the full time (1 minute) for the dog to sit to monitor stability, communication losses, and thermal output.
- Simulate network loss or sensor failures to validate the system can fail safely.

Mobile Application

- Finalize integration for scheduling, streaming, and alerts.
- Confirm synchronization between scheduled feeding times, feeding outcomes, and log updates.
- Test app under network delays.

Final Testing

- Compare all system performance to the specification table (Table 1)
- Verify accuracy thresholds for detection and classification, dispensing, sensor ranges, and electrical safety.
- Document limitation, propose future improvements, and confirm readiness for demonstration.

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 34: Bill of Materials

Component	Part name	Quantity	Unit price	Total price
Servo/Stepper Motor	Nema 17 Stepper Motor	1	$3.33	
Motor Driver	DRV8825	1	4.59	
LEDs	ROHM SML-211UTT86	2	$0.52	$1.04
Power Supply (12V adapter)	Facmogu 12V 3A adapter	1	$9.99	
Power Jack	CUI PJ-102A	1	$0.65	
Camera	OV5640	1	$13.76	
5V Voltage Regulator	LM2576-ADJ	1	9.32	
3.3V Voltage Regulator	LM2576-ADJ	1	9.32	
Speaker	Adafruit Speaker - 3" Diameter - 4 Ohm 3 Watt	1	$1.95	
Amplifier	LM386	1	5.95	
Programmer	CP2102	1	8.88	
Ultrasonic Sensor	RCWL-1601	1	$3.95	

Depth Module	TMF8701 ToF	1	$13	
PCB boards			$252.83	
Microcontroller	ESP32-S3-WROOM-2-N32R8V	1	$10.67	
Mechanical Assets (Casing, feeding mechanism, etc.)	PLA Filament	6		125.94
Total Cost				
Grand Total	~474.63			

10.3 Milestones

10.3.1 Senior Design 1 Milestones

Table 35: 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 36: 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 37: 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	Natalie Nguyen

	-Review with advisor	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

Chapter 11 – Conclusion

This dog feeder project began with a problem: some owners are too busy to train their dog food obedience. However, what started as a personal idea grew into a serious engineering journey that challenges in every direction: mechanical design, power regulation, PCB layout, embedded firmware, sensor research, software logic, testing, and real-world reliability. Through this process, we learn that engineering is not just about building a circuit that works once, it is about designing a system that works every day, and in conditions that are not always ideal. Life can be busy, and taking care of and training your pet can take a lot of time and effort. Sit & Chow presents a solution to automated pet feeding by reinforcing food obedience training at the same time.

The Sit & Chow smart dog training feeder is a system that integrates embedded systems, machine learning, and mobile application development. This report explores motivations, design goals and objects, hardware and software research that offers the foundation of this project, Sit & Chow. The system reinforces obedience behavior for dogs while maintaining accurate and consistent feeding. By combining automated feeding schedules with posture-based training reinforcement, Sit & Chow expresses innovation in convenience, interaction, and reliability.

There were many ways to supply power, and extensive research was done to determine the most efficient, sustainable power system. Sit & Chows power system was designed to support high-current components, such as the Nema 17 stepper motor, with the 12V power supply. Buck converters and LDOs provide regulated power at 5V and 3.3V for stable power to more sensitive circuits, such as the ToF sensor. However, thermal management is necessary with continuous operation, so hardware components were chosen to ensure the system minimizes heat.

Sit & Chow has the ability to verify posture behavior before dispensing food due to an attached camera. This requirement guided nearly every engineering decision, from selecting hardware components, sensors, dispensing accuracy, and machine learning performance. By thoroughly researching and validating each component that was chosen, Sit & Chow is not only functional but reliable, scalable, and cost-effective.

The embedded design was shaped heavily by the constraints of microcontroller-based designs. Evaluating between the FPGA and MCU demonstrated that an MCU-based architecture offers the best balance of capability, simplicity, and feasibility. The ESP32-S3 has built-in Wi-Fi, low-power operation, and sufficient peripheral support, and it became the central controller for sensor integration, motor actuation, communication to the cloud server, and more. However, the ESP32-S3's processing and memory limitations caused the decision to offload the detection and classification models to the cloud via Firebase. Understanding these limitations early allowed the team to construct a realistic, optimized division between firmware and backend tasks. The resulting architecture supports posture detection, reliable scheduling, efficient logging, and a robust mobile interface along with the embedded hardware.

Similarly, the mobile application serves as an essential user interface to connect the owner, dog, and feeder at any time of day. Firebase was the backend service chosen for authentication, real-time data synchronization, and scheduled command management

while Flutter offers a cross-platform interface for schedule management, live camera viewing, notifications, and other interface features. These features allow the user to have full control over feeding routines, even away from home. In addition, cloud communication and mobile integration reinforce system safety by allowing notifications that alert the user of failed feeding cycles and dispensing issues.

The most important goal of this project was reliability. A dog feeder cannot simply work most of the time. It must work when the user is not home, when Wi-Fi is unstable, and when the motor overheats. That is why we emphasize features such as sensor-based hopper monitoring, secured power delivery, silent motor control, and jam detection with TMC2209. These features do not just make the feeder smart, they also make it trustworthy, which is the real standard of engineering success.

The milestones outlined in the project timeline showcase the tasks, such as designing system diagrams and schematics, researching hardware and software, and procuring components for testing and assembly. Future phases focus on firmware development, mobile app construction, and system testing to create a finished prototype to demonstrate. This progression demonstrates the collaboration between the electrical and computer engineering team members coordinating complex parallel tasks across hardware and software.

When completed, Sit & Chow will not only automate feeding but also support daily training. The system's combination of automation, machine learning, and remote monitoring showcases the application of theoretical knowledge to a real-world challenge. The progress documented in this report demonstrates the foundation and vision to deliver a product that is robust, user-friendly, and helpful in improving the daily lives of both dogs and their owners.

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] "Arrow," 05 10 2018. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/fpga-vs-cpu-vs-gpu-vs-microcontroller.

[6] J. Schneider, "IBM," [Online]. Available: https://www.ibm.com/think/topics/fpga-vs-microcontroller.

[7] R. Ravi, "Wevolver," 30 05 2025. [Online]. Available: https://www.wevolver.com/article/fpga-vs-microcontroller.

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

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

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

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

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

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

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

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

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

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

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

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

[20] "Espressif," [Online]. Available: https://documentation.espressif.com/esp32-s3_datasheet_en.pdf.

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

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

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

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

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

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

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

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

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

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

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

[32] "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.

[33] "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.

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

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

[36] "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/.

[37] "Pololu," 2014. [Online]. Available: https://www.pololu.com/file/0j450/a4988_dmos_microstepping_driver_with_translator.pdf.

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

[39] "Handsontec," [Online]. Available: https://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf.

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

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

[42] "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.

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

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

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

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

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

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

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

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

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

[52] "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.

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

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

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

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

[57] "Texas Instruments," 08 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm386.pdf.

[58] "Texas Instruments," 05 2004. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm1875.pdf.

[59] "Texas Instruments," 12 2025. [Online]. Available: https://www.ti.com/lit/ds/symlink/ne5532.pdf.

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

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

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

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

[64] C. Rutz, "Hirosart," 08 07 2025. [Online]. Available: https://hirosarts.com/blog/led-vs-smd-led-vs-cob-led/?srsltid=AfmBOoosrgngOzJEH7EU3TbG2JSwr0nx7G1SLZ_RVA3zfmYz91rqggjD.

[65] "Future Electronics," ROHM, [Online]. Available: https://www.futureelectronics.com/p/semiconductors--optoelectronics--leds/sml-211utt86-rohm-6473904.

[66] "TME," [Online]. Available: https://www.tme.com/us/en-us/details/kp-2012sgc/smd-colour-leds/kingbright-electronic/.

[67] "Mouser," Lumex, [Online]. Available: https://www.mouser.com/ProductDetail/Lumex/SML-LX0805UWC-TR?qs=26yrLECgo495e2ft2n1zkA%3D%3D.

[68] "Sparkfun," [Online]. Available: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF.

[69] "Sparkfun," [Online]. Available: https://docs.sparkfun.com/SparkFun_RTK_Facet_mosaic/assets/component_documentation/CH340DS1.PDF.

[70] "FTDIchip," 21 05 2020. [Online]. Available: https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf.

[71] "SiliconLabs," 20 01 2017. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf.

[72] "ACT Advanced Control Technology," 27 09 2024. [Online]. Available: https://actsensors.com/blogs/resources/capacitance-level-sensor-advantages-and-disadvantages.

[73] J. S. Cook, "Arrow," 04 04 2018. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/ultrasonic-sensors-how-they-work-and-how-to-use-them-with-arduino.

[74] "Adafruit," [Online]. Available: https://www.adafruit.com/product/4007.

[75] "Ardupilot," [Online]. Available: https://ardupilot.org/copter/docs/common-jsn-sr04t.html.

[76] "Dfrobot," [Online]. Available: https://www.dfrobot.com/product-1935.html.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[93] I. C. Bertolotti, "MDPI," 19 06 2025. [Online]. Available: https://www.mdpi.com/1999-5903/17/6/269.

[94] "Freertos," 2017. [Online]. Available: https://forums.freertos.org/t/combine-freertos-tasks-and-c-objects/6414/2.

[95] "Stackoverflow," [Online]. Available: https://stackoverflow.com/questions/62506302/why-developers-use-c-c-for-embedded-systems-rather-than-high-level-language-li.

[96] I. Plauska, A. Liutkevičius and A. Janavičiūtė, "MDPI," 28 12 2022. [Online]. Available: https://www.mdpi.com/2079-9292/12/1/143.

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

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

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

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

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

[102] 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.

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

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

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

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

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

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

[109] 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.

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

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

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

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

[114] "AWS," [Online]. Available: https://aws.amazon.com/compliance/iso-27018-faqs/.

[115] D. L. PAN, "LARGE," 24 04 2025. [Online]. Available: https://www.large-battery.com/blog/portable-battery-charger-vs-power-bank/.

[116] A. Herrera, "Keysight," 21 02 2024. [Online]. Available: https://www.keysight.com/blogs/en/tech/educ/2023/bench-power-supply.

[117] M. Subbaroybhat, "RS," 12 2021. [Online]. Available: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/ac-dc-adapters-guide.

[118] "Grepow," 30 01 2024. [Online]. Available: https://www.grepow.com/blog/what-is-a-lithium-ion-battery-cell-battery-module-and-battery-pack.html.

[119] A. Vinci, "Tektronix," 25 03 2025. [Online]. Available: https://www.tek.com/en/blog/what-is-a-switching-power-supply.

[120] E. Walker, "Energysage," 03 04 2025. [Online]. Available: https://www.energysage.com/solar/solar-panels-work/.

[121] "Mouser," Mouser, [Online]. Available: https://www.mouser.com/c/power/power-supplies/plug-in-ac-adapters/wall-mount-ac-adapters/.

[122] "LEDsupply," MEAN WELL, [Online]. Available: https://www.ledsupply.com/power-supplies/mean-well-gst-desktop-adaptor.

[123] "Triadmagnetics," [Online]. Available: https://www.triadmagnetics.com/products/wall-plug-in-transformers/.

[124] "Mouser," CUI Inc, [Online]. Available: https://www.mouser.com/ProductDetail/CUI-Inc/SMI36-12-E-P5?qs=7yuRwnUml9ztMaoEdcVYLg%3D%3D.

[125] "Datasheets," [Online]. Available: https://www.datasheets.com/cui/smm30-12-4-p5.

[126] "LEDsupply," MEAN WELL, [Online]. Available: https://www.ledsupply.com/power-supplies/mean-well-gst-desktop-adaptor.

[127] "Mouser," CUI, [Online]. Available: https://www.mouser.com/c/power/power-supplies/plug-in-ac-adapters/wall-mount-ac-adapters/?m=CUI%20Inc.&series=SMI36&srsltid=AfmBOooAKoV9nCnNTObhd1UYaRWs4T_Rr5C2_NX_ccyguntrcppbTTwq.

[128] "onlinecomponents," Mean Well, [Online]. Available: https://www.onlinecomponents.com/en/productdetail/mean-well/gst36u12p1j-48609701.html.

[129] "Amazon," Facmogu, [Online]. Available: https://www.amazon.com/dp/B073WSWT34?th=1.

[130] B. Rose, "Bel," 22 12 2020. [Online]. Available: https://www.belfuse.com/resource-library/blog/selecting-appropriate-input-and-output-plugs-for-your-power-adapter.

[131] "Switchcraft," [Online]. Available: https://www.switchcraft.com/bkz-series-dc-power-jacks-and-plugs/.

[132] "Dalroad," 26 11 2024. [Online]. Available: https://www.dalroad.com/resources/sealed-connectors-for-electric-vehicles-operating-in-harsh-environments/.

[133] "Digikey," 25 09 2023. [Online]. Available: https://www.digikey.com/en/product-highlight/g/globtek/twist-locking-barrel-connectors.

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

[135] "Digikey," [Online]. Available: https://www.digikey.com/en/products/detail/kycon-inc/KLDX-0202-AC/9990098.

[136] "Switchcraft," [Online]. Available: https://www.switchcraft.com/bkz-series-dc-power-jacks-and-plugs/.

[137] "Mingch," 21 04 2025. [Online]. Available: https://www.mingchele.com/blog/voltage-regulator/linear-regulator-vs-switching-regulator/.

[138] R. Stull, "Bel," 08 12 2020. [Online]. Available: https://www.belfuse.com/resource-library/blog/a-comparison-between-dc-switching-regulators-and-linear-regulators.

[139] "Medium," 04 10 2025. [Online]. Available: https://medium.com/@aslikurt/power-electronics-nonsynchronous-and-synchronous-converter-b24c5b004d98.

[140] "Renesas," Renesas Electronics, 23 03 2023. [Online]. Available: https://www.renesas.com/en/document/apn/r34an0006-linear-vs-switching-regulators-taking-accurate-efficiency-measurements.

[141] S. G. B. M. R. &. H. S. H. SANI, "Tubitak," 2023. [Online]. Available: https://journals.tubitak.gov.tr/elektrik/vol31/iss3/4/.

[142] "Oku Electronics," [Online]. Available: https://www.okuelectronics.com/store/sensors-modules-shields/lm2596-lm2596s-dc-dc-step-down-buck-converter-module-without-smd-led.

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

[144] "Texas Instruments," 10 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm22678.pdf.

[145] "Texas Instruments," 02 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2576.pdf.

[146] "FDA," 17 02 2022. [Online]. Available: https://www.fda.gov/animal-veterinary/animal-health-literacy/fdas-regulation-pet-food.

[147] "Special Chem," 11 06 2025. [Online]. Available: https://www.specialchem.com/plastics/guide/flammability-ul94.

[148] "Professionalplastics," 06 2004. [Online]. Available: https://www.professionalplastics.com/professionalplastics/4910ansi1-2004.pdf.

[149] C. Jim, "Texas Instruments," 05 2024. [Online]. Available: https://www.ti.com/lit/an/spradi6/spradi6.pdf.

[150] . A. Schirn, "ANSI," 01 02 2024. [Online]. Available: https://blog.ansi.org/ansi/ansi-z535-4-2023-product-safety-sign-or-label/.

[151] "ASTM," 10 12 2019. [Online]. Available: https://store.astm.org/b0117-19.html.

[152] "ISO," 11 2010. [Online]. Available: https://www.iso.org/standard/51528.html.

[153] "ASTM," 10 12 2019. [Online]. Available: https://store.astm.org/b0117-19.html.

[154] "ISO," 10 2022. [Online]. Available: https://www.iso.org/standard/27001.

[155] "ISO," 03 2024. [Online]. Available: https://www.iso.org/standard/78175.html.

[156] "ISO," 01 2022. [Online]. Available: https://www.iso.org/standard/81291.html.

[157] "IEEE SA," 26 02 2021. [Online]. Available: https://standards.ieee.org/ieee/802.11/7028.

[158] "ISO," 2015, 12. [Online]. Available: https://www.iso.org/standard/43757.html.

[159] "ISO," 03 2022. [Online]. Available: https://www.iso.org/standard/73514.html.

[160] "ISO," 02 2019. [Online]. Available: https://www.iso.org/standard/68305.html.

[161] "ISO," 05 2020. [Online]. Available: https://www.iso.org/standard/77608.html.

[162] "ISO," 05 2020. [Online]. Available: https://www.iso.org/standard/77608.html.

[163] "ISO," 04 2012. [Online]. Available: https://www.iso.org/standard/59579.html.

[164] "Adafruit," Adafruit, [Online]. Available: https://www.adafruit.com/product/5840.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[186] A. Rico, "Batterystuff," 15 10 2025. [Online]. Available: https://www.batterystuff.com/blog/understanding-the-benefits-of-aa-alkaline-batteries.html.

[187] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Nickel–metal_hydride_battery.

[188] "Adi," [Online]. Available: https://www.adiglobaldistribution.us/articles-and-resources/what-are-sla-batteries.

[189] "Homedepot," [Online]. Available: https://www.homedepot.com/p/MIGHTY-MAX-BATTERY-12-Volt-1-3-Ah-SLA-Sealed-Lead-Acid-AGM-Type-Replacement-Battery-for-Alarm-Security-Systems-2-Pack-ML1-3-12MP2/309573194.

[190] E. Kume, "LBP," 11 06 2024. [Online]. Available: https://shop.lithiumbatterypower.com/blogs/news/comparing-lithium-batteries-to-lead-acid-and-nickel-metal-hydride-batteries.

[191] "Amazon," Panasonic, [Online]. Available: https://www.amazon.com/Eneloop-Rechargeable-Batteries-Controller-Flashlight/dp/B00JHKSN76.

[192] "Tenergy," [Online]. Available: https://power.tenergy.com/tenergy-aa-2000mah-nimh-flat-top-rechargeable-battery/.

[193] "Onlybatteies," Power Portable, [Online]. Available: https://onlybatteries.com/50-pack-aa-nimh-2000-mah-button-top-batteries/.

[194] "Batteries Inc," [Online]. Available: https://batteriesinc.net/nimh-batteries-explained/.

[195] "Digikey," 06 2020. [Online]. Available: https://forum.digikey.com/t/schottky-vs-standard-diode/6953.

[196] "Texas Instrument," [Online]. Available: https://www.ti.com/product/LM74610-Q1.

[197] P. Sachdev, "Analog Devices," 29 04 2016. [Online]. Available: https://www.analog.com/en/resources/technical-articles/primer-on-powerpath-controllers-ideal-diodes-prioritizers.html.

[198] J. Smoot, "Digikey," 13 02 2024. [Online]. Available: https://www.digikey.com/en/articles/power-relays-understanding-the-basics.

[199] E. Pino, "Texas Instruments," [Online]. Available: https://e2e.ti.com/support/power-management-group/power-management/f/power-management-forum/790809/lm74610-q1-mosfet-selection-requirements.

[200] "Texas Instruments," [Online]. Available: https://www.ti.com/tool/LM74610-DQEVM.

[201] "Digikey," [Online]. Available: https://www.digikey.com/en/products/detail/diodes-incorporated/DMN6075S-7/5149292.

[202] "Digikey," [Online]. Available: https://www.digikey.com/en/products/detail/texas-instruments/CSD18510Q5B/7056149.

[203] "Digikey," [Online]. Available: https://www.digikey.com/en/products/detail/vishay-siliconix/SQJ422EP-T1-GE3/4496421?s=N4IgTCBcDaIMoEUBSAWMYCiAFAtAFQEYcBxDAZhAF0BfIA.

[204] "Texas Instruments," 06 2016. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2679.pdf.

[205] "Adafruit," Adafruit, [Online]. Available: https://www.adafruit.com/product/2130.

[206] "Adafruit," Adafruit, [Online]. Available: https://www.adafruit.com/product/367.

[207] "Samesky," Samesky, [Online]. Available: https://www.sameskydevices.com/product/switches/tactile-switches/ts02-66-70-bk-160-lcr-d.

[208] "Elecrow," Arduino, [Online]. Available: https://www.elecrow.com/download/Starter%20Kit%20for%20Arduino%28user%20manual%29.pdf.

[209] "Pololu," [Online]. Available: https://www.tme.eu/Document/25459777e672c305e474897eef284f74/POLOLU-2128.pdf.

[210] "lastminuteengineers," [Online]. Available: https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/.

[211] "Texas Instruments," [Online]. Available: https://www.ti.com/product/DRV8825.

[212] "Facfox," 31 01 2023. [Online]. Available: https://facfox.com/docs/kb/the-8-best-stepper-motor-drivers-for-3d-printers.

[213] Peter, "3dwork," 09 05 2020. [Online]. Available: https://3dwork.io/en/tmc-drivers/.

[214] "Pololu," [Online]. Available: https://www.pololu.com/product/2133.

[215] P. D. Reynolds, "Xecor," 18 02 2022. [Online]. Available: https://www.xecor.com/blog/tmc2209-pinout-datasheet-alternatives-uses.

[216] "Analog Devices," [Online]. Available: https://www.analog.com/en/products/tmc2209.html.

[217] "ISO," 09 2017. [Online]. Available: https://www.iso.org/standard/68301.html.

Appendix B – Copyright Permission

We did not have anything for the copyright permission since all information is public and open source.

Appendix C – Data Sheet

SML-P11MTT86(R) - LED | ROHM

PJ-102A - Power Jack | Same Sky

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

OV5640 - Camera | Omnivision

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

ESP32-WROOM-32

ESP32-S3 Series

LM386- Amplifier

Programmer-CP2102

Voltage Regulator-LM2576-ADJ

Motor Driver-DRV8825

Appendix D – Software Code

Button Demo Code

```
#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "esp_log.h"
```

```
#define GPIO_BUTTON     GPIO_NUM_13   // Button pin

static const char *TAG = "BUTTON_TEST";

void app_main(void)
{
   esp_log_level_set(TAG, ESP_LOG_INFO);

   // Configure button pin as input with internal pull-up
   gpio_config_t io_conf = {
      .pin_bit_mask = (1ULL << GPIO_BUTTON),
      .mode = GPIO_MODE_INPUT,
      .pull_up_en = GPIO_PULLUP_ENABLE,
      .pull_down_en = GPIO_PULLDOWN_DISABLE,
      .intr_type = GPIO_INTR_DISABLE
   };
   gpio_config(&io_conf);

   ESP_LOGI(TAG, "Button test started.");

   int last_button_state = 1;      // 1 = not pressed (because pull-up)

   while (1) {
      int raw_state = gpio_get_level(GPIO_BUTTON);

      // Simple (very basic) debounce: sample every 20 ms
      vTaskDelay(pdMS_TO_TICKS(20));
      int confirm_state = gpio_get_level(GPIO_BUTTON);
```

```
        if (raw_state == confirm_state) {
            int button_state = raw_state; // stable state

            // Detect falling edge: HIGH -> LOW (button pressed)
            if (last_button_state == 1 && button_state == 0) {
                ESP_LOGI(TAG, "Button PRESSED");
            }

            // Detect rising edge: LOW -> HIGH (button released)
            if (last_button_state == 0 && button_state == 1) {
                ESP_LOGI(TAG, "Button RELEASED");
            }

            last_button_state = button_state;
        }

        // Slow down loop just a bit more
        vTaskDelay(pdMS_TO_TICKS(30));
    }
}
```

Ultrasonic Sensor Demo Code

```
#include <stdio.h>
#include "driver/gpio.h"
#include "esp_timer.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#define TRIG_PIN GPIO_NUM_5
#define ECHO_PIN GPIO_NUM_18
```

```
#define SOUND_SPEED_CM_PER_US 0.034f

void app_main(void)
{
    // GPIO setup
    gpio_reset_pin(TRIG_PIN);
    gpio_set_direction(TRIG_PIN, GPIO_MODE_OUTPUT);
    gpio_set_level(TRIG_PIN, 0);

    gpio_reset_pin(ECHO_PIN);
    gpio_set_direction(ECHO_PIN, GPIO_MODE_INPUT);

    printf("Ultrasonic Distance Test Started...\n");

    while (1)
    {
        // send trigger pulse
        gpio_set_level(TRIG_PIN, 0);
        esp_rom_delay_us(2);

        gpio_set_level(TRIG_PIN, 1);
        esp_rom_delay_us(10);
        gpio_set_level(TRIG_PIN, 0);

        // wait for echo
        int64_t timeout = esp_timer_get_time() + 30000; // 30ms timeout
        while (gpio_get_level(ECHO_PIN) == 0)
        {
            if (esp_timer_get_time() > timeout)
```

```
            {
                printf("No echo\n");
                goto delay_loop;
            }
        }

        int64_t start = esp_timer_get_time();

        // wait for echo end
        while (gpio_get_level(ECHO_PIN) == 1)
        {
            if (esp_timer_get_time() > timeout)
            {
                printf("Echo pulse timeout\n");
                goto delay_loop;
            }
        }

        int64_t end = esp_timer_get_time();

        // calculate distance
        float duration_us = (float)(end - start);
        float distance_cm = (duration_us * SOUND_SPEED_CM_PER_US) / 2.0f;

        printf("Distance: %.2f cm\n", distance_cm);

delay_loop:
        vTaskDelay(pdMS_TO_TICKS(250)); // take reading every 250ms
    }
}
```

ToF Demo Code

```
/*
 * Simple ToF TMF8701 example using old I2C driver
 *
 * !! Calibration data: 0x0F, 0x3F, 0x8A, 0x01, 0x00, 0x00, 0x00, 0x00, 0x41, 0x57, 0x01, 0x00, 0x00, 0x00
 */


#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "driver/i2c.h"          // OLD I2C DRIVER (not i2c_master.h!)
#include "tmf8701_c_api.h"


static const char *TAG = "example";


#define I2C_MASTER_SCL_IO           CONFIG_I2C_MASTER_SCL       /*!< GPIO number used for I2C master clock */
#define I2C_MASTER_SDA_IO           CONFIG_I2C_MASTER_SDA       /*!< GPIO number used for I2C master data  */
#define I2C_MASTER_NUM              I2C_NUM_0                   /*!< I2C port number for master dev */
#define I2C_MASTER_FREQ_HZ          CONFIG_I2C_MASTER_FREQUENCY /*!< I2C master clock frequency */
#define I2C_MASTER_TX_BUF_DISABLE   0                           /*!< I2C master doesn't need buffer */
#define I2C_MASTER_RX_BUF_DISABLE   0                           /*!< I2C master doesn't need buffer */
```

```
#define TMF8701_SENSOR_ADDR         0x41        /*!< Address of the TOF sensor */

static void i2c_master_init(void)
{
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };
    ESP_ERROR_CHECK(i2c_param_config(I2C_MASTER_NUM, &conf));
    ESP_ERROR_CHECK(i2c_driver_install(I2C_MASTER_NUM,
                                       conf.mode,
                                       I2C_MASTER_RX_BUF_DISABLE,
                                       I2C_MASTER_TX_BUF_DISABLE,
                                       0));
}

void app_main(void)
{
    // 1) Init old I2C driver on I2C_NUM_0
    i2c_master_init();
    ESP_LOGI(TAG, "I2C (old driver) initialized successfully");

    // 2) Create ToF handle
    // EN pin = GPIO_NUM_NC (not used), INT pin = GPIO_NUM_NC (not used),
    // I2C port = I2C_NUM_0, address = 0x41
    DFRobot_TMF8701_C *tof =
```

```
        tmf8701_create(GPIO_NUM_NC,  GPIO_NUM_NC,  I2C_MASTER_NUM,
TMF8701_SENSOR_ADDR);

  while (tof == NULL) {
    ESP_LOGE(TAG, "ToF instance is NULL");
    vTaskDelay(pdMS_TO_TICKS(1000));
  }

  // 3) Initialize sensor
  while (tmf8701_begin(tof) != 0) {
    ESP_LOGE(TAG, "ToF initialization failed");
    vTaskDelay(pdMS_TO_TICKS(1000));
  }

  // 4) Start measurement using that calibration
  if (!tmf8701_start_measurement_ex(
        tof,
        TMF8701_CALIB_MODE_NO_CALIB,
        TMF8701_DISTANCE_MODE_COMBINE)) {
    ESP_LOGE(TAG, "ToF start_measurement_ex failed");
    return;
  }

  ESP_LOGI(TAG, "ToF measurement started");

  vTaskDelay(pdMS_TO_TICKS(500));  // 0.5s – you can increase if needed

  // 5) Main loop: poll distance
  while (1) {
    if (tmf8701_is_data_ready(tof)) {
```

```
            uint16_t range = tmf8701_get_distance_mm(tof);
            ESP_LOGI(TAG, "Distance = %u mm", range);   // use %u, not %X
        }
        vTaskDelay(pdMS_TO_TICKS(10));
    }

    // Not really reached, but for completeness:
    tmf8701_destroy(tof);
    ESP_ERROR_CHECK(i2c_driver_delete(I2C_MASTER_NUM));
    ESP_LOGI(TAG, "I2C de-initialized successfully");
}
```

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."