PUMPED - The Smart Portable System for Weighing, Labeling, and Tracking Breast Milk Bags

Eric Hernandez, Jimmie Rawls, Kevin Fernandez, Augustine Rodriguez

University of Central Florida, Department of Computer and Electrical Engineering, Orlando, Florida 32816-2450, U.S.A.

***Abstract* — A sponsored project, PUMPED - The Smart Portable System for Weighing, Labeling, and Tracking Breast Milk Bags, is a senior design project that contains many aspects of CECS points, interfacing both Hardware and software designs together to form a cohesive, well-developed platform for mothers to ease with labeling breast milk bags. It features a load cell for accurate measurement, a tactile interface using physical buttons, and a thermal printer for generating unique alphanumeric labels. The system communicates with a mobile application via Bluetooth, enabling real-time data storage and retrieval through a cloud-based database linked to user email accounts. Device components: including the microcontroller, OLED display, and printer, interface via UART and I2C protocols, forming a cohesive embedded platform for efficient milk storage management.**

***Index Terms* — Embedded Systems, Mobile Applications, Internet of Things, Cloud Computing, Data Synchronization, Wireless Communication, User Interface.**

I. INTRODUCTION

Managing and labeling breast milk is a regular task for nursing mothers, yet it can be tedious and prone to errors. To address this, PUMPED – the Smart Portable System for Weighing, Labeling, and Tracking Breast Milk Bags was developed as a collaborative senior design project by Electrical and Computer Engineering students. The goal was to create a compact, user-friendly device that streamlines the labeling process and enhances milk storage accuracy.

At the heart of the system is the ESP32-WROOM microcontroller, chosen for its processing speed, Bluetooth capabilities, and peripheral support. It integrates key components: a high-precision load cell for weight measurement, an OLED display via SPI for user feedback, and a TTL-based thermal printer that generates labels with the weight, date, time, and unique ID. All components communicate efficiently over UART, I2C, and SPI protocols. The device syncs with a custom Android application developed in Android Studio and backed by Firebase. The app enables Bluetooth pairing, email-based user authentication, and secure, cloud-based data storage and retrieval.

Enclosed in a custom-designed 3D-printed shell, the system is optimized for portability and everyday home use. PUMPED offers a practical and reliable solution for breastfeeding mothers, eliminating the hassle of manual labeling while ensuring better tracking and organization.

II. SYSTEM COMPONENTS

The PUMPED device houses several components that work together to keep the device functioning within its expected specifications. This section will briefly detail each component and its functions.

A. Microcontroller

The core component of the PUMPED device is its microcontroller. The device utilizes an ESP32S-WROOM. This version of the ESP32 was chosen because of its Bluetooth capabilities, communication protocols, as well as its ability to interface with all of the other peripherals and sensors within the device. The ESP32 runs at clock speeds up to 240 MHz and operates at 3.3 volts DC. It also provides access to multiple GPIO pins, each with different functionalities. This MCU is compatible with the Arduino IDE, allowing for ease of programming and debugging.

B. Thermal Printer

The PUMPED device incorporates a compact thermal printer to produce adhesive labels for breast milk storage bags. The MC206H embedded thermal printer operates off of 5 volts DC at a max current of 2 amps. This printer connects to the ESP32 via a UART Serial connection for data integrity and reliability. The printer uses thermal paper, so no ink is required, making it a suitable component for low-maintenance operation.

C. Battery

To ensure portability and reliable operation for the PUMPED device, the device is powered by a USB rechargeable power bank. The ROMOSS Sense 8+ has a 111 Wh capacity with two USB-A output ports. The output port rated for 5 volts with a max current of 3 amps will power the printer, while the other output port rated at 5 volts with a max current of 2 amps will power all of the other systems. The power bank's capacity allows for several hours of continuous operation, making it useful for

daily use or travel. To recharge the PUMPED device, the power bank has a USB-C input port that can be used for charging.

D. Voltage Regulator

Since the PUMPED device utilizes components that run at different voltages, voltage regulation becomes necessary to ensure reliability and device longevity. The LM2576 is a buck switching regulator that can take up to 40 volts of input voltage and step it down to 3.3 volts. The voltage step down is necessary for the MCU, Load Cell, and OLED display. The regulator ensures that all components receive a steady supply voltage, even when the battery voltage fluctuates.

E. Load Cell

A strain gauge load cell is used to measure the weight of the breast milk bags placed on the device. The load cell is capable of accurately measuring weights up to 10 kg (22 lbs), which is more than sufficient for any breast milk weighing scenarios. The load cell converts mechanical forces into an analog voltage signal, which can then be converted by an ADC.

F. ADC

To process the signal created by the load cell, the system uses an HX711 24-bit ADC amplifier. This device is specifically designed for a variety of different weighing applications, utilizing a load cell. It amplifies the signal received from the load cell and converts it into a digital value that the ESP32 can read. The HX711's 24-bit resolution allows the device to detect weight changes as small as a tenth of a gram.

G. Battery Indicator

Due to the power bank being entirely enclosed within the 3D-printed shell, its internal battery indicator will not be visible. To combat this, the system will utilize 4 photoresistors in conjunction with the ESP32's built-in ADC to provide a constant indication of the power bank's actual battery level on the OLED display.

$$V_{OUT} = V_{IN} \bullet \frac{R_1}{R_1+R_2} \qquad (1)$$

In order for us to successfully get the battery indicator to work, we had to improvise on how that would occur. We figured out that the only thing we required was to create a voltage divider between the photoresistor and another resistor in series. We chose to use 10k Ohm resistors. There are a total of four LEDs so we had to come up with a pcb design that would house four voltage dividers for each LED. It became an easy task to implement but came with innovation.

H. OLED Display

To display information to users, the PUMPED device contains a HiLetgo SSD1309 128x64 pixel OLED display. This display provides real-time visual feedback to the user, including weight readings, unit mode, system status, and battery level. The display uses the SPI communication protocol to communicate with the ESP32. This allows for fast and reliable data transfer. The display also has high contrast, which allows for easy readability in various lighting conditions.

I. Buttons

Mounted on the front of the PUMPED device are four buttons.

Fig. 1. Button Configuration

Button 1 will allow the user to tare the scale on button press or reset the scale on button hold. Button 2 will allow the user to change the unit in which the device displays the weight, either grams or ounces. Button 3 will serve as the offline print button. Button 4 functions as the online print button, sending and receiving data while also thermal printing a label.

Ⅲ. SYSTEM FUNCTION

The function of the PUMPED device is designed to provide caregivers with an efficient and accurate way of weighing a bag of breastmilk and producing a label with all of the required information printed onto it, while also recording that data into an app for record purposes. This section serves to describe system operation, beginning with power on and ending with data sent to the user.

A. Power-On and Initialization

When the user powers on the PUMPED device via the switch, the battery bank will supply power to the MCU,

which in turn will initialize all of the other peripherals being powered. The user will see the OLED display displaying scale weight values as well as the power bank's battery level. Bluetooth functionality will be initiated by the ESP32, and the connection process between the device and the app will be completed.

B. Tare/Reset Load Cell Operation

Once the initialization step is complete, real-time sampling of the weight begins with the HX711 performing a 24-bit ADC conversion of the voltage signal generated by the load cell. The load cell utilizes strain gauges in a Wheatstone bridge configuration, pictured below, to quantify different weights applied to the load cell. The load cell is calibrated by booting the MCU with a calibration program that will produce a calibration factor that can be used during normal operation. With the calibration factor known, the ESP32 converts the amplified signal into a value in grams. The software to display the weight value contains delays, so that the displayed value will not change as rapidly, allowing the user to read a stable value.

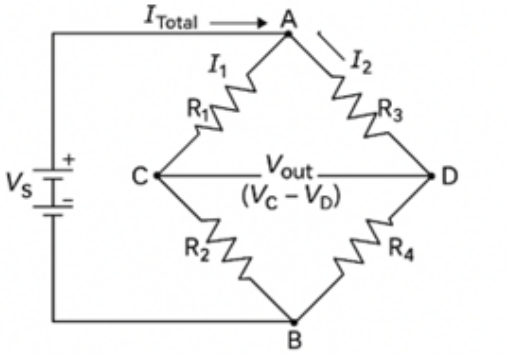

Fig. 2. Wheatstone Bridge Configuration

C. Measurement Display and Unit Conversion

Measured weight data is provided by the OLED display, and unit conversion between grams and fluid ounces can be facilitated by pressing one of the available four buttons on the front of the device. Measured weight values are formatted to display two decimal places.

$$Fluid\ Ounce = Grams \bullet 0.035274 \frac{Ounces}{Grams} \quad (2)$$

Equation 2 illustrates the unit conversion from grams to ounces. We obtain the value from the load cell in grams and then we multiply it by the conversion factor where 1 gram ≈ 0.035274 fluid ounces for water based liquids like breast milk.

$$Grams = Fluid\ Ounces \bullet 28.3495 \frac{Grams}{Ounces} \quad (3)$$

Equation 3 illustrates the unit conversion to grams. Once again we obtain the value from the load cell and then we multiply it by the conversion factor where 1 fluid ounce (water based) ≈ 28.3495 grams.

D. Label Formatting and Printing

Once the user is ready to print, and other information like date/time, and allergen information is available, the print button can be pressed to lock in the weight that is on the scale, and the data is transmitted to the thermal printer via UART at a baud rate of 9600. The printer will then generate a label using onboard heating elements and thermal paper, producing a durable, smudge-resistant tag that is suitable for cold storage.

E. BLE Communication

When the PUMPED device is first turned on, the ESP32 activates the Bluetooth module and makes a connection with the PUMPED app. The app will provide important information like time/date, allergen information, and label format. Once the print button on the device is pressed, weight data for what is currently on the scale will be sent to the app at the same time. The status of the label just printed will be in the app, along with all the previous milk bag information and status.

```
// BLE SETUP
 BLEDevice::init("Pumped Printer Integrated");
 pServer = BLEDevice::createServer();
 pServer->setCallbacks(new MyServerCallbacks());

 BLEService *pService =
pServer->createService(SERVICE_UUID);
 pCharacteristic =
pService->createCharacteristic(CHARACTERISTIC_UUID,
                    BLECharacteristic::PROPERTY_READ |

BLECharacteristic::PROPERTY_NOTIFY);
```

Fig. 3. Bluetooth Setup Code

The BLE code Initializes the ESP32 as BLE server, assigns connection callbacks, and defines a custom BLe service identified by SERVICE_UUID. It adds characteristics that support both read and notify properties to connect with our mobile app.

F. Hardware flow diagram

The hardware flow diagram pictured below depicts the operational flow of the hardware in the PUMPED device.

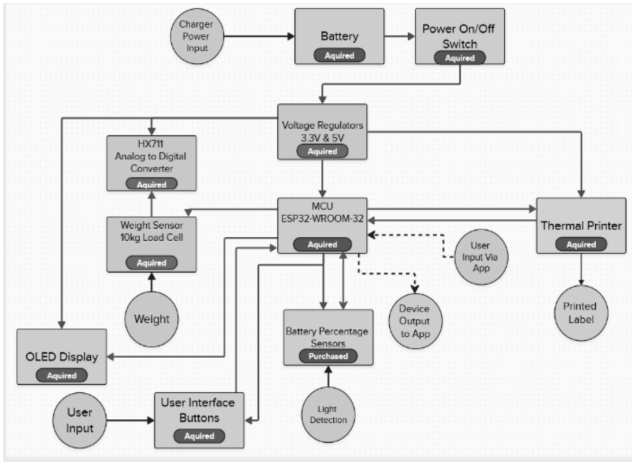

Fig. 4. Hardware Flow Diagram

IV. SOFTWARE DETAIL

The software for this project was encapsulated by three main areas: the C++ code for our ESP32 microcontroller, the Kotlin code for our Android mobile application, and our remote real time database hosted by Google's Firebase backend as a service (BaaS) platform.

To show a clear description on how the embedded system will work for our device, the process is shown down below,

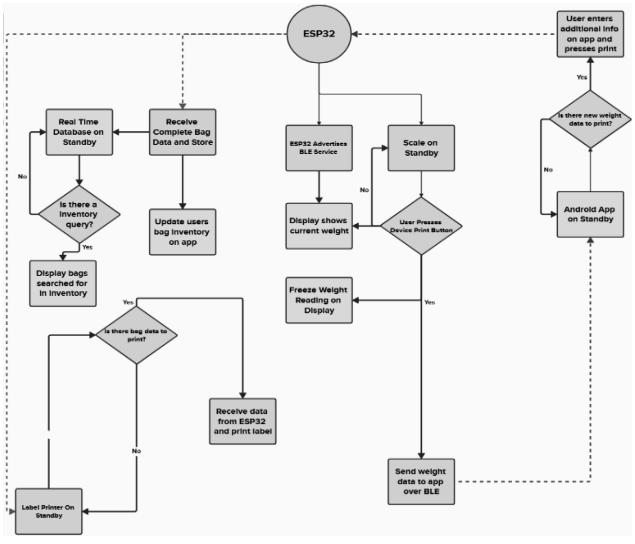

Fig. 5. Device Software Flow Diagram

This process outlines the operation of the system. The ESP32-WROOM communicates with firebase, the load cell, and our mobile app via bluetooth BLE. Upon user query, inventory data is retrieved and updated. When a user presses the device's print button, the current weight is frozen and transmitted to the mobile app within seconds. The user may then confirm and print the label through the app. The printing of labels will also have the ability to be used independently without the confirmation of the mobile app, if urgency is needed.

A. ESP32 Software

Our ESP32 microcontroller was programmed in C++ using the Arduino IDE. The ESP32 microcontroller's code managed the connection and readings from the various sensors that it was connected to. It made any necessary conversions from analog to digital for its readings and additionally any weight conversions needed to properly present the bag data to users. Additionally our ESP32 microcontroller was programmed to manage the Bluetooth low energy connection with the Android mobile application. This was done by the ESP32 being programmed to act as a Bluetooth low energy server. It advertised its connection by advertising a specific "service" that is represented by a universally unique identifier. This service itself contains "characteristics" which represent in this case data from specific sensors. The primary characteristic we were concerned with for this project was the "weight characteristic," the reading from our load cell that would represent the weight of the bag placed upon it. Once the Android mobile application scanned for the service's universally unique identifier it could form a connection. Upon successfully forming a connection with our ESP32 Bluetooth low energy server it would receive the data almost immediately. Ultimately this represents the general overview of how our ESP32 software functioned on a technical level.

B. Android mobile application software

The image below is a quick and concise description of our mobile app, it illustrates the process from beginning to end.

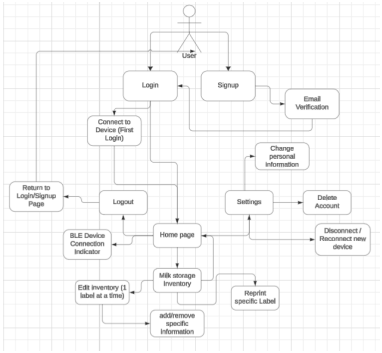

Fig. 6. Mobile App Software Flow Diagram

When a user first enters the app they will be prompted to login or signup with their email and password. First

time users have to create an account and verify their email for security reasons. Once logged in, the user will arrive at the connect device screen, where they can connect their new device. If it's not their first time logging in, then they will be directed to the home screen, where an encouraging message will appear, it will show today's pumps, last pump, connection between devices, and four tabs: Home, Inventory, Statistics, and Settings. Each tab has a unique role, and will serve the user a great assistance.

Our Android mobile application was programmed with Kotlin in Android Studio. The Android mobile application's code was responsible for the user interface system of the application, the business logic that defined how the users would interact with the app's data, the connection to the real time database hosted by Google's Firebase backend as a service, and the Bluetooth low energy connection with the ESP32 microcontroller. The user interface was built using Jetpack Compose which is a UI toolkit that uses the Kotlin programming language. The business logic for the Android mobile application was written in Kotlin as well and enabled the user's actions such as: deleting bags, changing account information, and more to be carried out accordingly and represented in the mobile application's real time database.

For the aforementioned user interface and business logic code a repository design pattern was utilized in order to achieve a separation of concerns and guarantee a high degree of efficiency for mobile application testing. Furthermore the Android mobile application managed its connection to Firebase's real time database through additional Kotlin code that relied on the use of a secure API key and additional dependencies that verified a secure connection between the Android mobile application and its database. Lastly the Android mobile application was programmed with Kotlin code to manage the application's connection to the ESP32 microcontroller which was acting as a BLE server. Our mobile application in response was acting as a BLE client that scanned for any available connections. The mobile application scanned for the appropriate connection and not enough erroneously by searching for the specific universally unique identifier that our BLE server (the ESP32) was advertising as previously mentioned in the last section.

Once our mobile application identified the correct connection via its scanning capabilities it automatically formed the connection with the ESP32 BLE server. Once that connection was successfully formed data could be sent to and from the ESP32 BLE server via the Android mobile application. This represents the overall complete functionality of our Android mobile application and its responsibility as an intermediary between the ESP32 BLE Server and the real-time database.

C. Real-time database

Our real time database for this project was hosted by Google's Firebase which is a backend as a service. The real time database is a NoSQL database and was selected for this project as opposed to a more traditional relational database due to the specific needs of our project. For this project we needed data to be transferred in real time, instantaneously and at any time at all. Additionally we wanted the scalability of enabling a high frequency of data transmissions and a NoSQL database presents upside in that department compared to a more traditional relational database. In order to achieve a high efficiency with our database calls within our Android mobile application we "flattened the structure" of our database and ensured that calls were made in such a manner that they did not need to go down deeply nested paths for information. Within our Android mobile application we wrote Kotlin code that facilitated the connection between the real time database and the mobile application.

An auxiliary benefit of using Google's Firebase backend as a service is that we also had access to Firebase authentication. While this is not strictly a part of the real time database it is directly adjacent and operates with it to manage the user's queries that ultimately interact with the real time database. The Firebase authentication in simple terms enables us to automatically register users with a unique identifier that is known to the Android mobile application and the real time database anytime after registration and that is what's used to facilitate user's queries to the real time database. Overall this is a look at the functionality of our real time database and how it interacts with our aforementioned real time database.

The overall software system that makes up this project was reliant on these three aspects working together through either Bluetooth low energy (BLE) and a Wi-Fi or cellular internet connection. The BLE connection was necessary for the ESP32 microcontroller and the Android mobile application to communicate with each other, and the Wi-Fi or cellular internet connection was needed for the Android mobile application to communicate with our real time database.

V. 3-D Enclosure Design

The PUMPED device features a fully custom 3D-printed enclosure that was designed for ease of assembly and optimal space usage. The design process involved precise measurement and iterative testing to ensure proper fitment

of all components such as the OLED screen, thermal printer, buttons, load cell, and power on/off switch.

All of the openings on the enclosure were dimensioned using calipers to match as closely as possible to the components sizes while also taking into account the tolerance of our 3D printer. To minimize print time and material consumption, all openings were first tested through segment prints prior to finalizing the final enclosure 3D model. This ensured that our final print would be properly aligned and sized holes for our parts to be mounted to and reduced plastic filament waste.

An important part of our design was the way that we integrated the load cell. The bottom of the load cell is seated on a platform that is raised 3 mm from the load cell enclosure's bottom. This spacing provides more than enough clearance for the laid cell to bend and measure the weight applied to it. It is held by 2 M5 screws that are counterbored and screwed from the bottom of the load cells enclosure. The load cell was then sealed with a top lid that has a hole in the center, this is where we have a custom extension that allows us to screw the top weight plate onto the load cell and keeps the sensing end of the load cell completely isolated from touching anything else.

Our weight plate was designed with a flat elliptical section in the center wide enough to stand a breast milk bag on, and is then sloped towards the outer edges of the plate. This allows users to place the bag standing upright, or layed flat on the weight plate. The plate is screwed onto the load cell with 2 M4 screws and includes 2 rubber O-rings underneath the screw heads. The O-rings provide an extra layer of protection for the device. Creating a watertight seal around the screws so that no liquid pours down into the load cell or the electronics inside the enclosure.

The enclosure also includes a port for USB-C charging and a rear hatch, held in place by embedded magnets. This hatch can be removed to expose a USB-A access port so that we can program the board without having to take it apart.

Threaded brass inserts and M3 screws were used to hold all 3D printed parts together which allows us to repeatedly disassemble the enclosure for troubleshooting without degrading the threads.

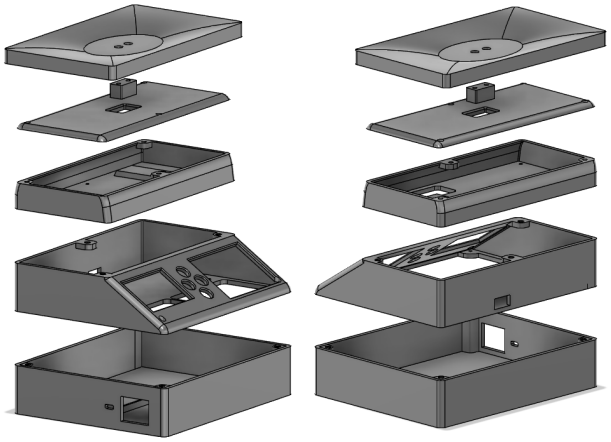

Fig. 7. Exploded Enclosure 3D Model (Front & Back)

Together, these design choices contribute to our idea of making the device look as complete and compact as possible for a prototype.

VI. TESTING

One of the key main features that we focused on was having an accurate measurement Load Cell that was capable of accurately weighing with a marginal error of +- 3 grams. After multiple tests we concluded that we had succeeded with our design.

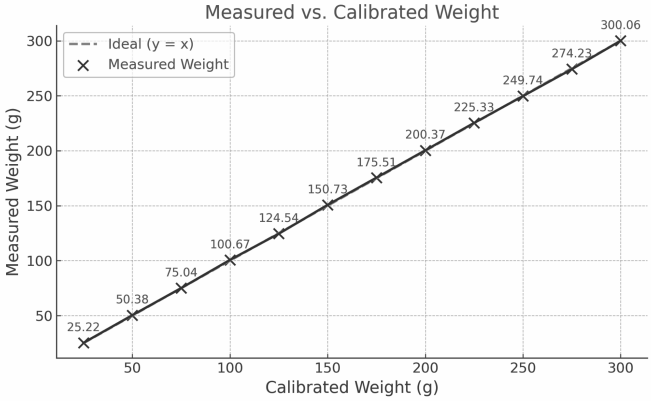

Table. 1. Measured Weight vs. Calibrated Weight

The table illustrates all of our test trials that we ran with one another. We measured the calibrated weight that we purchased online with different variations of values, we then tested them with a commercial grade scale to make sure those values were indeed correct. After doing so, we began testing our devices' weighing capabilities and to our surprise, those values were indeed very close to what the actual weight should be. A small margin of error was observed. With this information, we will continue to do more test trials in order to minimize the margin of error more.

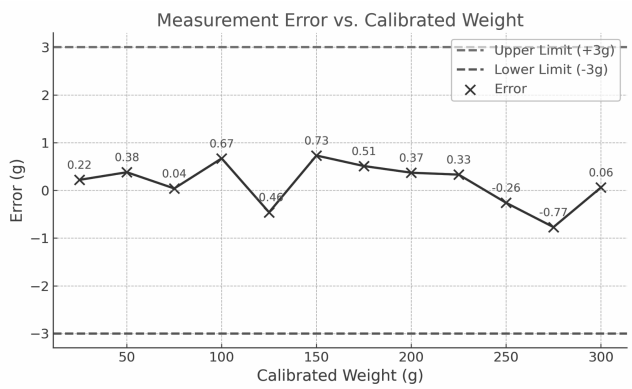

Table. 2. Sensor Error vs. Calibrated Weight

Table 2 shows the comparison between the error of our device compared to the actual values of the items being measured. There is a very slight error that does not exceed half a gram. Making our device very much accurate.

VII. DESIGN CHALLENGES & LIMITATIONS

Throughout the development of the PUMPED device, we ran into several engineering challenges. These obstacles ended up shaping our design based on the decisions or solutions we found for both hardware and software issues.

A. *Limited Printer Documentation*

One of the most significant challenges we faced was working with the embedded thermal printers sourced from Chinese Manufacturers. These printers lacked official datasheets and any significant technical documentation which made the integration to our system more difficult than expected. The first printer we received was advertised to operate between 5-9V, but after testing, we could only get to work at a minimum of 17V, which was beyond what we planned on using with our power supply and voltage regulators. After this, a second printer was ordered, and functioned properly at 5V which matched with our original system design.

However, even after we fixed the voltage issue, our printer was now outputting symbols and gibberish. After extensive debugging, we traced the issue to an undocumented switch on the back of the printer. This info was discovered via online forums and switched the printer from RS232 to TTL connection. After switching it to TTL mode, and flipping our RX and TX lines, we finally got the printer to consistently output ASCII based text.

B. *Battery Power Monitoring*

Since we decided to use a sealed ROMOSS battery bank we had limited access to the internal circuitry. This meant that we could not directly measure the battery percentage via any circuits. With no electrical interface to determine the status of our battery, our only indicators were the 4 LEDs on the top of the battery bank that illuminate to show the battery levels. To address this issue, we decided to use 4 photoresistors that are aligned with the LEDs. This allowed us to send an analog signal to our ESP32 that could allow us to read and display battery levels in 25% intervals.

C. *Label Alignment*

The initial use of pre-cut label rolls led to inconsistent text alignment on the labels. Since our printer does not have a positioning sensor or any way to monitor the rolls position, some labels printed across the cut lines, or were visibly misaligned vertically. To improve reliability, we decided to switch from rolls of pre-cut labels to rolls with a continuous label roll. This change removed any mechanical alignment dependencies since the user label can be ripped at any point throughout the roll.

D. *Enclosure Size Constraints*

The thermal printer and the battery bank are the largest components in the device, and they dictated the overall enclosure dimensions. Our goal was to accommodate these 2 components along with the other peripherals and wiring in as little space as possible. Despite these constraints, our team managed to keep it to a compact size of 8 × 6.87 × 5.32 inches (L×W×H).

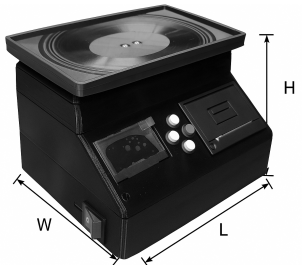

Fig. 8. PUMPED Device Enclosure

E. *Future Design Improvements*

We have found several opportunities for further optimizing the PUMPED device in future iterations. One significant enhancement involves changing the power supply. Our current design uses a commercial USB battery bank that was chosen for safety and time constraints by the accelerated Summer term. While This approach provided us with a convenient and safe solution to power the device, it introduced size and battery monitoring limitations. Future versions could adopt the use of a custom made Lithium-Ion battery pack using 18650 cells.

This would significantly lower the enclosures footprint, improve power distribution to the thermal printer, and allow for more accurate integrated battery monitoring through the use of fuel gauge ICs. Although, the battery system will still be limited by the high current demands of the thermal printer. They must be powerful enough to supply a little over 2 amps of current to ensure all systems can function simultaneously without damaging the batteries.

The weighing sensor also offers some room for improvement. While the 10 kg strain gauge load cell paired with the HX711 amplifier provided us with an accurate and effective measurement, it adds to the vertical size of the device. In future designs, the use of more compact weight sensors, like the ones commonly found in traditional kitchen scales could reduce the device's height while still providing accurate measurements.

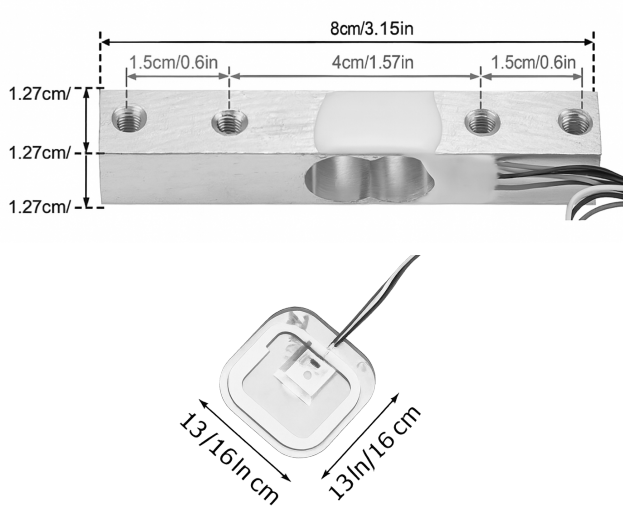

Fig. 9. Load Cell VS Scale Strain Gauge Sizes

The printer itself also offers some room for improvement. While the embedded printer we purchased includes its enclosure, servo motor, and control board, the actual thermal print head is very compact. A custom-designed printer using just the thermal element, a small stepper motor, and custom thermal paper roll storage could allow the final size of the device to be more compact. However, even with these optimizations, the device's size will still be limited by the diameter and length of the thermal label rolls.

Ⅷ. CONCLUSION

PUMPED delivers a practical, compact solution for weighing, labeling, and tracking breast milk bags. By integrating precise hardware with Bluetooth connectivity and cloud storage, the system enhances ease and accuracy for nursing mothers. Despite development challenges, the final prototype meets its core objectives and provides a strong foundation for future refinement and expansion.

THE ENGINEERS

Eric Hernandez is a 24-year-old Computer Engineering student who has accepted a position with Florida Palm in Fort Lauderdale, FL, as an Assistant Project Manager. His work will focus on large-scale infrastructure projects. Eric's long-term goal is to become a Licensed Professional Engineer (PE) in the state of Florida.

Jimmie Rawls is a 29-year-old Electrical Engineering student who is projected to earn a *cum laude* distinction upon graduation and is currently looking for employment in an electrical engineering position.

Kevin Fernandez is a 23 year-old Electrical Engineering student who has accepted a position with Merrick & Company in Aiken, SC. He spent over a year interning/working part time in their Nuclear Services and Technology (NST) sector, supporting glovebox design with a focus on instrumentation and controls (I&C). Kevin looks forward to continuing his work after graduation and plans to pursue his Professional Engineer (PE) License.

Agustin Rodriguez is a 23 year-old Computer Engineering student pursuing a career in software engineering. Agustin has previous experience working as a software engineering intern at NASA's Kennedy Space Center.

ACKNOWLEDGEMENT

The authors wish to acknowledge and thank the extensive assistance, support, and permission permitted to answer many of our questions from Dr. Lei Wei, Dr. Weeks, and Dr. Chan, as well as many of our friends and family.

REFERENCES

[1] " Build your first app with Jetpack Compose," *Android Developers*, Google. Retrieved July 7th, 2025 https://developer.android.com/codelabs/basic-android-kotlin-compose-first-app