AirSense: A Wearable Air Quality Monitoring Device

Jiayi Shao
University of Washington
206-468-6804
jyshao34@uw.edu

Chenwei Huang
University of Washington
206-889-7670
hcw315@uw.edu

Helen Lai
University of Washington
425-628-9425
helenyl@uw.edu

Mark Forsnes
University of Washington
mforsnes@uw.edu

ABSTRACT

The project is an IoT system using wearable devices as backpack clips to collect localized air and environmental data as users move. The edge device includes four key sensors: PM2.5, temperature/humidity/VOC, and an accelerometer. The Raspberry Pi Pico W will retrieve GPS data from the user's phone, helping distinguish accurate location. The accelerometer will monitor movement, and if the device is stationary—such as being left on a surface—this indicates inactivity, triggering the system to enter a low-power sleep mode by reducing the clock frequency and turning off the PM2.5 sensor to conserve battery. This is particularly important as our design is a wearable clip, ensuring portability and efficient power usage. All collected sensor data will be transmitted via BLE to the user's phone and then uploaded to the cloud. The user's phone will also further process the data and display, offering users a comprehensive overview of air quality and environmental conditions in real-time and the forecast.

Keywords

Air quality, Raspberry Pi, sensors, VOC, PM2.5, sleep mode

1. INTRODUCTION

Air quality is a critical environmental metric that has a direct impact on human health and quality of life. Living or spending extended amounts of time in areas with poor air quality can lead to various issues such as poor respiratory function, allergies, and other long-term health problems such as increased risk of respiratory system-related cancers. Traditionally, air quality monitoring stations are stationary and provide limited spatial resolution. This leads to often misleading or inaccurate information as air quality can vary greatly in urban environments. For example, areas by busy roads or freeways are much more likely to have poor air quality as opposed to an area by a large park. Stationary air quality monitoring stations also do not provide any information about air quality inside of buildings.

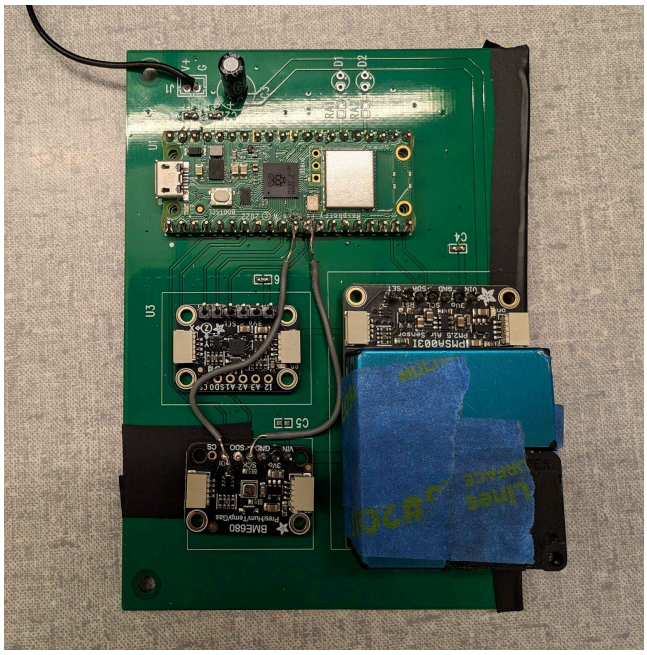

Fig. 1. Picture of Assembled Device

The project is an IoT (Internet of Things) system that uses a distributed network of devices that are designed to be worn by users to take localized air quality measurements and other environmental measurements as users walk around urban environments. These small, integrated monitoring devices will communicate with the user's phone to upload this data to cloud storage.

2. RELATED WORKS

Several existing air quality monitoring systems have inspired aspects of this project. Each demonstrates unique strengths but also highlights limitations that our design aims to address.

1. PurpleAir: PurpleAir provides stationary air quality monitoring devices capable of compiling and visualizing hyper-local air quality data. These systems are widely used in residential and community applications and provide reliable data for localized areas. However, they are fixed in place, making them unsuitable for providing dynamic air quality measurements across varying locations.
2. AirBeam: AirBeam systems are portable air quality monitors designed for personal use. While they offer mobility, they lack intelligent power management features like sensor switching and sleep mode, which are critical for prolonged use in wearable devices. Additionally, AirBeam primarily focuses on collecting particulate matter data, whereas our device expands functionality by integrating VOC, temperature, and humidity sensors, providing a more comprehensive environmental analysis.
3. Plume Labs Flow: Plume Labs Flow is a personal air quality monitoring sensor that communicates with a smartphone app for data visualization. While similar in concept to our design, Flow does not include intelligent power-saving mechanisms like accelerometer-based sleep mode or sensor switching. Furthermore, our project emphasizes not just visualization but also leveraging machine learning for air quality forecasting, enabling users to make proactive decisions based on predictive insights.

3. RESEARCH RELATED WORKS

3.1 Paper 1

This paper [1] is motivated by the need to address the significant health impacts of air pollution, particularly on vulnerable populations such as individuals with respiratory conditions, allergies, or asthma. The authors aim to provide a practical, affordable solution for detecting and mitigating air quality hazards in both indoor and outdoor environments.

The proposed system integrates three key sensors: the MQ-7 gas sensor for carbon monoxide (CO), the DHT-22 for humidity and temperature, and the GP2Y1010AU0F optical dust particle sensor for particulate matter. These sensors collect data on critical air quality parameters, which are processed by an Arduino microcontroller. When pollutant levels exceed predefined thresholds, the system uses a GSM module to send alerts to a specified phone number, giving users immediate feedback.

Additionally, the system interfaces with external devices like air purifiers and humidifiers via relays, enabling automated responses to improve air quality. A 16x2 LCD display provides real-time updates, while the GSM module facilitates remote communication.

Compared to our own system, the motivations are similar, and although different sensors are used, the workflows share common elements such as sensor data collection, processing, and user notification. The key difference lies in its stationary setup, which integrates with external devices, whereas our design prioritizes portability and power efficiency.

Some key takeaways from this paper include the potential for incorporating customizable thresholds for air quality parameters and exploring the possibility of connecting portable systems with external devices.

3.2 Paper 2

This paper [2] is motivated by the growing need to monitor and address air pollution. Airborne particulate matter and harmful gases are major contributors to cardiovascular diseases, respiratory illnesses, and ecological degradation. The authors aim to provide a real-time, IoT-enabled air quality monitoring system to help analyze and mitigate air pollution effectively.

The proposed system utilizes a Raspberry Pi as the central system, interfacing with sensors such as DSM501A (PM2.5), MQ9 (carbon monoxide), MQ135 (carbon dioxide), DHT22 (temperature and humidity), and BMP180 (pressure). Sensor data is processed by an Arduino microcontroller and transmitted via the Raspberry Pi to IBM Bluemix Cloud, enabling remote monitoring and data visualization on a dashboard. Communication between the devices and the cloud is facilitated using the MQTT protocol, ensuring efficient and reliable data transfer.

This IoT-based system emphasizes accessibility and scalability. It integrates low-cost sensors with a Raspberry Pi and provides a user-friendly interface through Node-RED, a visual programming tool. Real-time sensor data is displayed on a dashboard and compared against expected values to assess air quality trends.

Compared to our project, this system focuses on a stationary IoT setup with continuous data transmission to a cloud-based dashboard, while our system emphasizes portability and power efficiency. Both systems utilize a similar workflow involving sensor data collection, processing, and cloud-based user notification. However, while their system relies on Wi-Fi for real-time data transmission, our design schedules less frequent uploads via Bluetooth to conserve power and accommodate the instability of campus Wi-Fi.

3.3 Paper 3

The work by Kolehmainen et al. [3] presents a comparative analysis of neural network approaches for air quality forecasting, specifically examining NO2 concentration predictions. Their research evaluates two distinct neural architectures - Self-Organizing Maps (SOM) and Multi-Layer Perceptrons (MLP) - against traditional periodic component regression methods. The study utilizes a comprehensive dataset spanning 1994-1998 from Stockholm, incorporating NO2 measurements and meteorological parameters.

Of particular relevance to our implementation is their demonstration that direct application of MLP networks to raw sensor data outperforms both periodic regression methods and hybrid approaches combining neural networks with periodic component analysis. Their MLP implementation achieved a root mean square error (RMSE) of 10.72 $\mu g/m^3$, compared to 15.38 $\mu g/m^3$ for periodic regression alone. This validates our choice to

implement direct neural network processing of sensor data rather than pre-processing with periodic decomposition.

Kolehmainen's work also highlights a critical limitation in air quality prediction systems - the difficulty in accurately forecasting extreme pollution events due to their underrepresentation in training data. This insight informed our decision to implement a dual-threshold alert system that maintains sensitivity to both typical variations and extreme events. While their work focused on NO2 specifically, the architectural principles they established transfer well to our multi-pollutant monitoring approach using the BME680 and PMSA003 sensors.

The authors suggest that prediction accuracy could be improved by incorporating atmospheric stability parameters and historical time-series data. While our portable system's design constraints preclude direct measurement of atmospheric stability, we have implemented a rolling window of historical measurements in our forecast model to capture temporal dependencies, partially addressing this recommendation within our power and computational constraints.

3.4 Possible Future Research

1. Advanced Power Management: Investigate the use of deep sleep mode and dynamic voltage scaling to further enhance energy efficiency. Research could also explore the integration of energy harvesting technologies, such as solar energy to prolong battery life.
2. Expanded Sensor Suite: Future iterations could incorporate additional sensors, such as ozone (O_3) or nitrogen dioxide (NO_2) detectors, to provide a more comprehensive view of air quality. These enhancements would make the device more useful in urban and industrial environments.
3. Improved Connectivity Options: Beyond BLE, adding support for other communication protocols, such as LoRa, could enable wider deployment of the device in remote areas or for use in large-scale environmental monitoring networks.
4. Miniaturization and Wearability: Explore the use of advanced manufacturing techniques to create a more compact, and lightweight design. Flexible electronics or printable sensors could further enhance the device's portability and comfort as a wearable accessory.

4. TECHNICAL DETAILS

4.1 Overview

Fig 2 below shows the overall system architecture of our device.

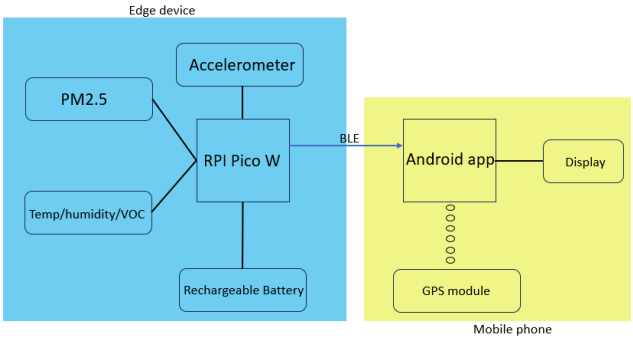

Fig 2. Overall System Block Diagram

4.2 Theory of Operation

In this project, we use the Raspberry Pi Pico W as the central microcontroller, integrating multiple sensors to collect air quality data, including temperature, humidity, PM2.5, and VOC measurements. An accelerometer is employed for power management by detecting movement, enabling the device to conserve battery life through intelligent sleep mode activation.

Once the device collects and processes the data, it transmits all information via BLE to a smartphone. The phone displays real-time air quality metrics, visualizes them on a map with precise coordinates, and simultaneously uploads the data to the cloud. This cloud integration enables advanced forecasting, providing users with both immediate insights and predictive air quality trends

4.3 Hardware Implementation

4.3.1 PCB Design

To make our entire system more compact and to clean up the wiring, we decided to design our own custom PCB.

4.3.1.1 Schematic

The circuit diagram includes the input voltage interface, the Pico W, three sensors along with their peripheral circuits, and two LEDs to indicate different levels of air quality.

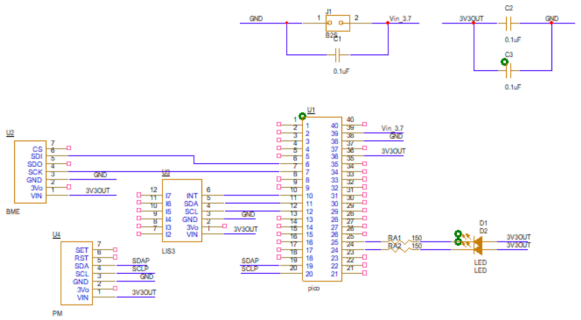

Fig 3. Schematic

4.3.1.2 PCB Layout

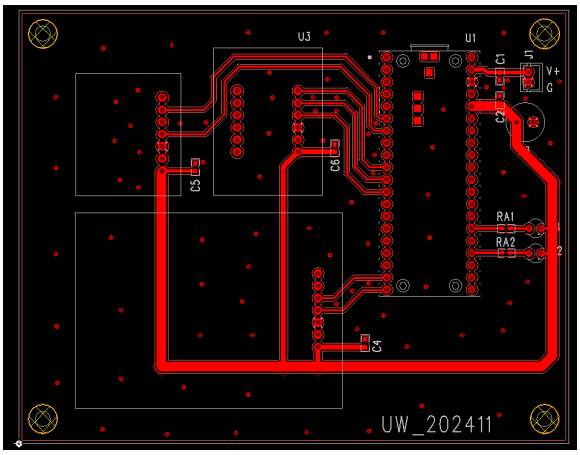

Fig 4. PCB Layout

4.3.2 Enclosure Design

Onshape was used to design the case for our device.

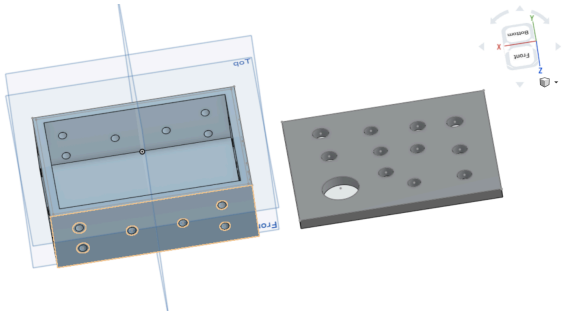

Fig 5. CAD Design of Case

This case was designed with the size of our PCB and size of battery in mind, ensuring there is enough space for them to be laid side by side. The holes seen on the sides of the case are to ensure enough air flow for the sensor measurements. The largest hole on the top face is where the on off switch will go.

Further, to ensure the robustness and durability of our design, calculations for the type of material needed to absorb a certain amount of impact is shown below.

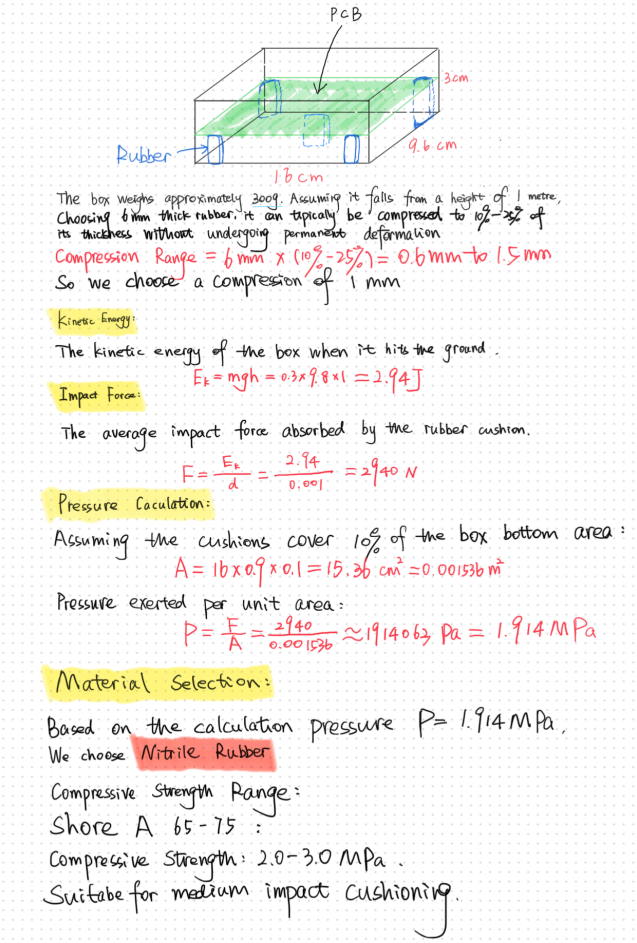

Fig 6. Case Shock Absorption Calculation

4.4 Software Implementation

4.4.1 Embedded Firmware and Sensor Implementations

4.4.1.1 BME680 Implementation

The BME680 sensor implementation on the RP2040 follows a structured approach for environmental data acquisition and processing. The system utilizes I2C communication at 400 kHz, with the BME680 configured as a slave device at address 0x77. The implementation is divided into three core components: initialization, data acquisition, and processing.

The initialization sequence configures sensor parameters through register programming, establishing 16x humidity oversampling, 1x pressure oversampling, and 2x temperature oversampling rates. The implementation sets an IIR filter size of 3 for noise reduction and configures the gas heating element to operate at 320°C with 150 ms heating cycles for VOC measurements.

Data acquisition operates through a state machine approach in the measurement process. The system initiates measurements using

forced mode operation, where the sensor completes a single measurement cycle before entering sleep mode. After a programmed 250 ms warmup period, the system performs sequential register reads to obtain raw sensor data. The implementation processes this data using manufacturer-provided calibration coefficients for converting ADC values to physical measurements, followed by temperature and humidity compensation algorithms.

The VOC concentration calculation converts compensated gas resistance measurements to parts per million, utilizing calibrated baseline resistance values of 55.12 kΩ as reference points. The implementation achieves measurement accuracies of ±1°C for temperature, ±3% for relative humidity, and ±0.12 kPa for pressure, with VOC measurements spanning 0-10 ppm.

4.4.1.2 PMSA003I Implementation

The PMSA003I particulate matter (PM) sensor implementation comprises three primary functional blocks: initialization sequence, measurement configuration, and data acquisition processing. The sensor communicates via I2C protocol at a standard clock speed of 100 kHz, interfacing with the Raspberry Pi Pico W's I2C peripheral. The PMSA003I is configured as a slave device with the default I2C address of 0x12.

The initialization sequence involves setting up the Raspberry Pi Pico W's I2C interface, including configuring the GPIO pins for SDA and SCL, enabling pull-up resistors, and verifying proper communication with the sensor. This ensures stable I2C communication, which is critical for retrieving data accurately.

The measurement process uses a state-based acquisition system. Each measurement cycle starts with the Pico W sending a data read request to the PMSA003I sensor. The sensor responds with a 32-byte data packet containing raw particulate matter concentration values for PM2.5. A delay of 200 ms is introduced between measurements to allow the sensor to stabilize and provide reliable data.

4.4.2 Power Management

To conserve power, we decided to incorporate an accelerometer into our implementation. The rationale behind this is that, as a wearable device designed to monitor air quality on the move, significant environmental changes are less likely to occur when the device is stationary for an extended period of time. This is an opportunity to put the device in a sleep mode, thereby conserving energy without compromising functionality.

The software implementation began with developing the sensor driver code for the LIS3DH accelerometer. The main functionalities needed were sensor initialization and motion detection for both the wake-to-sleep and sleep-to-wake transitions. The accelerometer measures acceleration along its x, y, and z axes. Initially, we planned to use the magnitude of acceleration across all axes, which should be approximately 1 g when stationary, to determine whether the device is moving. However, after investigating the Pico's sleep modes, we discovered that such calculations cannot be performed in sleep mode.

To address this, we configured the accelerometer to trigger an interrupt upon detecting an acceleration exceeding a threshold on any of the axes. The challenge was that the 1 g due to gravity would be unpredictably distributed across the three axes, as the device's orientation is not fixed, making it impossible to set a reliable static threshold. Further research revealed that enabling the accelerometer's high-pass filter could eliminate the effect of gravity. After iterative testing, we determined that a threshold of 400 mg per axis was optimal for reliable motion detection.

Putting this together in the main code, in wake mode (normal operation), the logic follows that the device will read and send data over BLE every 7 seconds. And if no movement is detected for over 20 seconds, the device enters a sleep mode. In sleep mode, the device wakes up every 30 seconds to update the sensor data to the phone. If the updated data is abnormal, the device will return to normal operation and alert the user, otherwise it will go back to sleep. At the same time, if a movement interrupt is generated by the accelerometer, the device will wake up and return to normal operation.

The Pico offers several sleep modes. Initially, we attempted to implement the deep sleep mode, where all clocks except the RTC are disabled. However, this caused the BLE to disconnect, and were unable to re-establish that connection reliably. As a compromise, we switched to using a light sleep mode, which reduces the clock frequency from 125 MHz to 12 MHz while maintaining the BLE connection. Since BLE is not the most power-intensive component, we focused on the PM2.5 sensor which consumes significant power. By using its SET pin, we configured the sensor to turn off when entering sleep mode and to power back on 10 seconds before the device wakes up to check for abnormal data.

4.4.3 Android Application

4.4.3.1 System Architecture

The Android application implements a modular architecture designed to handle sensor data acquisition, processing, and visualization. The system utilizes Kotlin's coroutine framework for asynchronous operations and follows the Model-View-ViewModel (MVVM) pattern. The application comprises five primary modules: BLE communication, location services, cloud data management, real-time visualization, and forecasting.

4.4.3.2 BLE Communication

The BLEManager class handles device scanning, connection management, and data reception using Android's BLE API framework. The communication protocol implements a notification-based data transfer system, with the air quality monitor sending six 4-byte floating-point values representing sensor measurements. The system maintains continuous scanning while disconnected, automatically reconnecting to the air quality monitor's unique UUID when available.

4.4.3.3 Location Services Integration

The location service implementation utilizes Android's FusedLocationProvider API, chosen for its optimized battery consumption and automatic selection between GPS and network-based location providers. However, the test phone used in this project relies only on the built in GPS. The system implements adaptive location updates based on device movement and environment type.

Indoor/outdoor detection is determined using GPS time-to-fix measurements. When the time to obtain a GPS fix exceeds 800 milliseconds, the system classifies the location as indoors, and outdoors otherwise. This classification is used in the weighted AQI calculations, detailed later in this section. The approach of using time-to-fix proved highly accurate, with the phone determining indoor/outdoor location at a high success rate, however it should be noted that this struggled while in basement spaces due to the phone GPS' inability to connect to a satellite.

4.4.3.4 Data Processing and Analysis

The application implements an air quality analysis system that interpolates between transmitted sensor readings and a range of AQI values. EPA guidelines for PM2.5 breakpoints and indoor air quality standards were used to interpolate from 0-151+ for each metric. The indices were normalized across metrics to maximize readability in the application.

The total weighted AQI calculation employs environment-specific weightings based on established indoor and outdoor pollutant behavior patterns. For indoor environments, PM2.5 and VOC measurements receive equal weighting (0.4 each) due to limited ventilation conditions and VOC accumulation. Outdoor environments shift priority to PM2.5 (0.5) over VOC (0.3) due to rapid VOC dispersion, higher PM2.5 prevalence from vehicular emissions, and longer PM2.5 suspension times. This adaptive system is expressed in equation 4:

$$AQI_{total} = (w_{pm} * PM_{AQI}) + (w_{voc} * VOC_{AQI}) + (0.2H * 50) \quad (4)$$

For indoor environments: $w_{PM} = 0.4$, $w_{VOC} = 0.4$
For outdoor environments: $w_{PM} = 0.5$, $w_{VOC} = 0.3$

This adaptive weighting system gives the application user a more comprehensive metric that should be useful for health monitoring on top of the PM2.5 and VOC indices.

4.4.3.5 Cloud Database Backend

The system implements Firebase Realtime Database for data storage and synchronization, chosen for its integration capabilities and free usage tier. This platform enables automatic offline data caching and real-time synchronization across multiple devices. The database organizes measurements by date and timestamp, supporting multiple air quality monitoring devices simultaneously. Each measurement record stores sensor data, GPS coordinates, indoor/outdoor classification, and timestamp information, with atomic write operations ensuring data consistency during concurrent access.

4.4.3.6 Map Integration

The mapping implementation utilizes OpenStreetMap through the osmdroid library, selected for its free-to-use nature and customization capabilities. The system implements custom overlay rendering for air quality visualization, displaying measurement data through color-coded markers and interpolated heat maps. The visualization system employs a spinner for viewing each index: PM2.5 AQI, VOC AQI, and Total AQI.

4.4.3.7 Forecast Model Management

The application stores a TensorFlow Lite model with associated parameters JSON file as assets. The model architecture employs a 64x64 spatial grid to represent the geographical area, with each grid cell containing PM2.5, VOC, and total AQI values. The forecast generation process converts historical GPS-based readings into the model's grid representation, maintaining temporal ordering across the 7-day sequence.

The manager employs bilinear interpolation for multiple readings within cells and applies MinMaxScaler normalization before processing through the TFLite interpreter. Post-processing focuses on converting predictions back to GPS coordinates, specifically targeting areas with significant historical activity. The implementation uses smoothing algorithms to reduce prediction artifacts and employs noise filtering to remove predictions without significant deviation from baseline values, optimizing computational efficiency and forecast relevance.

4.4.4 Forecasting Model

The air quality forecasting function uses a variational auto-encoder architecture implemented in TensorFlow to predict pollutant distributions across urban environments. The model processes 7-day historical sequences of PM2.5, VOC, and total AQI measurements through a grid-based representation system, utilizing a 64x64 spatial discretization to capture local pollution patterns.

The VAE architecture implements a hybrid convolutional-LSTM structure in order to encode both spatial patterns as well as temporal patterns–both are important given our application. The encoder pathway processes daily measurements through three convolutional layers (32, 64, and 128 filters) with batch normalization and max pooling operations. These spatial features feed into dual LSTM layers (512 and 256 units) for temporal pattern extraction. The decoder reconstructs predictions through a series of transposed convolutions, maintaining the original spatial resolution while generating next-day forecasts.

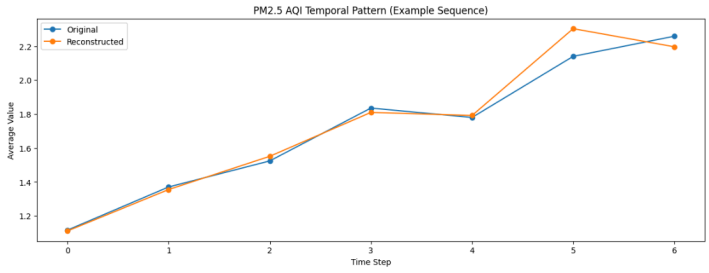

Fig 7. TIme step evaluation of forecasting model comparing average AQI values

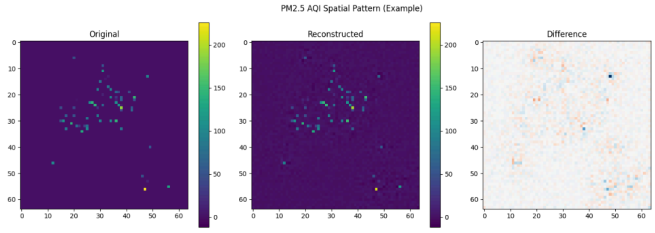

Fig 8. Sample reconstruction evaluation graphs for PM2.5 metric

The training implementation utilizes a composite loss function combining three components: reconstruction loss (mean squared error between predictions and ground truth), temporal consistency loss (mean squared error between consecutive day differences), and KL divergence loss for latent space regularization. These components are weighted to place the most importance on reconstruction loss, medium importance on temporal loss, and least importance on KL divergence loss. Overall, our model achieved excellent results with an MAE of 1.49 and an RMSE of 2.13 for PM2.5, an MAE of 0.50 and RMSE of 0.70 for VOC, and an MAE of 0.77 and RMSE of 1.08 for the total weighted AQI.

5. EVALUATION AND RESULTS

To reach one of our distinctions, it was important to evaluate our power management solution to determine if we reached our goal of a 15% power saving improvement. To clarify, this improvement is in reference to the baseline where the device is in normal operation mode. A small load resistor with a nominal value of 1.6 Ω was placed between the 3.7 V power source and the power input to the Pico. Then an oscilloscope was used to accurately determine the voltage across the resistor, which can be used to calculate the total current drawn by the device at any given time. The data collected over a period of time is shown in Fig. 7 below. Notice that the overall data is very noisy, however, a distinct pattern can still be seen. The series of lower current readings correspond to that low-power sleep mode, whereas the higher readings correspond to normal operation.

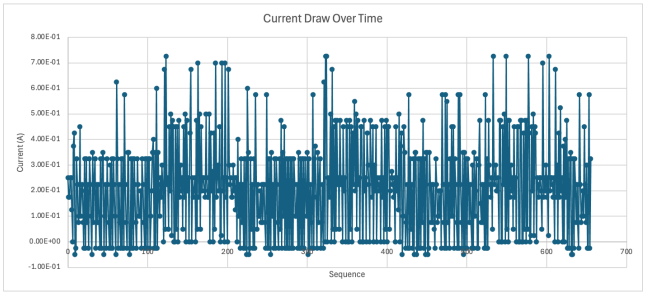

Fig 9. Current Draw of Device

Taking the average over 100 data points in each region, Fig. 8 shows the average current in normal operation is 0.25 A (250 mA) and is 0.16 A (160 mA) in sleep mode. This indicates a 36% power saving in sleep mode as compared to normal operation.

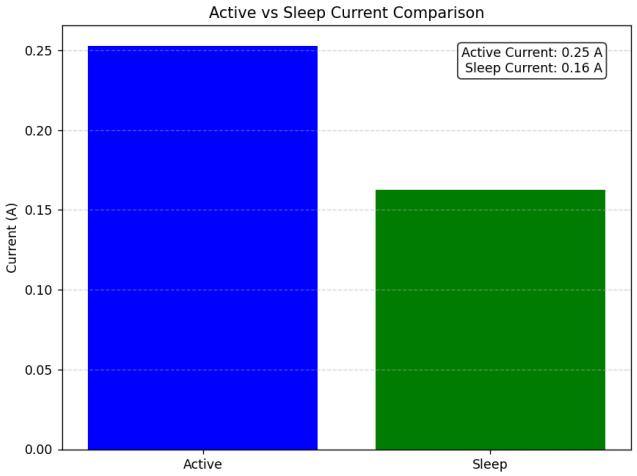

Fig 10. Active vs. Sleep Current

Another way our device was evaluated was through a live drop test. Unfortunately, because the PCB had to be removed from its case to show serial output on a computer, we were unable to maximally secure the lid of the case. As a result, upon hitting the ground, the lid did come unattached. Despite this not being the ideal result, we were able to show the device could still function as expected. It was reading and sending data from the sensors, and successfully triggered a notification on our Android app in the presence of abnormal air quality.

6. DISCUSSION

In this project, the sensors accurately collect data and transmit it over BLE to a smartphone. The phone effectively displays all metrics clearly and displays them on a map for easy visualization. In terms of power management, the device achieves approximately 36% energy savings compared to active mode. However, there is still room for improvement in the overall system. For instance, implementing a deep sleep mode instead of a light sleep mode could further enhance power savings by fully turning off the CPU and system clock frequencies during inactivity. Additionally, predictive forecasting can be improved by utilizing more robust models and incorporating larger datasets to provide more reliable results. Lastly, the current case design is bulky, making it less convenient as a wearable clip. Redesigning the case to reduce its size and enhance portability would improve usability and practicality.

7. CONCLUSION

This project successfully demonstrates the feasibility and functionality of a wearable IoT device for localized air quality and environmental monitoring. By integrating the Raspberry Pi Pico W as the central microcontroller, the system effectively combines PM2.5, temperature, humidity, VOC, and accelerometer sensors into a portable solution. Intelligent power management, enabled by the accelerometer, significantly reduces energy consumption, with light sleep mode saving approximately 36% of the energy

compared to the active mode. The device's BLE capabilities ensure data transmission to a smartphone for real-time display and cloud integration, supporting predictive forecasting through machine learning.

8. REFERENCES

[1] S. Jangid and S. Sharma, "An embedded system model for air quality monitoring," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2016, pp. 3003-3008.

[2] S. Kumar and A. Jasuja, "Air quality monitoring system based on IoT using Raspberry Pi," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, 2017, pp. 1341-1346, doi: 10.1109/CCAA.2017.8230005.

[3] M. Kolehmainen, H. Martikainen, and J. Ruuskanen, "Neural networks and periodic components used in air quality forecasting," Atmospheric Environment, vol. 35, no. 4, pp. 815-825, 2001, doi: 10.1016/S1352-2310(00)00385-X.