American University of Sharjah
Department of Computer Science and Engineering

AIoT Based Energy Efficient Approach for Air Quality Monitoring and Predictive Analysis

This Project is submitted to the department of Computer Science and Engineering at the College of Engineering in partial fulfillment of the requirements for the degree of

Bachelor of Science Degree in
Computer Engineering

Presented By	**Student IDs**
Joel John Dcruze	b00089160
Warda Ul Hasan	g00090299
Dwayne Fonseca	b00088027
Ahmed Al Refiay	b00087610

Advised By

Dr. Mohamed Alhajri

Date of Submission

4th May 2024

Table of Contents

List of Tables

List of Figures

List of Equations

List of Abbreviations

Abbreviation	Definition
AQI	Air Quality Index
UAE	United Arab Emirates
WHO	World Health Organization
WSN	Wireless Sensor Nodes
ANN	Artificial Neural Networks
GUI	Graphical User Interface
PWM	Pulse Width Modulation
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
LR	Linear Regression
SLR	Single Linear Regression
MLR	Multiple Linear Regression
KNN	K-Nearest Neighbors
KNNR	K-Nearest Neighbors Regression
R^2	Coefficient of Determination
SVM	Support Vector Machine
SVM	Support Vector Machine

SVR	Support Vector Regression
RBF	Radial Basis Function
SSE	Sum Square Error
NLR	Non-Linear Regression
NN	Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory

Abstract

Air pollution, a recognized major environmental risk by the World Health Organization (WHO), poses a significant global threat, contributing to a spectrum of health issues, including heart disease, stroke, cancer, and respiratory diseases. In the United Arab Emirates (UAE), outdoor air quality stands as a pressing challenge, ranking as the primary cause of environmental-linked deaths, even affecting adolescents who suffer from asthma due to environmental factors. The UAE government's strong commitment to improving air quality, as evidenced by initiatives such as the UAE Vision 2021 and the UAE Air Quality Agenda 2031, underscores the pivotal role of air pollution measurement in its environmental strategy. This project addresses the Monitoring Pillar of the national agenda with an innovative solution: an AIoT-based Energy-Efficient Approach to Air Quality Monitoring and Prediction Analysis. Our goal is to create an energy-efficient and cost-effective system for monitoring air quality and predicting pollution patterns, with a particular focus on regions lacking adequate coverage due to a shortage of weather stations. We propose deploying a network of low-power weather stations across key locations in the university, leveraging cost-effective low-power components, and energy-efficient communication protocols. This comprehensive system will empower individuals to proactively engage with environmental challenges while enhancing predictive capabilities, ultimately contributing to a healthier UAE with improved air quality.

1. Introduction

Our project aims to address the challenge of limited air quality monitoring and forecasting in the UAE by proposing a low-power consuming system incorporating artificial intelligence and the Internet of Things to monitor and predict outdoor air quality: AIoT Based Energy Efficient Approach for Air Quality Monitoring and Predictive Analysis.

1.1 Motivation

Air pollution is a major global concern with severe complications for public health and the environment. The World Health Organization (WHO) in 2022 recognized air pollution as a major environmental risk to health, causing death and disability worldwide, including heart disease, stroke, COPD, cancer, and pneumonia [1]. It is a serious issue not limited to only low-income countries but affects people across all income levels. In the United Arab Emirates (UAE), poor outdoor air quality ranks as the top cause of environmental-linked deaths [2]. Alarmingly, it also impacts adolescents, with at least 13% of Emirati children suffering from asthma due to indoor and outdoor environmental factors [3]. While numerous studies have focused on indoor air quality monitoring in the UAE, research on outdoor air quality monitoring systems is limited. However, the UAE government has demonstrated a strong commitment to improving air quality, as evident in the UAE Vision 2021 and the recently released UAE Air Quality Agenda 2031 [4, 5, 6]. This ambitious agenda is built upon three pillars: Monitoring, Mitigation, and Management. The Monitoring pillar emphasizes the measurement of air pollution and related parameters as a crucial component of the UAE's environmental strategy [6].

1.2 Project Description

Our project aims to address the first pillar of the national agenda — Monitoring Pillar — by proposing an innovative solution: an AIoT-based Energy Efficient Approach to Air Quality Monitoring and Prediction Analysis. We aim to offer an energy-efficient and cost-effective AIoT-based approach for Air Quality Monitoring and Predictive Systems. Our aim is to develop an energy-efficient and cost-effective system for monitoring air quality and predicting pollution patterns. To address the issue of inadequate weather stations, our project proposes the design and deployment of a network of low-power weather stations across our university campus and target regions. These stations will provide high-resolution, real-time

air quality data. This access to environmental data will empower individuals to actively engage with and address environmental challenges. Moreover, we will use relatively inexpensive and low-power components to ensure easy implementation and maintenance of stations throughout their lifecycle. Our IoT-based air quality monitoring system will integrate high-accuracy, low-power sensors to measure key parameters such as temperature, humidity, and pollutant gasses. To ensure energy efficiency, we plan to incorporate low-power microcontrollers and leverage renewable energy sources, such as solar panels. Communication will rely on energy-efficient protocols like WiFi, or Zigbee for transmitting data over long distances with minimal power consumption. Real-time data processing and storage in the cloud and time-series databases, along with a user-friendly dashboard, will facilitate easy access and interpretation of environmental metrics. Machine learning techniques will further enhance predictive capabilities.

Our project offers several significant benefits, including improved air quality monitoring through high-resolution, real-time data, prioritizing energy efficiency with low-power components and renewable energy sources, cost-effectiveness by using relatively inexpensive materials, expanding coverage by deploying a network of low-power weather stations, empowering the public with access to environmental data, facilitating data-driven decision-making with a user-friendly dashboard and cloud data storage, and enhancing predictive capabilities through machine learning techniques. These advantages collectively contribute to a more comprehensive understanding of air quality challenges, timely interventions, and informed decision-making, aligning with the UAE's commitment to environmental improvement and public health.

1.3 Organization of the Report

The report is organized into four main sections, each focusing on a different aspect of the AIoT-based approach for air quality monitoring and predictive analysis in the UAE. Section 1 introduces the project, outlining the motivation and overall description. Section 2 delves into the problem statement and design objectives, detailing the specific air quality issues in the UAE, the proposed solution, the design objectives, and potential limitations of the project. Section 3 presents a literature review, covering the air quality situation in the UAE, existing IoT-based weather monitoring systems, machine learning algorithms for air quality prediction, and cloud-based weather station using IoT devices. Finally, Section 4

concludes the report, summarizing the project's objectives and its relevance to the UAE's goals for air quality monitoring and environmental sustainability.

2. Problem Statement and Design Objectives

2.1 Problem Statement and Proposed Solution

In the UAE, the Air Quality Index (AQI) national platform, provides air quality monitoring and predictive analysis. The national AQI platform provides valuable insights into air quality, but its coverage is limited. For example, in many parts of Sharjah, air quality remains unmonitored and unforecasted due to the scarcity of weather stations. Despite the presence of 54 weather stations, there is a lack of comprehensive coverage, leaving many regions unmonitored and unforecasted. This limitation hinders the understanding of air quality challenges and timely interventions. The consequence is an incomplete picture of the extent of air pollution and its adverse effects on public health and the environment.

Our project aims to address the first pillar of the national agenda — Monitoring Pillar — by proposing an innovative solution: an AIoT-based Energy Efficient Approach to Air Quality Monitoring and Prediction Analysis. We aim to offer an energy-efficient and cost-effective AIoT-based approach for Air Quality Monitoring and Predictive Systems. Our aim is to develop an energy-efficient and cost-effective system for monitoring air quality and predicting pollution patterns. To address the issue of inadequate weather stations, our project proposes the design and deployment of a network of low-power weather stations across our university campus and target regions. These stations will provide high-resolution, real-time air quality data. This access to environmental data will empower individuals to actively engage with and address environmental challenges. Moreover, we will use relatively inexpensive and low-power components to ensure easy implementation and maintenance of stations throughout their lifecycle. Our IoT-based air quality monitoring system will integrate high-accuracy, low-power sensors to measure key parameters such as temperature, humidity, and pollutant gasses. To ensure energy efficiency, we plan to incorporate low-power microcontrollers and leverage renewable energy sources, such as solar panels. Communication will rely on energy-efficient protocols like WiFi, or Zigbee for transmitting data over long distances with minimal power consumption. Real-time data processing and storage in the cloud and time-series databases, along with a user-friendly dashboard, will

facilitate easy access and interpretation of environmental metrics. Machine learning techniques will further enhance predictive capabilities.

2.2 Design Objectives

2.2.1 Campus Monitoring: Develop a comprehensive air quality monitoring and prediction system for our university campus, the American University of Sharjah.

2.2.2 Low Power and Energy Efficiency: Prioritize low power consumption, energy efficiency, and cost-effectiveness in designing both hardware and software components of the air quality monitoring and prediction system, including low power-consuming microcontrollers. Create a low-powered and energy-efficient solution that minimizes power usage, ensuring uninterrupted operation while being financially sustainable for the campus.

2.2.3 High-Quality and Low-Power Sensor Integration: Employ high-quality sensors capable of measuring critical parameters PM2.5, PM10, CO2, NO2, Ozone, and CO, while ensuring they are low-powered to minimize energy consumption. These sensors must provide accurate and reliable data for air quality assessments within the campus, contributing to both data quality and energy efficiency.

2.2.4 Real-Time Data Processing: Establish an efficient and responsive data processing pipeline tailored for real-time analysis. This enables timely responses to air quality changes within the campus area.

2.2.5 Predictive Analysis Capability: Integrate machine learning techniques for predictive analysis tailored to the campus environment. Utilize historical datasets from the campus for model training, enabling the system to forecast future air quality trends within the campus.

2.2.6 Reliable Communication: Develop a communication infrastructure within the campus that enables seamless data transmission between monitoring stations.

2.2.7 User-Friendly Campus Dashboard: Create a user-friendly dashboard for the campus that provides intuitive data visualization. Enable campus users to access current and historical air quality data.

2.3 Limitations

2.3.1 Cost: Selecting accurate but low powered sensors to detect the target gasses will be costly in terms of monetary resources. There will be budget constraints in terms of buying more sensors and microcontroller components to establish the scalability of our device into multiple duplicates to span over a specific area.

2.3.2 WiFi in node locations: Our project requires a thorough scanning of areas with strong wifi infrastructure. This is important in order to establish our devices outside those buildings such that they have a reasonable data rate to transmit sensor readings to a system dashboard.

2.3.3 Sensor Resilience: There can be random errors in sensors with respect to measuring the target gasses which can result in them not working as detailed in their respective data sheet.

2.3.4 Weather Conditions: As these devices are to be implemented in the exterior areas of nearby buildings, there is uncertainty on how the weather conditions might affect the devices. An example of this can be heavy rains or sand storms that can affect how the sensor works as

intended.

2.3.5 Accessibility to node locations: As the experimentation covers vast land space, for example, the AUS campus, our project will require permission from the respective authorities to access certain areas around the campus in order to install our devices.

2.4 Assumptions

Our project operates under the assumption that the sensors utilized for measuring the target gases—O3, NO2, CO, PM 2.5, and PM 10—accurately adhere to their respective datasheets. We acknowledge the potential for random errors but, for the purposes of this project, we consider these sensors to perform reliably and with precision. Another critical assumption involves the stability of power consumption throughout the monitoring period. We expect the ESP32 MCU, chosen for its low power consumption, integrated Wi-Fi module, and user-friendly IDE, along with other hardware components, to maintain consistent energy usage, ensuring reliable and uninterrupted data acquisition from the sensors. The reliability of data transmission is pivotal to the project's success. We assume that the integrated Wi-Fi

module of the ESP32, responsible for transmitting sensor data from the MCU to the cloud system, functions reliably, avoiding significant data loss or interruptions in transmission. Moreover, ESP32's versatility allows it to interface with other systems, providing Wi-Fi and Bluetooth functionality through its SPI / SDIO or I2C / UART interfaces.

In considering the cloud system's role, we assume its ongoing availability for data reception and storage. This assumption extends to the stability and uptime of the dashboard system and server, enabling developers to visualize and analyze air quality data without enduring notable downtime.

For the machine learning component, we assume that the models employed for predictive analysis are trained on accurate and representative datasets. Furthermore, we expect these models to maintain their accuracy, providing reliable predictions based on incoming sensor data. Considering external factors, we assume that the system's performance is not significantly impacted by extreme weather conditions. We account for potential challenges such as heavy rain or storms that might affect sensor readings or disrupt data transmission. We acknowledge the importance of response time in our system. Thus, we assume a reasonable response time, from data acquisition to analysis and visualization, to deliver timely and relevant information to end-users.

The assumption of user accessibility underlines the expectation that the website delivering air quality information remains accessible to users without prolonged downtime. This accessibility ensures continuous availability of vital information for the public. Users play a crucial role in the effectiveness of our system. Thus, we assume that users interpret air quality information accurately and understand the limitations of the system. This understanding is vital as the system provides estimates based on sensor readings and predictive models. In presenting these assumptions, we aim to provide a comprehensive understanding of the conditions under which the system is expected to operate optimally, allowing for transparency and clear expectations.

3. Literature Review

The Literature review will be divided into 6 parts as shown below in the figure.

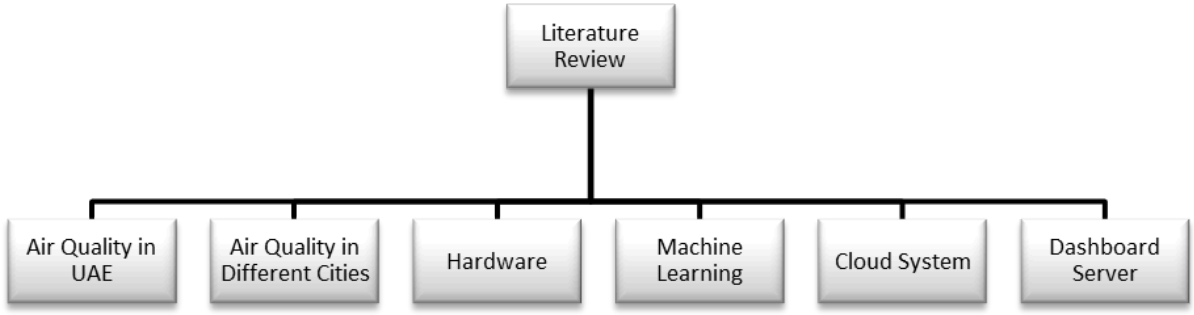


Figure 1: Literature Review Sections

3.1 Air Quality in the UAE

Air pollution is a major concern in the United Arab Emirates (UAE) and worldwide, with severe implications for public health and the environment. According to the World Health Organization (WHO) in 2022, air pollution is recognized as one of the greatest environmental risks to health [1]. It has been classified as a leading cause of death and disability worldwide due to heart disease, stroke, cancer, and pneumonia [2]. Moreover, outdoor air pollution is a major environmental health problem affecting people in low-, middle-, and high-income countries [1]. In fact in the UAE, poor outdoor air quality stands as the top cause of environmental-linked deaths [3]. It also affects adolescents and is associated with industrial proximity and geography as predictors of asthma. At least 13% of Emirati children in the UAE suffer from asthma, due to indoor and outdoor environmental factors [2]. Although there are numerous studies about indoor air quality monitoring in the UAE, there is a dearth of research focused on outdoor air quality monitoring systems. The UAE government is consistently expressing its strong commitment to improving air quality, as outlined in the UAE Vision 2021 [4, 5]. More importantly, recently, the UAE Air Quality Agenda 2031 has been built upon three pillars: Monitoring, Mitigation, and Management. The Monitoring pillar focuses on measuring air pollution and related parameters; the Mitigation pillar involves a spectrum of actions to reduce emissions and exposure; and the

Management pillar ensures effective implementation, tracking, and control of initiatives to ultimately enhance air quality [6]. In the UAE, the Air Quality Index (AQI) national platform, provides air quality monitoring and predictive analysis. The national AQI platform provides valuable insights into air quality, but its coverage is limited. For example, in many parts of Sharjah, air quality remains unmonitored and unforecasted due to the scarcity of weather stations [9].

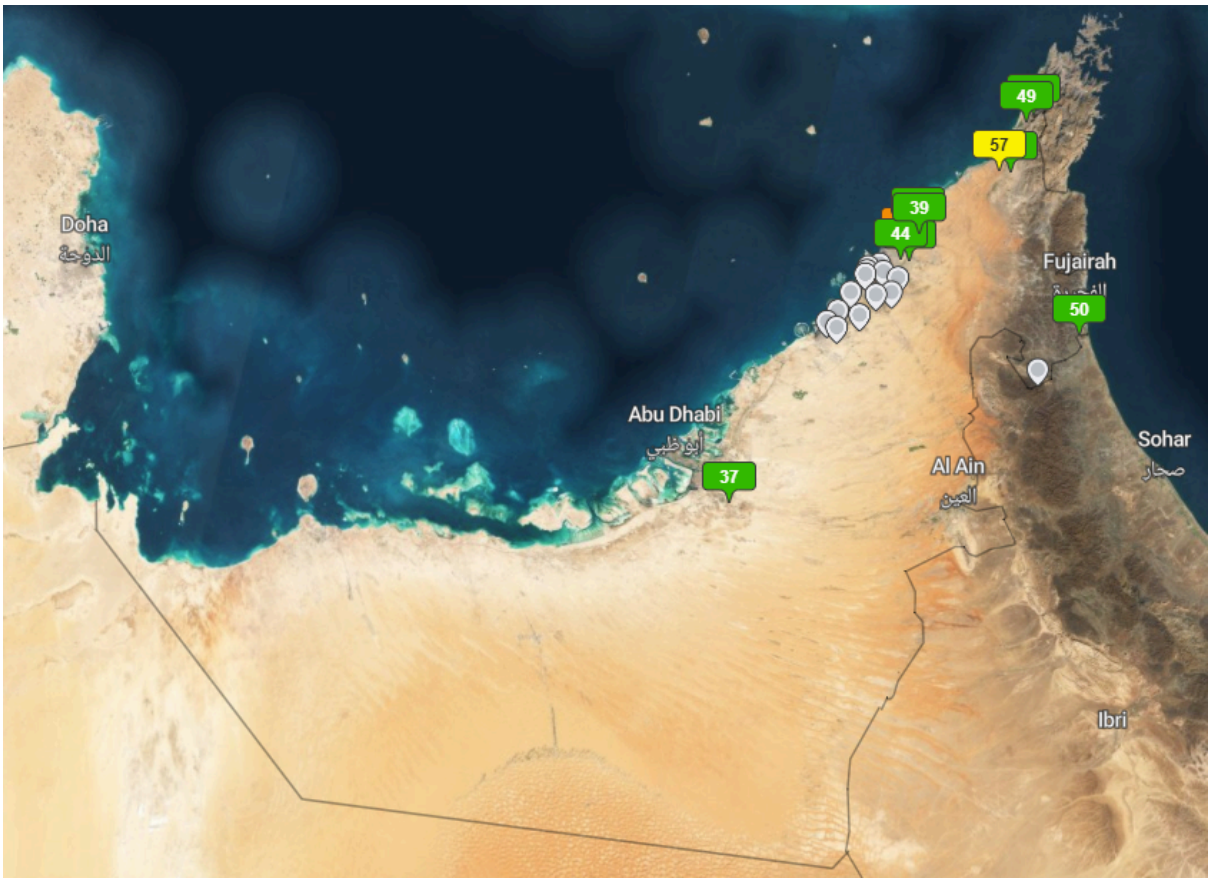


Figure 2: Air Quality Monitoring in UAE

3.2 Air Quality in the Different Cities

Figure 3: Locations of cities regarding air quality

The Air Quality Index (AQI) varies significantly across cities around the world like Delhi, Los Angeles, Lahore, Shenyang, Kolkata, Karachi, Baghdad, Dubai, and Tashkent, reflecting the diverse sources and magnitudes of air pollution in different regions. The range of AQI values, from hazardous levels in Delhi to unhealthy levels in cities like Lahore and Shenyang, underscores the urgent need for proper air quality monitoring. These variations can be attributed to a myriad of factors, including industrial emissions, vehicular traffic, geographical conditions, and meteorological patterns. The AQI serves as a critical tool in quantifying the extent of air pollution, providing a standardized metric that allows for easy comparison across cities. Proper air quality monitoring is essential for the development and implementation of targeted interventions and policies aimed at mitigating pollution sources, protecting public health, and fostering sustainable urban development. As evidenced by the diverse AQI values in these cities, a tailored and localized approach to air quality management is necessary, emphasizing the vital role of continuous monitoring systems in understanding, addressing, and preventing the detrimental effects of poor air quality on global populations.

- Delhi, India :

Delhi, the capital of India, grapples with severe air pollution, particularly during winter months. Contributing factors include crop burning, vehicular emissions, and industrial

activities. An AQI of 701 categorizes the air quality as "Hazardous," indicating a significant health risk for the entire population.

- Los Angeles, United States of America:

With an Air Quality Index (AQI) of 115, the city falls into the category of 'Unhealthy' air quality. The AQI value reflects the presence of pollutants in the atmosphere that may pose health concerns, especially for sensitive groups. In Los Angeles, factors such as vehicular emissions, industrial activities, and the region's topography contribute to air quality challenges. The inclusion of Los Angeles in this discussion further highlights the global nature of air quality issues, as even cities in developed countries face challenges in maintaining optimal air quality.

- Lahore, Pakistan :

Lahore, a major city in Pakistan, faces elevated air pollution levels attributed to industrial emissions and heavy traffic. An AQI of 328 places the air quality in the "Very Unhealthy" range, posing substantial health risks, especially for individuals with pre-existing health conditions.

- Shenyang, China :

Shenyang, an industrial hub in China, contends with air quality challenges stemming from industrial emissions and coal usage. An AQI of 185 falls into the "Unhealthy" category, suggesting that adverse health effects may be experienced by the general population.

- Kolkata, India :

Kolkata, a bustling city in eastern India, faces air pollution issues from various sources, including industrial activities and traffic congestion. An AQI of 183 categorizes the air quality as "Unhealthy," raising concerns about potential health impacts on the population.

- Karachi, Pakistan :

Karachi, the largest city in Pakistan, deals with air quality challenges arising from industrial emissions and high vehicular traffic. An AQI of 174 falls into the "Unhealthy" range, indicating potential health risks for residents.

- Dhaka, Bangladesh :

Dhaka, the capital of Bangladesh, grapples with air pollution linked to rapid urbanization and industrialization. An AQI of 169 classifies the air quality as "Unhealthy," emphasizing health concerns, especially for sensitive groups.

- Mumbai, India:

Mumbai, a densely populated coastal city, contends with air pollution from diverse sources, including traffic and industrial emissions. An AQI of 167 categorizes the air quality as "Unhealthy," raising health considerations for the general population.

- Baghdad, Iraq:

Baghdad, the capital of Iraq, faces air quality challenges, including dust storms and pollution from industrial sources. An AQI of 157 designates the air quality as "Unhealthy," indicating potential health effects for everyone.

- Dubai, United Arab Emirates:

Dubai, a major global city known for its rapid development, encounters air quality issues linked to industrial activities and construction. An AQI of 146 is classified as "Unhealthy for sensitive groups," suggesting potential health effects for vulnerable individuals.

- Tashkent, Uzbekistan :

Tashkent, the capital of Uzbekistan, contends with air quality challenges arising from industrial activities and rapid urbanization. An AQI of 128 categorizes the air quality as "Unhealthy for sensitive groups," highlighting potential health effects for vulnerable individuals.

These details shed light on the specific factors contributing to poor air quality in each city and emphasize the urgent need for comprehensive air quality management strategies and monitoring systems.

3.3 Hardware

Effectively managing power consumption in wireless sensor networks (WSNs) is imperative for sustainable and efficient environmental monitoring. In this context, three research studies provide valuable perspectives. One study delves into an Internet of Things (IoT)-enabled, low-power environmental monitoring system, emphasizing the importance of specific components and predictive modeling [6]. The study pioneers an IoT-enabled environmental monitoring system designed for real-time tracking of various air quality parameters. Demonstrating commendable low power consumption, the system integrates an ultra-low-power microcontroller, a wireless transceiver, and sensors measuring diverse environmental parameters. Notably, the study employs Artificial Neural Networks (ANN) to predict PM2.5 levels, circumventing the need for power-intensive sensors and offering an innovative approach to predictive modeling.

Another study introduces an energy-efficient wireless environment monitoring system based on the IEEE 802.15.4 standard, leveraging an electrochemical sensor array to achieve both accuracy and reduced power consumption [10]. Building on the pursuit of energy efficiency, the second study introduces a wireless environment monitoring system based on the IEEE 802.15.4 standard. Employing an electrochemical sensor array, this system achieves a notable reduction in power consumption while maintaining extended operational duration. The architectural components include a wireless transducer interface and an application processor module, offering a cost-effective and accurate solution for monitoring greenhouse gasses and environmental parameters. The study not only addresses the challenge of power consumption but also emphasizes the importance of cost-effectiveness and precision in environmental monitoring systems.

In this paper titled “ISSAQ: An Integrated Sensing Systems for Real-Time Indoor Air Quality Monitoring”, explores real-time indoor air quality monitoring, presenting an integrated sensing system that incorporates sensor nodes and WSN. This study focuses on real-time indoor air quality monitoring, emphasizing the intricacies of monitoring various gasses and environmental parameters. Utilizing Raspberry Pi architecture, the study introduces an integrated sensing system with sensor nodes and WSN for seamless data transmission. This research addresses challenges related to outdoor monitoring, battery

limitations, and environmental conditions, providing insights into the nuances of real-time indoor air quality monitoring.

Taken together, these studies underscore the multifaceted approach required to achieve energy efficiency in environmental monitoring through wireless sensor networks. While one study highlights the significance of low-power components and predictive modeling, another introduces an electrochemical sensor array for accurate and energy-efficient monitoring. Simultaneously, the third study explores an integrated sensing system, emphasizing the importance of real-time monitoring and indoor air quality. The collective insights offer a holistic understanding of power management strategies in WSNs, providing comprehensive guidance for addressing the challenges of monitoring diverse environmental parameters with a primary focus on energy efficiency.

Silviu et al. [12] introduce a compact, battery-powered wireless sensor system designed for continuous indoor ambient monitoring. The system, powered by a single 3 V CR123A lithium battery, incorporates a low-power PSoC 3 microcontroller and a suite of sensors, including the CozirTM CO2 Ambient Sensor, DHT22 digital temperature and humidity sensor, MPL115A2 barometer sensor, and TSL2561 light sensor. These sensors were chosen based on their satisfactory range, accuracy, and minimal power consumption, contributing to the system's efficiency. The CozirTM CO2 sensor, tailored for battery-powered applications, exhibits an average power consumption of less than 3.5 mW and can measure concentrations between 0 and 2000 ppm. The DHT22 sensor provides temperature and humidity data with a power consumption of 1 mA in active mode and 40 μA in sleep mode. The paper emphasizes the integration of Wi-Fi connectivity using the WiFly module, enabling data transmission based on IEEE 802.11 b/g standards. The device's hardware architecture includes a sensing unit, processing/storage unit, transceiver, and power supply, ensuring its suitability for wireless sensor networks and Internet of Things scenarios. With meticulous attention to power-saving strategies, the system demonstrates a remarkable battery life of up to three years under specific measurement rates. It is a competitive and viable solution for prolonged autonomous operation in ambient monitoring applications.

In another paper, James et al. [13] design a highly modular environmental sensing system designed for monitoring airborne particulate matter and other environmental factors in enclosed spaces. Central to the system is the use of an Atmel AVR32UC3A3256S 32-bit

microcontroller, known for its high performance, low power consumption, and extensive integrated features. This microcontroller serves as the main processor for sensor nodes within a WSN. The nodes, part of a self-organizing mesh network, communicate real-time data from a suite of sensors, including those for particulate matter, selected gases, humidity, temperature, and pressure. The reference implementation for the National Institutes of Health incorporates a custom optical particle counter and off-the-shelf sensors for CO2, CO, temperature, humidity, pressure, and acoustic noise. The optical particle counter operates on the principles of Mie scattering and uses a 3mW, 650nm laser assembly. The sensor nodes offer flexibility, enabling the deployment of custom sensor suites with minimal development effort. These nodes communicate with a coordinator node, which relays time-correlated information to a server-hosted database through wired or wireless networks. The system's adaptability and affordability make it well-suited for monitoring ambient air quality in various environments, providing continuous, real-time data for analysis. Additionally, the WSN design focuses on a low-cost implementation of a particulate matter sensor, addressing limitations of size, standalone nature, and cost found in existing commercial devices. The design incorporates multiple communication options and local data storage, enhancing the robustness of the system. The paper emphasizes the use of off-the-shelf components and provides an approximate cost breakdown for the particle counter, highlighting the feasibility of mass production and potential improvements in component cost and quality.

Another example of a similar system can be attributed to Helton et al. [14] which addresses the pressing issue of air pollution and its severe impact on human health, particularly relevant in the context of the COVID-19 pandemic. The proposed solution introduces an Internet of Things (IoT)--based air quality monitoring system utilizing cost-effective sensors recommended by the WHO. The device hardware incorporates PMSA003, MICS-6814, and MQ-131 sensors to measure concentrations of Particulate Material (PM2.5 and PM10), Ozone, Carbon Monoxide, Nitrogen Dioxide, and Ammonia. Facilitated by the ESP-WROOM-32 microcontroller with Wi-Fi and Bluetooth capabilities, the device transmits data to a cloud server. What sets this proposal apart is its incorporation of periodic notifications and alerts for pollutants surpassing acceptable concentrations. In addressing the scarcity of information on air quality, the study underscores the significance of IoT devices and affordable sensors in accurately gauging atmospheric gas and pollutant concentrations. Positioned as a tool to combat environmental and health impacts, the monitoring system holds potential significance in the ongoing fight against diseases like

COVID-19 by providing real-time air quality data for informed decision-making and impact minimization. The comprehensive IoT-based air quality monitoring solution not only involves hardware components but also contemplates future advancements. The ESP-WROOM-32 microcontroller, sensors, and additional features like temperature and humidity monitoring contribute to an estimated hardware cost of $75.00. Looking forward, the proposal envisions expanding sensor capabilities to monitor new pollutants, capturing meteorological data, and leveraging artificial intelligence for forecasting pollutant concentration trends. These developments promise to enhance the functionality and versatility of the monitoring system, contributing to more effective environmental management and public health initiatives.

Jalpa et al. [15] focus on the development and practical implementation of an Internet of Things (IoT) enabled Environmental Monitoring System tailored for Smart Cities. The primary aim is to enhance resource utilization efficiency and provide superior services to citizens, particularly in areas such as air quality management, weather monitoring, and home/building automation. The foundational parameters of this smart city system include temperature, humidity, and CO2 levels. The IoT-enabled sensor nodes (IoT-SN) are crucial components of this system, incorporating temperature and humidity sensors (SHT11), a CO2 sensor, an ultra-low-power microcontroller (PIC24F16KA102), and a wireless transceiver. These sensor nodes collect data, which is then transmitted to a receiver node. The received data is not only monitored and recorded in an Excel sheet on a personal computer through a Graphical User Interface (GUI) designed in LabVIEW but also made accessible remotely through an Android application on smartphones. A detailed breakdown of the sensor specifications reveals that the SHT11 sensor provides a fully calibrated digital output with a wide operating range for temperature (-40°C to ±123.8°C) and relative humidity (0 to 100% RH). The microcontroller, a PIC24F16KA102, operates on extreme low-power (XLP) technology with multiple power management modes, making it ideal for low-power algorithms in WSN applications. The paper concludes with a successful implementation and validation of the IoT-enabled environmental monitoring system in various locations in the city of Gandhinagar, Gujarat, India. Notably, the system demonstrates lower power consumption (4.99mW) and a reliability rate of approximately 65% in a multi-hopping mechanism, while a single-hopping mechanism ensures more than 99% reliability. Future work is outlined, focusing on improving multi-hopping mechanism reliability, optimizing power consumption, and extending network lifetime through microcontroller sleep time adjustments.

Sharafat et al. [16] introduces an innovative solution for comprehensive air pollution monitoring through the development of a low-cost sensor node equipped with IoT LoRaWAN connectivity and machine learning-based calibration. Traditional air quality monitoring stations, constrained by high costs, are often limited in number and struggle to capture local variations. The proposed sensor node, designed for real-time and widespread monitoring, utilizes cost-effective electrochemical sensors for carbon monoxide (CO) and nitrogen dioxide (NO2), along with an infrared sensor for particulate matter (PM) levels. The unit can be powered either by solar-recharged batteries or mains supply, offering both long-range, low-power communication via LoRaWAN and short-range, high data rate communication through Wi-Fi. In terms of affordability, the sensor unit's overall design prioritizes cost-effectiveness. Excluding PCB manufacturing and assembling costs, the construction of one sensor node is estimated at approximately $250. This stands in stark contrast to the high costs associated with commercially available air pollution sensors, with handheld devices reaching around $5000 and permanent installations costing considerably more. The power management module of the sensor unit is a pivotal component. It incorporates a 4000-mAh LiPo battery charged by a 3 W, 9-V solar panel, providing a two-state constant current—constant voltage charge cycle. The power draw includes a fixed component (0.6 mA for standby power consumption) and variable components, with the gas sensor circuits being the major contributor at 0.556 mA. Maintaining a constant draw of 2 mA allows for excess capacity, accommodating data logging or on-board computation. The intermittent power consumption during data acquisition and transmission introduces a tradeoff between reading frequency and power conservation, emphasizing the meticulous consideration of power dynamics in the sensor's design.

For outdoor purposes, a Low-Cost Outdoor Air Pollution Monitoring Device (APMD) has been designed by Payali et al. utilizing wireless communication and IoT technology. The APMD is designed to measure eight environmental parameters, including PM2.5, PM10, carbon monoxide, sulfur dioxide, nitrogen dioxide, ozone, temperature, and humidity. Equipped with a solar energy harvesting unit and rechargeable battery, the device utilizes GPS for geotagging and sends collected data to a cloud server via WiFi or NB-IoT connectivity. The primary focus is on the energy-efficient design of the PM sensor module, achieving up to 94% energy savings compared to traditional sensors. A power control mechanism, employing pulse width modulation (PWM), further enhances efficiency, saving 97% energy while maintaining low sensing errors for PM2.5 and PM10. Outdoor deployment

studies demonstrate that APMD is 90.8% more power-efficient than a reference setup, providing a significantly higher coverage range with acceptable sensing error. The system architecture comprises a microcontroller, radio module, sensor module, power module, and cloud platform. The power module integrates a solar panel, LiPo battery, and energy harvester, ensuring continuous operation during low light hours. The sensing module includes gas and PM sensors, while the communication module supports NB-IoT and WiFi. The APMD's ability to operate using solar power emphasizes its potential as an energy-sustainable pollution monitoring solution. The paper concludes by highlighting the contributions: the design of an energy-efficient multi-sensing module, the innovative on-board PM sensor, and a power control mechanism. The APMD proves 90.8% more power-efficient than reference setups and 98.3% more efficient than commercial devices, positioning it as a promising solution for widespread, sustainable air pollution monitoring.

In the paper titled "A Low-Power IoT Framework: From Sensors to the Cloud" a wireless sensor network for the Internet of Things (IoT), focusing on configurable nodes equipped with various sensors is designed [17]. These nodes collect environmental data, which is then transmitted to the cloud for universal accessibility. The proposed system enables users to analyze data related to temperature, humidity, motion, illuminance, CO gas, and air quality. The feasibility of the IoT transducer framework is validated through real-time hardware implementation. The system architecture involves plug-and-play sensors communicating through wires using I2C protocol within nodes and wirelessly to the hub via an ultra-low-power RF transceiver. The design emphasizes energy efficiency by incorporating wired communication between sensors and nodes and uniform packet transmission to the hub. The wired connection between the hub and the cloud ensures faster data transmission and detailed data processing occurs in the cloud for user accessibility. The node architecture comprises four modules: transducer, communication, data collection, and interface. These modules work together to collect, process, and transmit data to the regional hub. The regional hub, in turn, consists of interface, data collection, microcontroller, and optional super-node modules. The proposed system uses an Adafruit pro trinket 3V as the primary wireless node and a Raspberry Pi 2 single-board computer as the regional hub. The regional hub can act as a super-node for computationally intensive sensors. The paper also discusses power consumption, with a test revealing an average power consumption of 45 mW during operation at a 1 Hz sampling frequency. However, occasional peaks are attributed to

miscommunication between the hub and the node, causing the node to continue broadcasting information in the absence of acknowledgment, leading to increased power consumption.

Another proposed IoT sensing system [18] emphasizes energy efficiency, addressing power consumption through various design considerations. The wireless sensor node incorporates a sleep mode control to conserve operating power. The Microchip ATmega 1284p microcontroller, a central component of the sensor node, is designed for low-energy applications, with a current consumption of 0.4 mA in active mode and 0.6 μA in power-saving mode at 1 MHz, 1.8 V, and 25 °C. The real-time counter embedded in the MCU enables the device to transition from a wake-up mode during data transmission to a sleep mode, reducing power consumption during normal operations. Furthermore, the LoRaWAN communication module, RN2483, is selected for its low power characteristics. The instant transmission current consumption of the RN2483 is 44.5 mA, and the idle current consumption is 3.1 mA when connected to a 3.6 V power supply. These features contribute to the overall power efficiency of the sensor node, ensuring that energy is utilized judiciously. In terms of the energy harvesting methods, the solar panel serves as a sustainable power source, charging the 6000 mAh Lithium Polymer battery during sunny days. This stored energy is then utilized when solar energy is insufficient, such as during cloudy or rainy periods. The wireless energy harvester, operating with an input RF power level of -10 dBm, provides a steady energy output of up to 40 μW, and its consumption drops to less than 5 μW at ambient signal levels below -30 dBm. This dual-energy harvesting approach enhances the sensor node's autonomy by minimizing reliance on manual battery replacements, thereby reducing maintenance costs and improving the overall sustainability of the environmental monitoring system.

3.4 Predictive Model

Air quality monitoring and forecasting have become increasingly crucial in environmental science, driven by growing concerns over pollution and its effects on health and climate. The ability of machine or deep learning (M/DL) to successfully predict a wide range of phenomena globally emphasizes its significance for our project. Our focus on predicting the air quality index highlights the importance of thoroughly analyzing and reviewing diverse M/DL models. M/DL is a subset of Artificial Intelligence (AI); it has shown significant potential in interpreting complex environmental data including the prediction of air pollutants in the surroundings [20]. This review explores different M/DL or

in short, predictive learning models and to gauge the suitability of each for the proposed system.

3.4.1 Linear Models

3.4.1.1 Linear Regression

Linear Regression (LR) is a fundamental technique in predictive modeling for machine learning. It uses a linear equation to examine the relationship between two variables: an independent or explanatory variable and a dependent variable, with coefficients that adjust the scale and the intercept, which adds flexibility to the model's predictions. In its most basic form, the relationship can be represented by a simple linear equation shown below.

$$Y = mx + c \quad (1)$$

where Y is the dependent variable, m is the gradient of the linear function, x is the independent variable and c is the y-intercept

Linear Regression (LR) applies this concept by modeling the relationship between the dependent variable y and independent variable/s x. In this case, the independent variable serves as the predictor or input feature and the dependent variable represents the target feature or outcome of interest after modeling. LR models work by identifying the best-fitting line that minimizes the difference between both features. This line is represented by the equation:

$$Y = \beta_0 + \beta_1 X + \epsilon \quad (2)$$

where Y is the dependent variable, X is the independent variable, β_0 is the intercept, β_1 is the slope of the linear function and ϵ represents the error term, accounting for the variability in Y that X cannot explain

For instance, in the context of air quality prediction, the output variable is the concentration of pollutants such as $PM_{2.5}$, NO_2, or O_3, and the inputs include environmental conditions like temperature, humidity, as well as historical pollutant concentrations. LR assumes a direct proportional relationship between input variables and the output. This means changes in inputs are expected to result in predictable changes in the output [21].

One of the main strengths of LR is that the model's predictions are easy to interpret, making it a valuable tool for initial analysis and for contexts where explainability is crucial. Additionally, due to its straightforward nature, it requires less computational power compared to more complex models. However, its applicability in air quality (AQ) forecasting, which requires high computational complexity, is evidently limited. In a study conducted by Singh et al. [22], in Coimbatore city, the researchers applied an LR model to predict the AQI using data on pollutants such as $PM_{2.5}$ and O_3. Following a meticulous data preprocessing phase, the model was trained on 80% of the dataset and tested with the remaining 20%. To validate the model's assumptions, scatter plot matrices were examined. Figure 1 illustrates the linear relationships between these pollutants and AQI obtained from the study [22]. The plots suggested a normal distribution of variables, supporting the suitability of linear regression for basic AQI predictions.

Figure 4: Scatter Plot for Checking Linear Regression [22]

As illustrated in Figure 5, the relationships between the AQI and pollutants such as $PM_{2.5}$ and O_3 are depicted through individual scatter plots, with the linearity of these relationships suggesting that the LR model can predict AQI values but with limited accuracy. The diagonal kernel density plots within the same figure indicate the distribution of each variable, further supporting the assumption of normality in the regression analysis. The model's performance yielded a coefficient of determination (R^2) score of 50.8%, denoting that just over half of the variation in AQI could be accounted for by the model [22]. R^2 is a score that indicates how well the independent variable(s) explain the variability of the dependent variable. It's a way of measuring the strength of the relationship between the model and the observed outcomes, with a higher value denoting a 'better fit' between variables and higher accuracy of data representation by the model [23].

As stated above, the LR model for predicting AQI achieves an R^2 value of 50.8%, indicating a moderate model fit. While this suggests the model could account for approximately half of the variance in AQI, it also signals the necessity for more sophisticated approaches in certain cases. Given this context, Linear Support Vector Machines (LSVM) are considered a potent alternative. The strength of LSVM lies in its robustness and its ability to handle complex data scenarios that are linearly separable but may not fit the strict assumptions of LR [24]. LSVM and LR are fundamentally different in their purposes and methodologies. LR is used for predicting continuous outcomes and focuses on minimizing the error between the predicted and actual values. In contrast, LSVM is a classification technique that aims to separate different classes by maximizing the margin between their data points. While LR is sensitive to outliers and aims to fit the data closely, Linear SVM is less affected by outliers and prioritizes the best separation between classes, often at the expense of not fitting all data points tightly. This makes SVM particularly robust for classification tasks, in contrast to the predictive nature of linear regression for continuous data [25].

3.4.1.2 Linear SVR

SVMs are a class of supervised machine learning algorithms widely used for classification. On the other hand, the less known SVRs use the concept of SVMs but for the purposes of regression instead. In both, the model aims to find an optimal hyperplane that best fits the training data. In this context, the hyperplane refers to a decision boundary in the feature space that best represents a functional approximation of the relationship between the input features and the target variable. The optimality of this hyperplane is determined

between a balance of maximizing the margin, which is the distance between the hyperplane and the closest data points called support vectors, and minimizing the errors between the predicted and actual values. These support vectors are critical as they are the only data points that influence the positioning of the decision boundary, contributing to the model's efficiency and scalability [26]. In Figure 6, the Support Vector Regression(SVM) hyperplane is depicted as the solid line that efficiently separates the two classes of data points—blue for one class and green for another. The margin, represented by the space between the dashed lines, is the zone of the maximum distance from the hyperplane to the nearest points of both classes, which are termed support vectors, highlighted here by circles around the nearest points. These support vectors are pivotal as they define the hyperplane's position and orientation, ensuring the best possible separation between the data points.

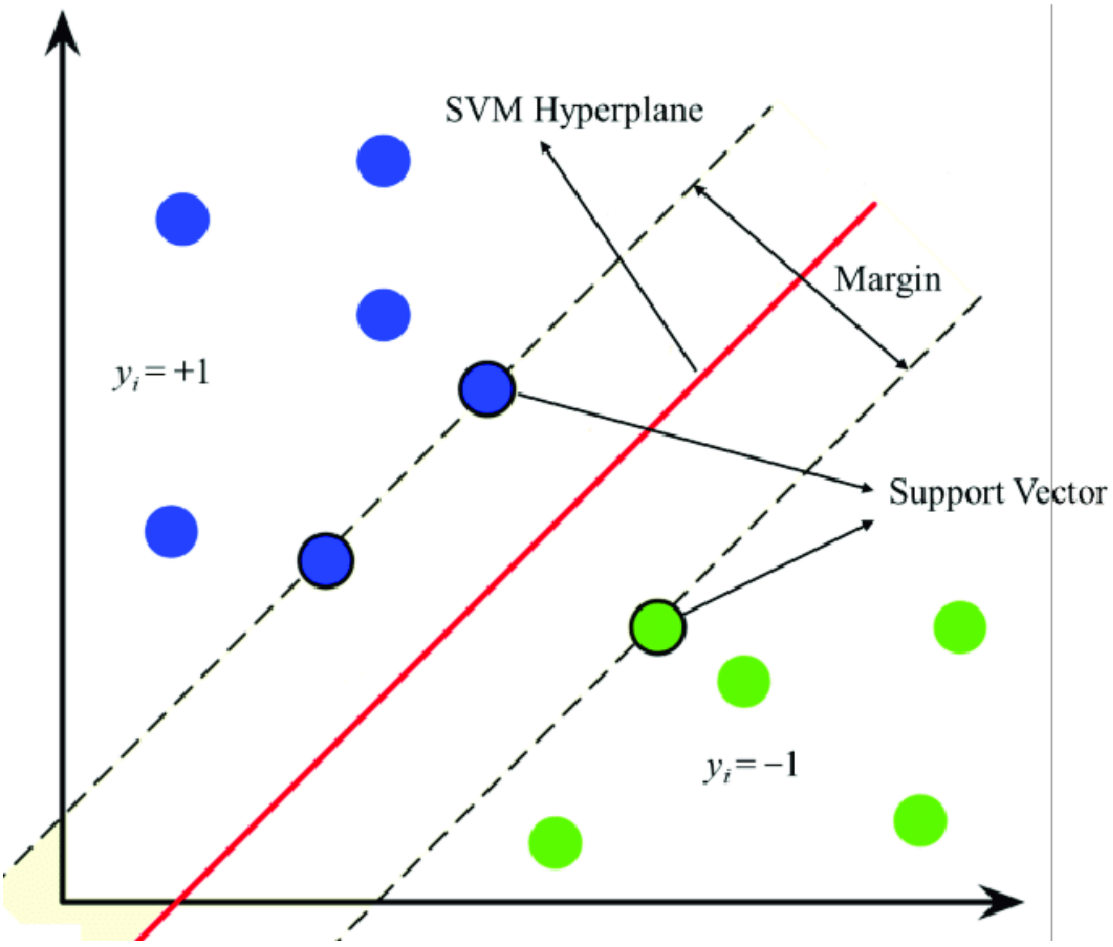


Figure 5: Hyperplane, Support Vector, and Margin [27]

SVR operates by transforming the feature space using a kernel function, which allows it to work in higher dimensional spaces where data may be more linearly separable or where complex non-linear relationships can be captured depending on the nature of the kernel

function used. A hyperplane is then constructed, equation shown below, based on support vectors, data points closest to a hypothetical optimal hyperplane. During inference, the SVR model makes predictions for new data points by evaluating their proximity to this optimal hyperplane using the learned model hyperparameters, including the combined weighted contribution of support vectors and the chosen kernel function.

$$\omega^T x + b = 0 \quad (3)$$

where ω is the weight vector perpendicular to the hyperplane, T represents a transpose function when in function with x, the input feature vector and b is the bias term or intercept

SVR excels in handling outliers by focusing on the support vectors found near the decision boundary. This method minimizes the influence of outliers, as SVR prioritizes the maximization of the margin between different classes. Consequently, SVR is less affected by data points that deviate from the overall pattern, making it robust for regression tasks in diverse datasets [27]. Linear SVR is especially effective for linearly separable data, where classes can be divided by a straight line. In the study conducted by Leong et al. [28], the prediction of the air pollution index was approached using a Support Vector Machine (SVR). This method includes three techniques with Linear SVR being one of them, alongside Polynomial and Radial Basis Function (RBF) kernels. They found that SVR with an RBF kernel performed the best among the three kernels and Linear SVR showed the least accurate prediction from the three. To decide which among the three kernel functions perform better they compared the values of Sum Squares Error (SSE), and R^2 values. The SSE is a measure of the total errors between predicted values from a model and the actual values observed, focusing on penalizing larger errors more severely. The lower the SSE value, the better the model's predictions match the observed data [29]. The linear kernel resulted in the highest SSE of 4198 and the lowest R^2 value of 0.9673, indicating lower prediction accuracy. In contrast, the RBF kernel achieved the best performance with the lowest SSE of 2008 and the highest R^2 of 0.9843, suggesting higher precision in prediction. The polynomial kernel displayed results comparable to the RBF kernel. Consequently, the RBF kernel function was identified as the most suitable for accurately predicting the API model. SVR with RBF and Polynomial kernels are types of Non-linear ML models. Their results suggested that the prediction of air quality is more accurate with non linear ML which shows that air quality data is non linearly separable [30]. Furthermore, a study conducted by Singh et al. [21], shows that while linear regression as mentioned above is a Linear ML model, has its uses in

some parameters of air quality prediction, it does not allow the consideration of complex and non-linearity in data. When the data is not linearly separable, linear models like linear regression or linear SVR may not be suitable without applying more complex techniques such as kernel methods to project the data into higher dimensions for separation [21].

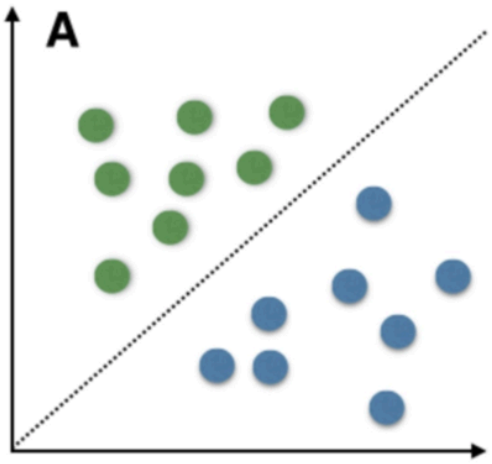


Figure 6a: Linearly Separable Data [30]

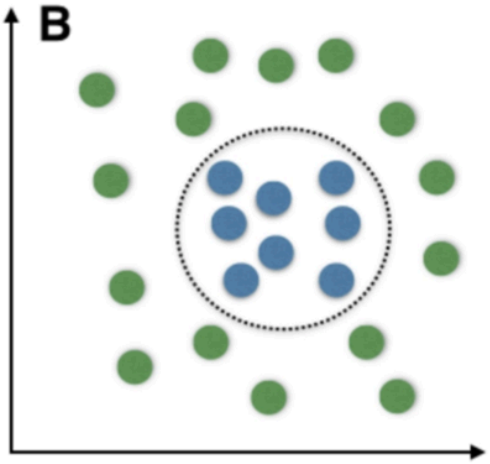


Figure 6b: Non-linearly Separable Data [30]

The images above illustrate two scenarios of data classification: Figure 7a illustrates linearly separable data, where a single straight line can effectively distinguish between the two classes of data points (depicted in green and blue) [30]. This is indicative of situations where a linear model, such as LR or LSVR, can perform adequately by establishing a clear boundary based on the linear relationship between variables. Figure 7b, on the other hand, shows non-linearly separable data, where the two classes are not separable by a straight line [30]. Instead, a non-linear boundary (represented by the dotted circle) is required to differentiate between the classes. This exemplifies the inherent limitation of linear models; they cannot capture the complexity present in such data structures. Non-linear models can accommodate these interactions by learning from the intricate patterns in the data, leading to more accurate predictions and classifications . Thus, while linear models serve as an excellent starting point for modeling simpler relationships, the evolution towards non-linear models is a necessary progression to address the complexities of real-world data.

3.4.1.3 Multiple Linear Regression

While Univariate or Single Linear Regression involves a single feature for the independent variable, MLR can involve two or more variables as input or independent variables, also called predictors. While the relationship between both variables is still

assumed to be linear, in MLR, multiple predictors can be considered to influence the dependent variable. The equation for MLR, shown below, is identical to the SLR equation in (3) but with the addition of multiple input variables and their corresponding coefficients.

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon \quad (4)$$

where Y is the dependent variable, x_1, x_2,..,x_n are the independent variables, β_0, β_1, β_2 ,...,β_n are the coefficients each feature with n representing the number of input variables and ϵ representing the error term

Compared to Univariate LR, Multivariate or Multiple LR (MLR) can capture the combined effects of multiple variables that have a potential impact on forecasts. This also makes them relatively more flexible since new features can easily be added to existing ones without having to make any substantive changes to the model architecture itself. Additionally, MLR can help identify the most influential factors affecting the target variable over a period of time. In a comparative study between univariate and multivariate (or multiple) linear regression by Iwok [31], both models were exhaustively tested against different types of datasets with varying numbers of input features. In the majority of them, particularly for those datasets with samples larger than 1000, MLR outperformed the univariate model on the MAE metric. Moreover, MLR computed outputs considerably faster in all tests, partly since the univariate model needed to run separately for each input feature while MLR needed to do it only once accounting for all features at once.

While MLR has shown definite results against SLR, there are also notable limitations, particularly when compared to the nature of the model itself. Although it takes into account the effect of various variables on the target, MLR strictly functions on the assumption that their relationship is linear. Although, this is likely not true with regards to gas emissions where factors affecting have exhibited randomness and non-linearity consistently [32]. MLR models also tend to be sensitive to outliers in the dataset [33]. Although, imputation is employed to convert null or zero values to data points closer to the trend of values around it, non-zero extreme values (possibly a result of errors in the measurements or actual random fluctuations in concentration values) can disproportionately influence the estimation of regression coefficients since the algorithm aims to minimize the differences in residuals.

Moreover, while theoretically there might not be a strict limit to the number of features that can be used as predictors, an excess amount can introduce notable complications.

Firstly, a high-dimensional feature space, characterized by a greater number of predictors relative to the number of observations or samples, increases the risk of overfitting in MLR. A common phenomenon in machine learning, overfitting occurs when the model wrongly captures noise or random fluctuations in the dataset rather than the true underlying patterns. As a consequence, the model fails to learn the meaningful relationships between features, degrading its predictive performance. [34]

Secondly, datasets with excess features increase the risk of making multicollinearity, or high correlation between the variables, more noticeable and prevalent. This is particularly severe for MLR models since multicollinearity can inflate the standard errors of regression coefficients {SEBn} which are a measure of the uncertainty associated with an estimated regression coefficient (Bn). When predictors are highly correlated, it becomes difficult for the model to distinguish the contributions of the different variables. Consequently, estimated regression coefficients become less precise, leading to a larger SEBx and distorting the interpretation of the model for further forecasting.[35]

3.4.2 Non-Linear Models

3.4.2.1 Non-Linear SVR

For regression tasks, input data is mapped to a high-dimensional, non-linear space using a kernel function, which facilitates the identification of a hyperplane. With regards to forecasting, polynomial (Poly) and radial basis function (RBF) are the most well-known and best-used kernel functions in SVR.

In general, non-linear SVM models outmatch their linear counterparts. A 2020 study by Leong et al. [36] compared an SVM model with a linear kernel and another with RBF kernel. The RBF model noticeably outperformed the linear model with a higher R^2 (0.9843 to 0.9673) and a lower MSSE value (1.444 to 3.018). However, both SVM models compared poorly with deep learning models where Artificial Neural Network (ANN) models saw an R^2 value of 0.99, in comparison.

3.4.2.2 Neural Networks

Neural Network (NN) methods, also known as Artificial Neural Networks (ANN), were originally developed in the late 1950s, inspired by the McCulloch-Pitts simplified model of a biological neuron [37]. In ANNs, these neurons consist of interconnected processing elements or nodes, organized into layers. These layers typically include an input and an output layer sandwiching various hidden layers. Pre-processed data, including text and images, is passed through this input layer in the form of vectors and successively through the hidden layers before producing an output in the final layer. This entire process is facilitated through mapping functions referred to as "activation functions" which are applied to the input variable through weight and, occasionally, offset parameters [38]. These weights are quantified through a training phase that usually includes additional algorithms supplementing the model. With an operation based on parallel processing, they are typically characterized as computational models with particular properties including the ability to adapt and learn, to generalize, or to cluster and organize data [38].

Numerous studies illustrate the effectiveness of NN models in achieving relatively accurate forecasting. Maleki et al. [39] used ANN algorithms to predict hourly air pollutant concentrations and two air quality indices, air quality index (AQI) and air quality health index (AQHI) in the city of Ahvaz, Iran over a period of one year (2009-2010). The applied algorithm involved nine factors in the input phase (five meteorological patterns and pollutant concentrations 3 and 6 hours in advance), 30 neurons in the hidden layer, and one output at the final stage. For the six criteria pollutants tested (O_3, NO_2, PM_{10}, $PM_{2.5}$, SO_2, and CO) across four sites, the correlation coefficient, R, between ANN predicted and physically measured values was 0.87, indicating a strong correlation and therefore a strong forecast capability. Another study by Cordova et al. [40], implemented two NN techniques to predict hourly PM_{10} values based on past values and three meteorological variables from five monitoring stations. The two models showed good forecasting performance with a relatively high Spearman score of over 0.6 (with 1.0 indicating strong positive monotonic relation) when compared to the actual data from the stations. Although, performance notably diminished for stations subject to unpredictable external sources of pollution or due to short-term changes in climate such as in heavy industrial areas.

However, ANN exhibits varied results when compared to other models. Cakir and Sita [41], compared the predictive efficacy of ANN models in contrast to Multiple Linear Regression (MLR) methods. MLR are a subtype of linear regression models that make use of multiple independent variables to predict outcome of the dependent variable [42]. Previous day's pollutant concentration, atmospheric pressure, wind speed, relative humidity and temperature data acted as input or independent variables while observed pollutant concentrations for each pollutant (PM_{10}, NO_2, and O_3) were chosen as the output or dependent variables. While ANN performed satisfactorily in predicting NO_2 and O_3 values, it underperformed in PM_{10} forecasting and only managed to exceed its initial performance with the aid of additional algorithms. Overall, however, ANN failed to produce markedly superior results in contrast to MLR. Although, it is noteworthy that the authors point to another study, which will be explored in subsequent sections, reporting contrary findings for a particular type of ANN, referred to as RNN, or Recurrent Neural Networks which they note hinted at a better performance.

Another main challenge in developing an NN model is addressing the problem of overfitting. Overfitting occurs when a model fits to the noise in the data and so will not generalize well to new datasets. As illustrated in figure 8, An overfitted model could fit the data very well during training, but produce poor forecast results during testing therefore suffering a lack of generalization [43].

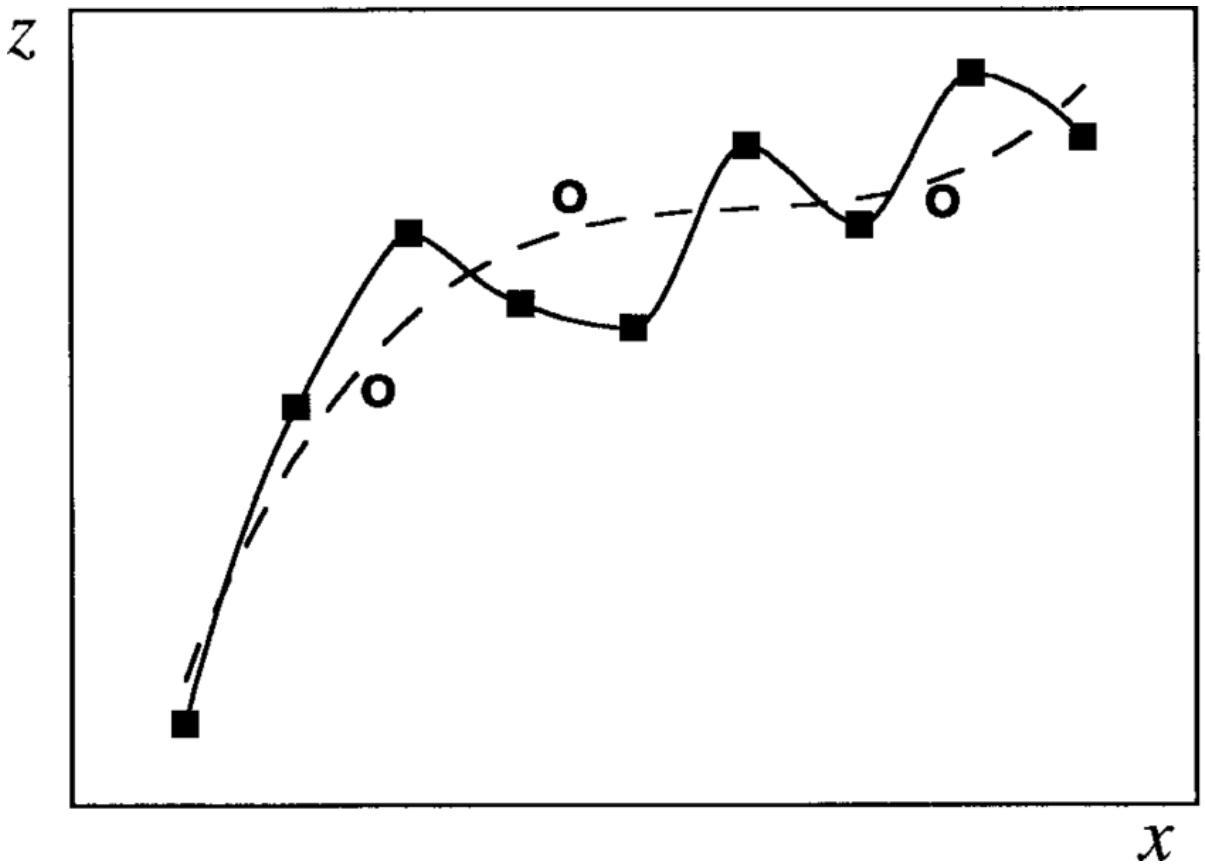


Figure 7. A diagram illustrating the problem with overfitting. The dashed curve shows a good fit to noisy data (squares), while the solid curve illustrates overfitting, where the fit is perfect on the training data (squares), but is poor on the test data (circles) [26]

However, typically regularization methods are used to correct the overfitting issue. Regularization involves imposing a penalty on the weight parameters of the model, however, reducing model freedom. Chu et al. [44] recommend another solution to maintain both model freedom and performance. They confine the overfitting of ANN by combining the model with another ML method, k-Nearest Neighbour (kNN). The kNN predicts the target value from the values of the nearest *k* points according to special operational rules. Although, this limits its prediction capabilities to make accurate forecasts for datasets exceeding the range of its training data. Therefore, the authors suggest combining kNN methods to specific hidden layers and mapping functions of the ANN models to avoid the overfitting. This combined method, they call AKC or ANN-kNN combination, resulted in a markedly lower average absolute relative deviation for prediction at 2.5%, better than either ANN (5.9%) and kNN (19.2%), on their own.

There are different types of ANN models which add their own enhancements to the initial models proposed.

3.4.2.3.1 Feed-Forward Neural Networks

Feed-Forward neural networks are the simplest and earliest type of ANN where the input travels only in a unidirectional manner from the input to the output node. In its most basic form, this type of NN is known as a Single-Layer Perceptron (SLP) wherein the input data is multiplied by weights and then summed together. If this sum is greater than a predetermined threshold, the output value is 1 indicating a positive outcome, otherwise -1 indicating a negative outcome. Although this makes the single-layer perceptron the most suitable for binary classification systems, its application becomes limited for predicting complex and dynamic datasets such as those in air quality monitoring which involve various unpredictable, non-linear factors [45].

Although limited in scope, feed forward exhibits some decent results, albeit it is contingent on increasing architectural complexity. For instance, Sahin et al. [46] use a four-layered feed forward NN model to determine the effect of NO_x and SO_2 concentrations on PM_{10} values. Consequently, NO_x and SO_2 were used as input parameters along with temperature, air humidity, pressure, and wind speed. Parameters were obtained from daily public sources over a span of 5 months from 2018-9, a relatively shorter time frame compared to some of the other papers discussed in this review. Root Mean Square Error (RMSE) and coefficient of determination (R^2) were used as performance metrics. RMSE and R^2 were found to be 27.89 and 0.881, respectively. These results were relatively satisfactory despite low data, however, increased architectural complexity in turn led to increased computational cost and decreased efficiency.

3.4.2.3.2 Multi-Layer Perceptron

In contrast, Multi-Layer Perceptrons (MLP), offer a more advanced and versatile approach by incorporating additional hidden layers and functions into the SLP model. Abdulla et al. [47] trained two MLP models analyzing daily PM10 concentration datasets from 2010 to 2014. Utilizing Tansig-Purelin activation functions which require 13 neurons per layer, the models explained up to 69% and 80% of the observed data variance during the training and testing phases of the study, respectively. Consequently, the authors recommend the use of this model to predict early excesses of PM10 in the air. Another study by Rahimi

[48] used MLP models to predict NO2 and NOx in the atmosphere. The optimum number of neurons in each hidden layer was determined by obtaining the minimum RMSE (root mean square error) for each testset. Rahimi concluded that MLP was a useful model for short-term predictions of NO2 and was generally superior over traditional linear models with lower RMSE and higher R^2 values. This is corroborated by older studies [49, 50] that recommend the use of Back-Propagation (BP) learning algorithms to fine-tune the weights of successive iterations based on the perceived error rates from previous iterations, greatly reducing overall error rates.

Additionally, MLP has shown to perform consistently better than regression models in general. For instance, Chaloulakou et al. [51] compared MLP with an MLR model to forecast PM_{10} values in Athens. Based on a data inventory for a fixed two year period, meteorological variables such as wind speed and relative humidity and other pollutant concentrations including NO_X and SO_2 were used as input parameters. Figure 9 illustrates the architecture of their MLP model, consisting of three hidden layers as well as the input and output layers.

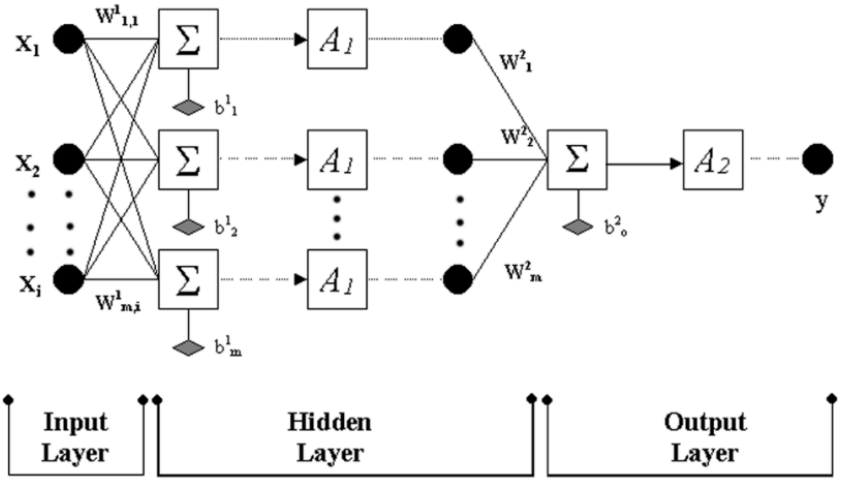


Figure 8: Architecture of the three-layered MLP method employed by [Chaloulakou] of i input variables and m hidden processing units and their corresponding weights

During the model's training phase, Levenberg-Marquardt (LM) algorithms were used instead of the conventionally popular BP algorithms employed by [48, 49, 50]. LM showed significantly faster convergence to the output than BP and was able to find comparatively better local error minima [52]. Post-training, calculated RMSE values indicated a better performance for MLP than the MLR models. MLP in both test runs of the models produced

an average of 19.1 while MLR produced a mean of 20.9. The difference in their R^2 values too was noteworthy with a difference of 13% between both models that showed MLP outmatching MLR.

Nevertheless, a number of challenges hamper the effectiveness of MLP methods. Conventional MLPs are generally trained under the assumption that network weights and offsets remain constant even after training [53]. However, this time-invariant assumption has proven increasingly inadequate for modern urban conglomerations. Urban environments are dynamic, influenced by factors including rising energy consumption and climate change. These factors, among others, affect pollutant emissions and airflow patterns, in turn leading to unpredictable and rapid changes to air quality values and factors affecting AQI at a particular time. Studies carried out in Central London using a conventional MLP model, for instance, showed its R^2 values failed to reach the 50% mark in many of its test runs [54]. Additionally, emerging ML models, including Deep Learning (DL), can obtain consistently accurate results with less bias and support of training algorithms like BP and LM. Deep learning models will be discussed in the upcoming sub-sections.

3.4.2.3.3 Recurrent Neural Networks

Recurrent neural networks (RNN), one of many deep learning approaches, is an ANN model with internal loops. These loops function by saving the output of a layer and feeding this back to the input to help in predicting a more accurate outcome for the same layer. The initial layer resembles SLPs with no back-propagation. However, consecutive neurons will retain some information from the previous time-step, allowing each neuron to make small changes to reduce the overall error rate during the backpropagation. Figure 10 illustrates a representation of a fully-connected RNN. The initial propagation of this particular representation behaves akin to an MLP with internal looping depicted through virtual “context” layers which spatially depict the same node but from the previous time-step [55].

.

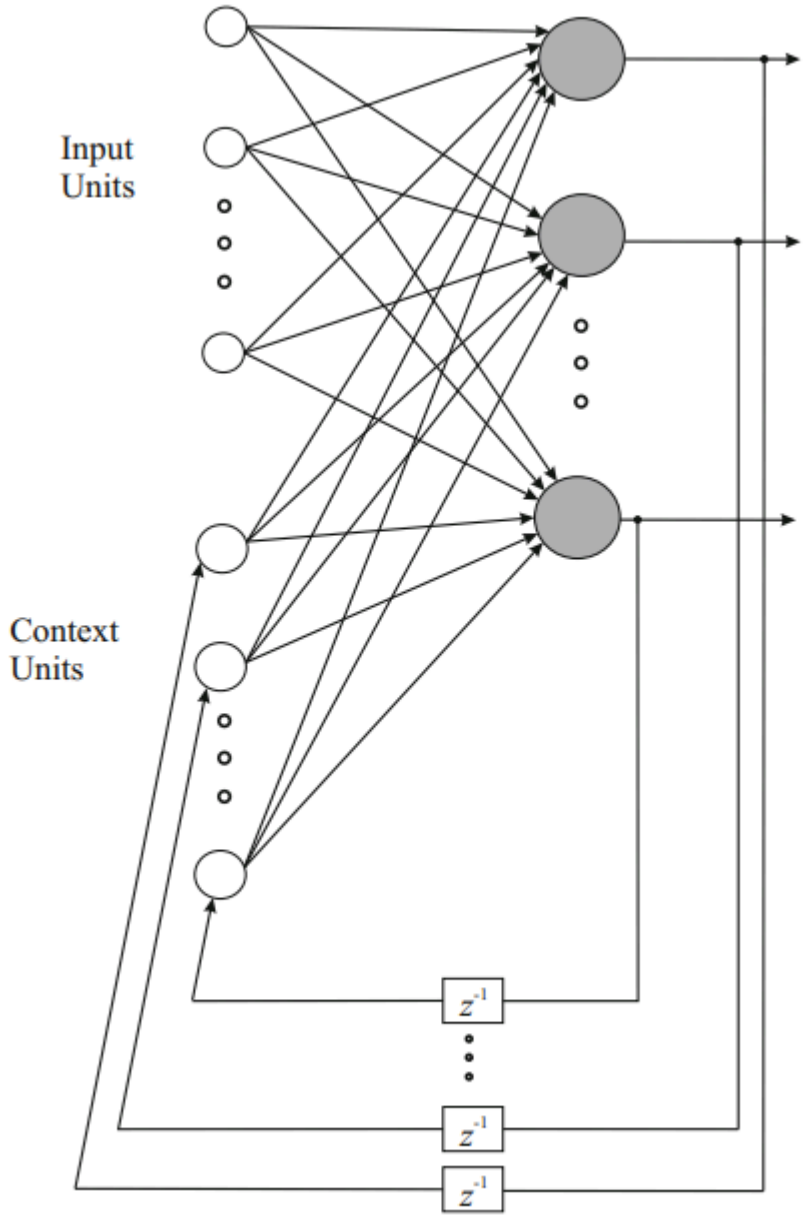


Figure 9: A simplified illustration of a fully-connected RNN with context layers

The most popular type of RNN architecture is the Long Short-Term Memory (LSTM) model. While simple RNN models are able to persist some information from the previous time-steps, it is unable to hold onto that information for long enough if the gap between the previous node and the current task is too large. In such cases, LSTM becomes useful. The critical section of an LSTM-RNN unit is the cell state which stores the information held by the unit, with information movement regulated by structures known as gates. The particular functionality of each gate is decided by separate weight parameters which are learned over time and are assigned by the model itself. Thus, it allows LSTM-RNN models to effectively

capture long-term temporal dependencies in their cell states and overcome issues related to vanishing gradient or slipping memory faced by more conventional RNN models. Since air quality forecasting requires data that is a sequence with long-range dependencies, LSTM-RNN could be of significant practical importance to our project [56].

Belavadi et al [57] propose using LSTM-RNN for air quality forecasting as part of their project in Bangalore, India. They monitor and gather real-time air pollutant concentration data from two sources. The first from their own wireless sensor network that gathers and sends data from sensor nodes placed across the city to a server that aggregates the data. The second is the data available vis-a-vis the Indian government's air quality database [58] which was used to form the historical datasets to train the LSTM models. The dataset included pollutants such as CO, NO_X, and $PM_{2.5}$. Features of interest in their work included outlier removal using z-score, shown in equation x, and an imputation process to replace the removed outliers with non-outliers from previous datasets.

$$z_i = \frac{(x_i - \mu)}{\sigma} \tag{5}$$

where μ is the mean and σ is the standard deviation of the aggregated dataset

Wang et al. [59] proposed an LSTM model that used chi-square test (CT) to determine the influencing factors of air quality in a time series forecasting. Hourly air quality and meteorological data spanning a period of two years was used to train this model. To gauge their model's performance, it was compared with five methods including MLP, Simple RNN, a BP-supplemented NN model, and a Support Vector Regression (SVR) model, a regression model that combines SVM functionalities. Under the same computer operating environment, the same training set and appropriate training parameters, the RMSE values were calculated for all the models. The proposed LSTM model remained consistently superior over the other models with the lowest RMSE values of the model set as illustrated in figure 11. Additionally, the proposed model also topped the model set in prediction accuracy with an accuracy rate of 93.7%, higher than the 90.7% for the MLP model.

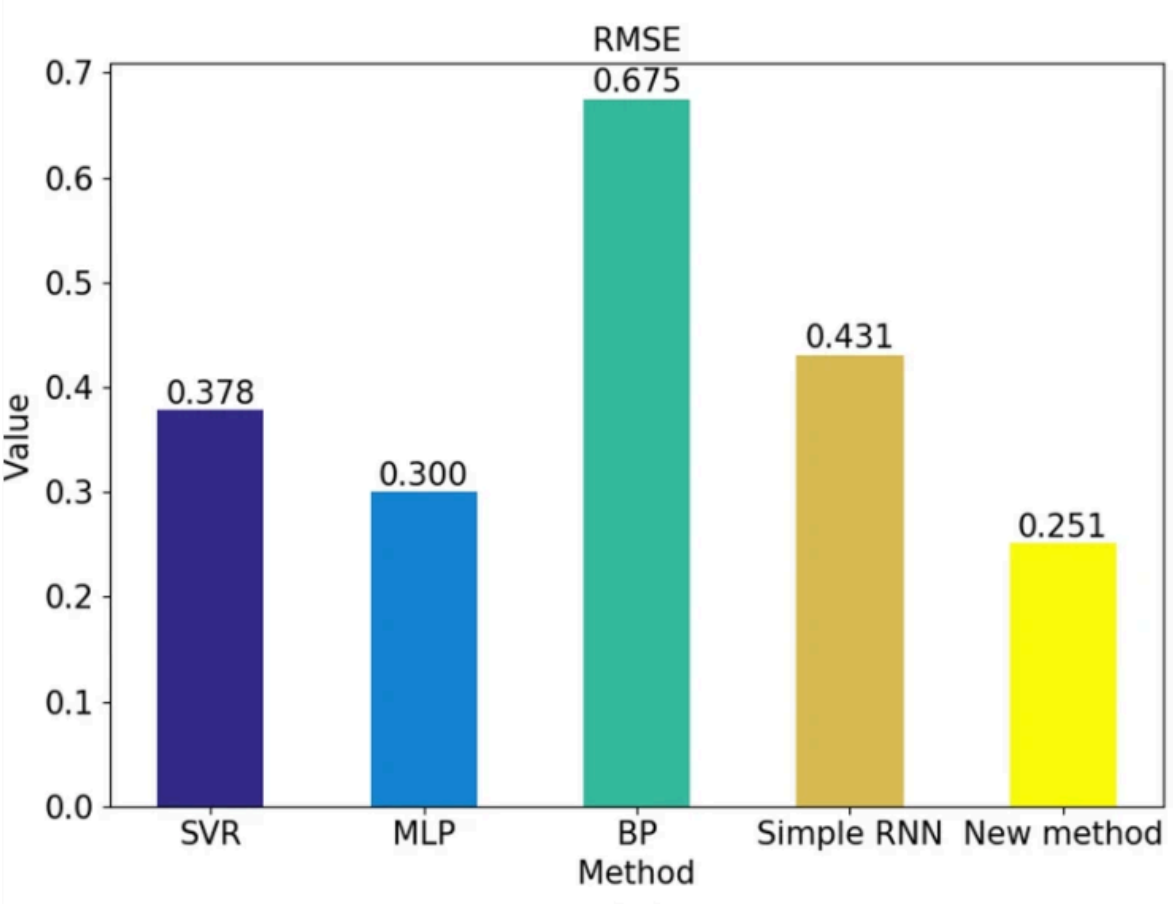


Figure 10: Bar graph depicting RMSE values for five ML models from []. The "New Method" refers to the proposed CT-LSTM model.

3.4.2.3.4 Convolutional Neural Networks

Originally designed to process image data, Convolutional Neural Networks (CNN) - or *convnets* for short- is a deep NN structure with multiple hidden layers. Whereas there are several layers employed in CNN, the three most fundamental are convolution layer, pooling layer, and fully connected layers. These layers are stacked to form a full convolutional layer. The convolutional layer employs artificial neurons representing convolutional filters, which create feature maps by splitting the input into smaller blocks and convolving them with specific weights. This process generates different sets of features as the filters slide over the input with the same weights. The pooling layer, on the other hand, serves to reduce the spatial size of the input representation and the number of parameters. It identifies similar information within a local region and outputs the dominant response. This layer is particularly advantageous and sets CNN apart from familiar models since it reduces the risks of overfitting and increases computational power with reduction in data size. Finally, the fully

connected layers are attached on top of the CNN, flattening the output from the pooling layer to a singular vector [60].

The essential differentiation between CNN and other conventional NN models is that CNN architecture makes the implicit assumption that the inputs are image-like. This assumption proves advantageous in the AQ domain, where the distribution of pollutants across geographical locations forms intricate spatial patterns akin to images [61]. By leveraging this assumption, CNNs can effectively capture and analyze spatial dependencies within the data, offering a significant advantage over other ML models. Furthermore, convolutions can capture translation invariance [60]. This feature allows the filters within the CNN's hidden layers to remain unaffected by the spatial location of features in the input. Since pollution can vary across different locations/nodes and times, the ability of CNN to recognize patterns irrespective of specific spatial locations further enhances their suitability for air quality forecasting.

However, a major limitation to the applicability of CNN in AQ forecasting is that there is a considerable dearth of studies that deal with comprehensive datasets of sufficiently long time intervals. Additionally, there is still no clarity what type of network architecture is best suited for extracting the spatiotemporal features of evolving and chaotic pollutant datasets. Chauhan et al. [62] attempt to solve this, suggesting that CNN, given its use in processing images and videos, can be used for AQ monitoring. In their study, they utilized datasets of air quality at two different time intervals (hourly and daily). AQI was calculated from PM_{10}, SO_2, NH_3, $PM_{2.5}$ and CO values sourced from publicly available data. The overall study is discussed in two phases where the first phase focuses on preprocessing and data analysis, whereas the second phase is used for the purpose of testing the model accuracy as well as how well the data has been classified accurately. The overall model was implemented using Python programming and scripting language. Calculated R^2 only reached a decent 70%, lower than other NN models in the same domain. However, results drastically improved when CNN was paired with a LR model indicating the need for a specific spatiotemporal response to AQ forecasting.

3.4.2.3.5 Hybrid Model: CNN + LSTM

While LSTM models are widely employed in many deep learning-based methods for making predictions, they are particularly used in time-series data processing. They excel in

handling the temporal aspect of pollutant data, i.e. the data that changes with time. CNN has been applied to predict urban pollutant concentrations, often by analyzing satellite images. However, occasionally, rather than image data only abstracted monitoring data is available, such as wind direction, temperature, and location. As discussed LSTMs work well with the temporal aspect of data, it's important to note that changes in pollutant levels are not only dependent on time but also exhibit spatial correlations. Pollutants originating from one location can disperse to nearby areas, which requires an understanding of spatial information. Therefore, incorporating the distinct but complementary spatial aspects of CNN and temporal aspects of LSTM could provide for the ideal spatiotemporal response required in the most accurate air quality forecasting [63].

In one study [64], a combination of CNN and LSTM models was employed to predict urban PM2.5 concentration. The CNN, known for its spatial feature extraction abilities, captured spatial relationships among monitoring stations and quantified the magnitude of spatial effects during air pollutant diffusion. The output from the CNN was then fed into the LSTM. LSTM excels in predicting future values based on past inputs, mitigating challenges associated with gradient issues. This combined approach aimed to leverage the strengths of both models: CNN for spatial features and LSTM for temporal correlations. The prediction model integrated meteorological factors and pollutant concentration from previous time periods, which were processed through the CNN for feature extraction. The LSTM layer further refined the predictions, and a fully connected layer decoded the LSTM output to yield the final PM2.5 concentration forecasts. Using RMSE as a metric to gauge accuracy, the CNN+LSTM method outperformed the lone RNN, CNN, and LSTM models, achieving the best performance. The CNN-based method significantly improved pollutant concentration prediction correlation compared to RNN. The CNN-alone method exhibited a higher correlation than the proposed method, indicating their ability to predict the trend of air pollutant concentration effectively. However, the CNN-alone method had higher RMSE values, suggesting limitations in handling long-term sequence prediction. Comparing the LSTM-alone method with the proposed model revealed the superior prediction performance of CNN+LSTM. This underscores the improvement achieved by incorporating CNN for spatial features and enhancing the prediction performance of air pollutant spatiotemporal data. The CNN, RNN, and LSTM models were found to be unsuitable for spatiotemporal sequence prediction due to poor prediction accuracy over time. The proposed model exhibited superior prediction accuracy, as evidenced by the final RMSE and correlation

values. The CNN served as the foundation, extracting spatial features from air pollutant data, while the LSTM addressed the time-dependent aspects. The hybrid model offered notable advantages, including the compression of input data to eliminate redundancy and identify actual features using CNN, along with the reduction of model complexity through shared weights. LSTM effectively handled time series dependencies by fine-tuning both trained CNN and untrained LSTM components, incorporating regularization to prevent overfitting. This model proved suitable for processing data from multiple monitoring sites within a single city, capturing the interaction between sites and temporal variations in air pollutants for enhanced predictive accuracy [64].

3.5 Cloud System

3.5.1 Cloud Based Weather Station using IoT Devices

Palak et al. [65] presents a smart cloud-based weather station system using Raspberry Pi and various sensors. The system aims to monitor and predict weather conditions such as temperature, humidity, wind speed, pressure, and rainfall in a cost-effective and low-maintenance manner. It involves collecting real-time weather data from multiple sensors, transmitting this data to a cloud database for storage and analysis, and deploying machine learning models in the cloud for weather predictions. Users can access weather data and insights in real-time through a web application. The paper discusses the motivation behind the project, highlighting the limitations of existing professional weather stations in terms of cost, customizability, and accessibility. It emphasizes the need for a modular, cost-efficient, and compact weather monitoring solution that leverages the latest technologies.

The proposed system consists of Raspberry Pi 3 as the central unit and multiple Raspberry Pi Zero W boards connected to sensors. The data collected by these sensors is sent to the central Raspberry Pi 3 for processing and subsequent transmission to a cloud-based database. The use of Amazon IoT services and AWS IoT Analytics for data processing and analysis is discussed, and the system's architecture is outlined. The paper also addresses the characteristics of the system, including accuracy, availability, scalability, reliability, portability, and modularity. It describes the system's components, which include Raspberry Pi boards, various sensors, jumper wires, a power source, and optional components like a breadboard. Finally, the paper provides results in the form of insights and data visualizations obtained from Amazon Quicksight, showcasing trends in temperature and pressure. The system's prediction model is highlighted, along with the real-time weather data accessible

through the web application. In conclusion, this paper presents a comprehensive solution for a cloud-based weather station using Raspberry Pi and IoT technologies. It addresses the limitations of traditional weather monitoring systems and provides a cost-effective, modular, and easily accessible alternative for weather data collection and analysis.

3.6 Dashboard System

3.6.1 Web Dashboard Development for Cloud ServerBased Air Quality Monitoring System

Via et al. [66] considers the development of an air quality monitoring system that integrates IoT, cloud computing, and web dashboard technology. The focus is on addressing the prevalent air pollution issue in Indonesia, providing a comprehensive insight into the system's design, theory, and performance. The introduction discusses the severity of air pollution in Indonesia and its detrimental effects on human health. It emphasizes the need for a monitoring system to assess air quality and mitigate the adverse health impact. The review discusses the five major air pollutants identified by the WHO, shedding light on the importance of monitoring air quality to provide essential information for both government and public use. The theoretical background of the project discusses key elements. The web dashboard, essential for providing comprehensive real-time data, covering its frontend and backend, technologies used, and the significance of effective information display. The air quality index (AQI) and its significance in evaluating air pollution are also detailed, especially the Indonesian ISPU standardization used to categorize air quality and provide real-time monitoring. The system design and implementation provides a detailed technical overview, starting from hardware to software systems. It describes the sensors, data transmission, and the development of the web dashboard. It outlines the backend and frontend design, their functionalities, and architecture, as well as the testing processes for capacity, response time, and overall functionality. In conclusion, this paper includes a comprehensive guide to the development of an air quality monitoring system. It offers an extensive review of the technological and theoretical details involved in such a system.

3.6 Conclusion

In summary, our literature review encompasses a comprehensive overview of our project covering topics that will be of importance during the construction of our project. By integrating innovative hardware components, advanced machine learning models, and

cloud-based technologies, our solution aims to provide a user-friendly platform for individuals to make informed decisions about their surroundings, especially in relation to the levels of five crucial target gasses: O_3, NO_2, CO, and $PM_{2.5/10}$.

The chosen ESP-WROOM-32 stands out as a key hardware component due to its low power consumption and relatively low cost, making it an ideal choice for sustainable and energy-efficient operation. The sensors employed in the system play a pivotal role in measuring the concentrations of the specified gasses, facilitating the generation of reliable data for further analysis. Machine learning, a cornerstone of our project, introduces a multi-faceted approach through Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) models. The CNN ensures high accuracy in gas concentration detection, leveraging its ability to effectively learn spatial dependencies within the sensor data. Meanwhile, the LSTM model specializes in time series analysis and temporal correlation, enabling our system to make predictions regarding future air quality conditions. This combination of models enhances the predictive capabilities of our solution, providing users with foresight into potential air quality fluctuations. The integration of a cloud-based infrastructure facilitates seamless data transfer between the ESP-WROOM-32 and a centralized server, enabling real-time monitoring and analysis. This cloud architecture not only ensures efficient communication but also serves as a robust storage and processing hub for the vast amounts of data generated by the sensors.

Emphasizing the significance of Air Quality Index (AQI), our project addresses a critical need in contemporary urban environments. The AQI serves as a vital metric for gauging the overall air quality and potential health risks associated with exposure to specific gases. By providing users, including those with health conditions, with accessible and accurate AQI information through our website, we empower them to make informed decisions about their daily activities, such as choosing safe routes for outdoor walks. Ultimately, our AIoT-based system strives to contribute to healthier living environments, promoting individual well-being and fostering a community-centric approach to air quality management.

4. System Specification

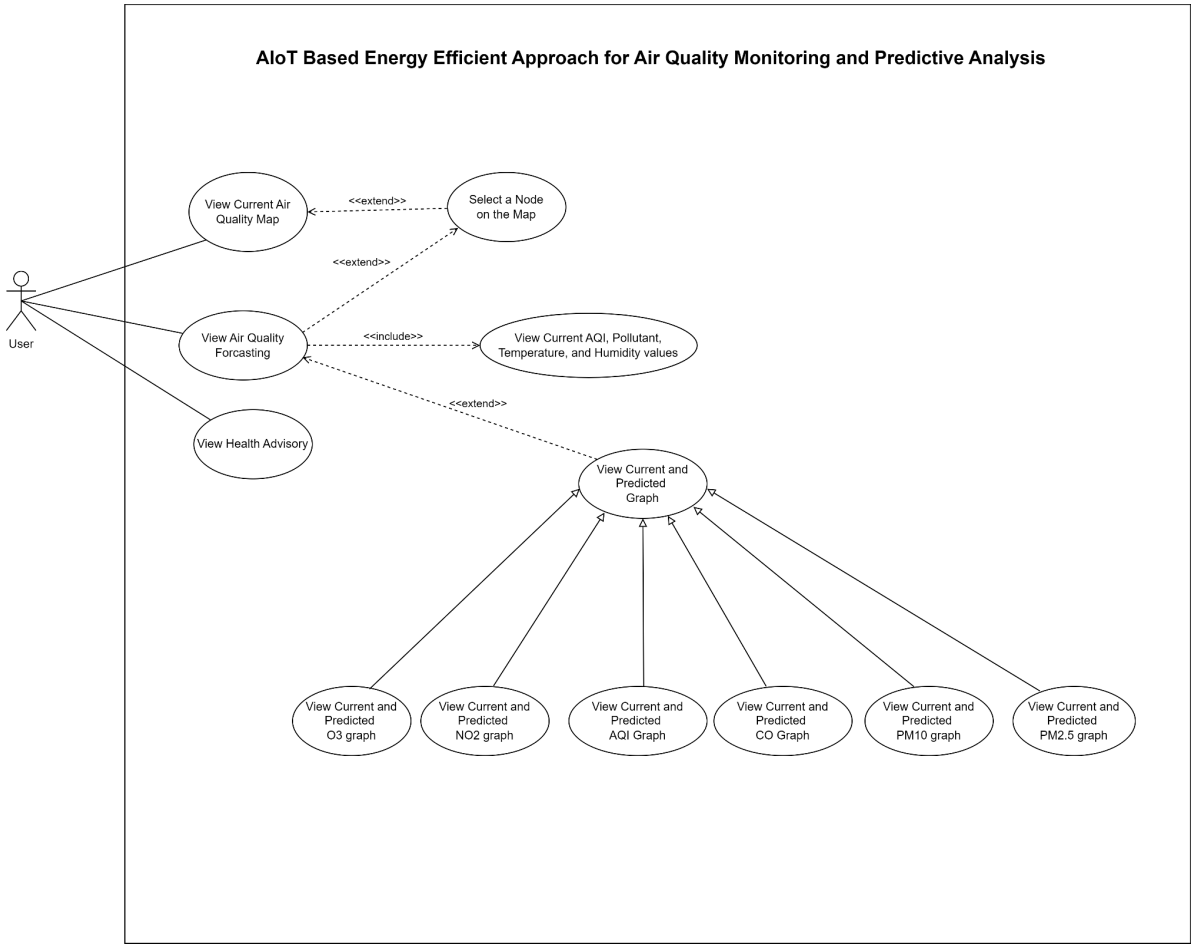


Figure 11. User Case Diagram

4.1 Functional Requirements:

1. Inputs:

1.1. The user shall be able to view current AQI readings, pollutant sub-indexes, temperature, and humidity levels.

1.2 The user shall be able to view predicted AQI readings and pollutant sub-indexes.

1.3 The user shall be able to view current and predicted air quality details based on the node they select.

2. Outputs:

2.1. The website must display the air quality index (AQI) of the nodes on a map.

2.2. The website must provide individual data about the specific pollutants contributing to the AQI, such as $PM_{2.5}$, PM_{10}, ozone, carbon monoxide, and nitrogen dioxide.

2.3. The website shall display general weather-related information including temperature and humidity.

2.4. The website must provide area-specific AQI data.

2.5. The website must display AQI forecast data and current AQI

2.6. For all data displayed, the website shall allow data visualization in the form of graphs for the user's convenience and ease of use

2.7. The website shall provide health advisories based on the current AQI value of the selected node.

3. **Computations:**

3.1. The website shall show present and future AQI trends through an interactive bar graph

4.2 Nonfunctional requirements:

1. **Performance:**

1.1. Balance effectiveness and efficiency by using a low powered microcontroller like the ESP32 microcontroller with low-power features that enable energy-efficient operation, ensuring high effectiveness in air quality monitoring while conserving power.

1.2. Select affordable yet precise and secure sensors, microcontrollers, dashboard, and cloud-based services. Each node shall be under 1000 AED including miscellaneous costs.

2. **Quality:**

2.1. Ensure the operation reliability by the microcontroller's robust data processing. Combined with high-quality sensors, it guarantees consistent, accurate air quality data.

2.2. The system will consistently operate the AQI based data throughout the day and update on the dashboard and user interface

2.3. Streamlining of design which reduces the need for frequent upkeep. High-quality sensors minimize sensor drift and replacements, extending system life and lowering maintenance costs.

2.4. Swift response of the system due to the microcontroller and efficient communication protocols, providing real-time air quality data for quick decision-making during environmental events.

3. Safety:

3.1. Use robust, weatherproof enclosures or casings to shield the system's components from rain, dust, and physical impacts. Ensure that these enclosures are rated for the specific environmental conditions in which the system will be deployed.

3.2. Ensure proper ventilation and airflow within the enclosures to dissipate excess heat, preventing overheating in warm conditions.

3.3. Prior to deployment, subject the system to environmental testing to ensure it can withstand the expected conditions. This may involve testing for waterproofing, dust resistance, and temperature tolerance.

3.4. Establish a maintenance schedule to inspect and clean the system and its protective enclosures, addressing wear and tear, damage, or environmental debris that may accumulate over time.

3.5. Choose electronic components that comply with safety standards. Look for components that are certified and tested for electrical safety, such as power supplies, connectors, and sensors.

3.6. Ensure that all wiring is properly installed and insulated. Conduct routine inspections to identify exposed wires, damaged insulation, or wires in close proximity,

and address these issues promptly. Regularly test and maintain electrical connections to detect and rectify any loose or frayed wires that could lead to short circuits.

3.7. Implement voltage regulation and control mechanisms to prevent overcharging of solar powered batteries or voltage spikes in the system.

4. Interface:

1. Graphical User Interface (GUI):

1.1. The GUI should be intuitive and user-friendly to ensure that users of varying technical backgrounds can operate it effectively.

1.2. The GUI should respond promptly to user inputs and display real-time data without significant delays.

2. Hardware/Software Interface:

2.1. The system should be compatible with a range of hardware components, including sensors, communication modules, and display devices.

2.2. The interface between hardware and software should be reliable to prevent data corruption or system failures.

2.3. Ensure that data from hardware sensors is accurately and reliably processed and integrated into the system software.

5. Technical Approach and Design Alternatives

5.1 Problem Statement and Proposed Solution

Air pollution is a major concern in the United Arab Emirates (UAE) and worldwide, with severe implications for public health and the environment. According to the World Health Organization (WHO) in 2022, air pollution is recognized as one of the greatest environmental risks to health [1]. It has been classified as a leading cause of death and disability worldwide due to heart disease, stroke, COPD, cancer, and pneumonia [2]. Moreover, outdoor air pollution is a major environmental health problem affecting people in low-, middle-, and high-income countries [1]. In fact in the UAE, poor outdoor air quality stands as the top cause of environmental-linked deaths [3]. It also affects adolescents and is

associated with industrial proximity and geography as predictors of asthma. At least 13% of Emirati children in the UAE suffer from asthma, due to indoor and outdoor environmental factors [2]. Although there are numerous studies about indoor air quality monitoring in the UAE, there is a dearth of research focused on outdoor air quality monitoring systems. The UAE government is consistently expressing its strong commitment to improving air quality, as outlined in the UAE Vision 2021 [2]. More importantly, recently, the UAE Air Quality Agenda 2031 has been built upon three pillars: Monitoring, Mitigation, and Management. The Monitoring pillar focuses on measuring air pollution and related parameters; the Mitigation pillar involves a spectrum of actions to reduce emissions and exposure; and the Management pillar ensures effective implementation, tracking, and control of initiatives to ultimately enhance air quality [4]. In the UAE, the Air Quality Index (AQI) national platform, provides air quality monitoring and predictive analysis. The national AQI platform provides valuable insights into air quality, but its coverage is limited. For example, in many parts of Sharjah, air quality remains unmonitored and unforecasted due to the scarcity of weather stations. Our project aims to address the first pillar of the national agenda — Monitoring Pillar — by proposing an innovative solution: an AIoT-based Energy Efficient Approach to Air Quality Monitoring and Prediction Analysis.

We aim to offer an energy-efficient and cost-effective AIoT-based approach for Air Quality Monitoring and Predictive Systems. In contrast to existing models worldwide, our project's distinctive aim is to create a very low-powered and cost-effective solution. Using advanced technology and innovative methods, we intend to set a new benchmark in air quality monitoring efficiency at both national and international levels. The core issue lies in the inadequate coverage of weather stations in the UAE, where 54 weather stations, while extremely valuable, are unable to cover many parts of the nation. This limitation impedes our ability to comprehend the full extent of air quality challenges and make timely interventions. To address this pressing concern, our project proposes to design and deploy a network of low-power weather stations across our university campus. These stations, distributed across target regions, aim to deliver high-resolution, real-time air quality data. This democratized access to environmental data will empower communities, researchers, and policymakers to proactively engage with and address environmental challenges. Moreover, the use of relatively inexpensive and low-power components used to build the project will ensure both easy implementation and maintenance of stations throughout its planned lifecycle. Our design and implementation plan for an IoT-based air quality monitoring system includes integrating

high-accuracy, low-power sensors to measure key parameters like temperature, humidity, pollutants, and gasses.

To ensure energy efficiency, we plan to incorporate low-power microcontrollers and leverage renewable energy sources, such as solar panels or other suitable options, for power supply. Additionally, we intend to establish a robust central node integrated with cloud infrastructure for data aggregation and processing. Communication will rely on energy-efficient protocols like WiFi, or Zigbee to transmit data over long distances with minimal power consumption. Real-time data processing and storage in cloud and time-series databases, along with a user-friendly dashboard, will enable easy access and interpretation of environmental metrics. Machine learning techniques will further enhance predictive capabilities. Our project endeavors to contribute significantly to the UAE's ambitious vision for a cleaner, healthier environment as outlined in the UAE Vision 2021 and the UAE Air Quality Agenda 2031. Successful implementation within our university campus will serve as a proof of concept, laying the groundwork for a comprehensive solution that extends its benefits to the entirety of the UAE. This will not only support the national agenda but also showcase the UAE's commitment to innovative solutions that address pressing environmental challenges on both a local and global scale.

5.2 Design of the Solution

5.2.1 Hardware Architecture Block Diagram:

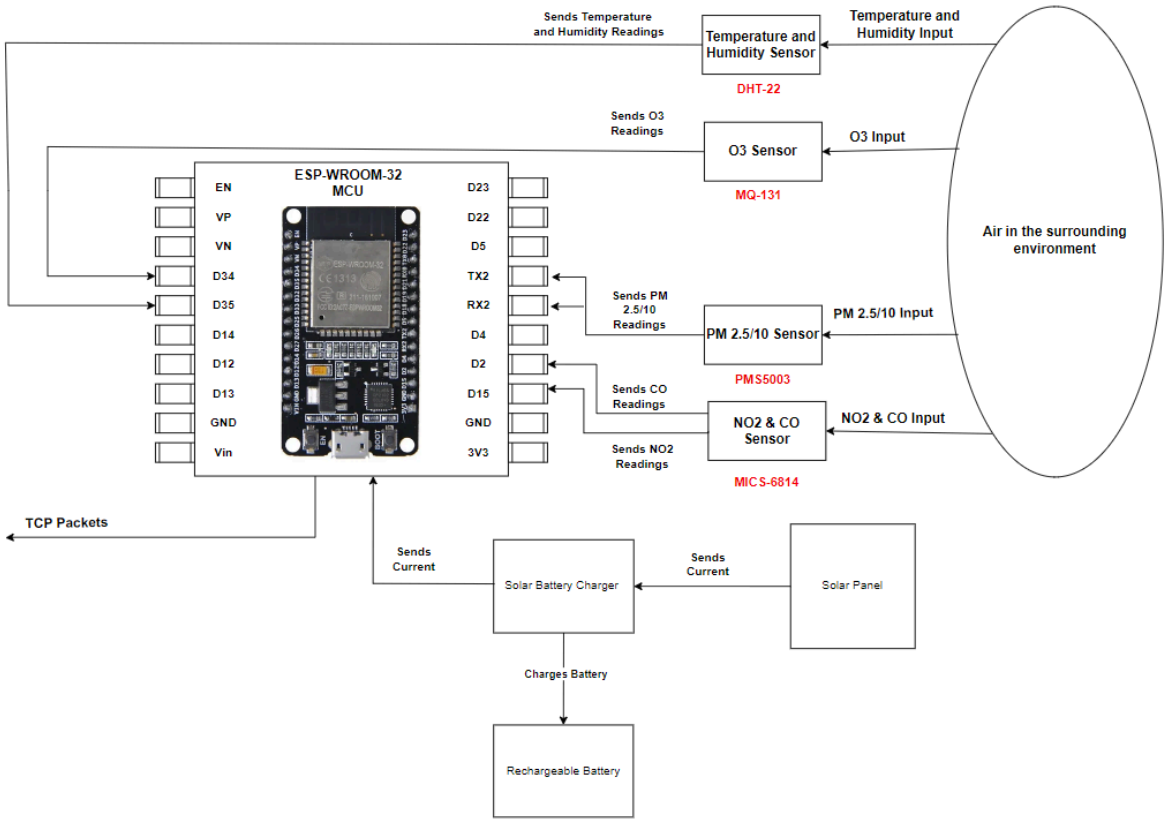


Figure 12. Hardware Architecture Block Diagram

The sensors for each target gas capture the respective inputs from the surrounding environment. Subsequently, the ESP32 MCU processes these inputs and receives the raw sensor data, leveraging its low power consumption and integrated Wi-Fi module for efficient data management. This data is then segmented and organized for transmission by the MCU. The ESP32 MCU collaborates with its integrated Wi-Fi module and can function as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor. The Wi-Fi module sends the TCP packets containing the segmented sensor data over the network, ensuring reliable and uninterrupted transmission.

5.2.2 Software Architecture Block Diagram:

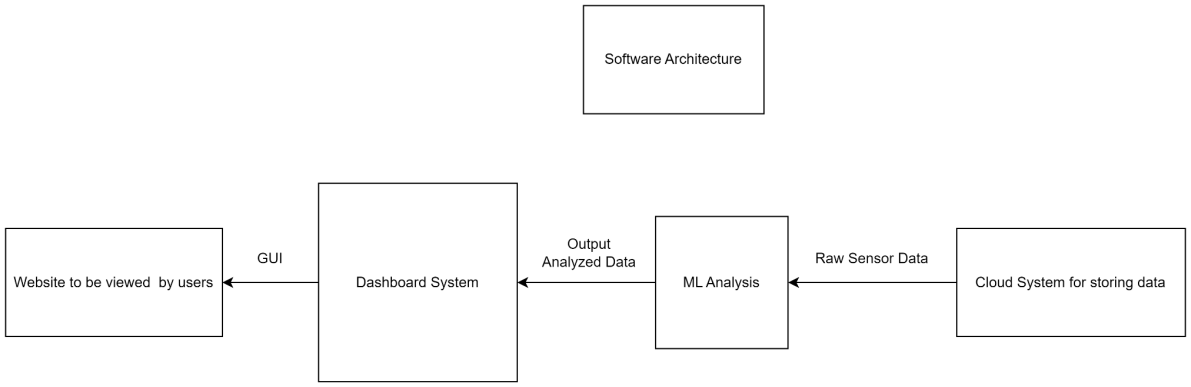


Figure 13: Software Architecture Block Diagram

Cloud storage is responsible for storing the raw sensor data sent by the Wi-Fi module. Next, the predictive model is used to process the raw sensor data and outputs the forecasted results, the Firebase dashboard engages with the cloud services and receives the analyzed data, and presents it in a user-friendly interface on the website.

Our software is committed to delivering precision and relevance in data analytics. We aim to empower users with real-time, location-specific information that supports health-conscious decision-making. By addressing the nuances highlighted in this feedback, we aim to ensure that our system caters to the specific needs of each user, providing the most accurate and user-centric air quality insights.

5.2.3 Workflow Diagram:

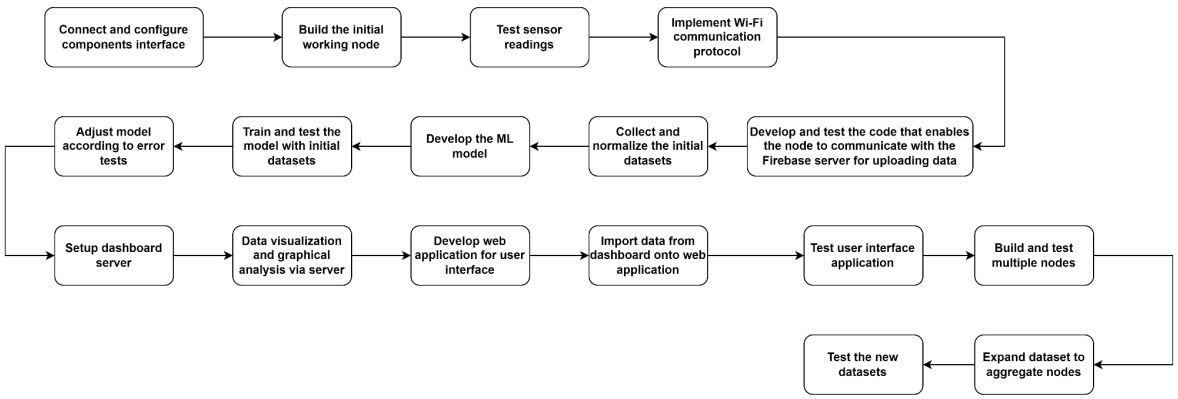


Figure 14. Workflow Diagram

5.2.4 Flowchart

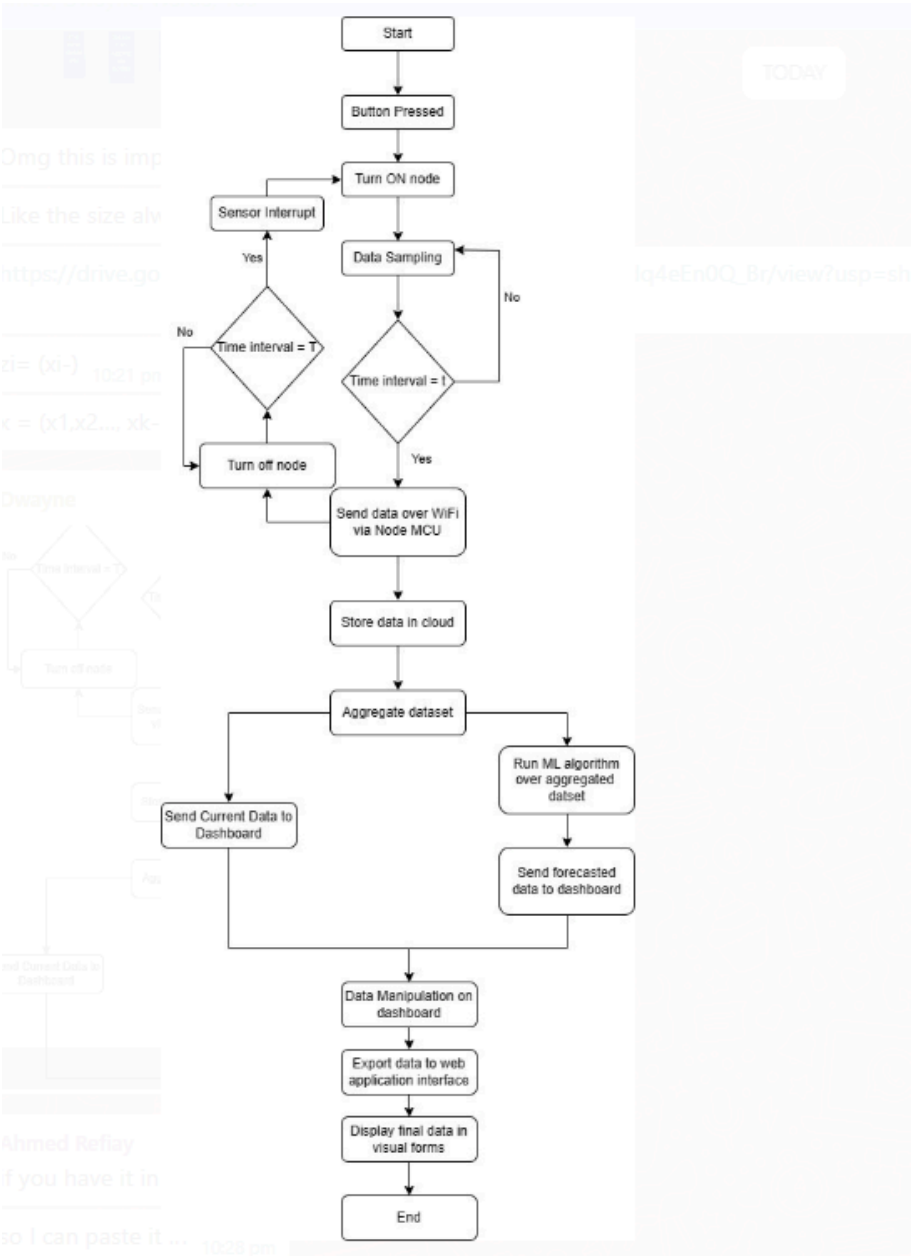


Figure 15. Flowchart

The flowchart depicts a procedural sequence for the data collection and analysis, part of the project design, initiated with the activation of the monitoring stations, where data is collected. The process begins with manually switching on the node (depicted in the "Button Pressed" block). Once the button is pressed, the node is powered on as a result of sensor interrupts. Upon turning on to its active mode, the node, utilizing its sensors, begins active collection of all vital sensor data as we have pre-determined. It will continue data collection

until a short time has passed (only a few seconds, t). Once so, the node will turn off to its sleep mode. Beyond this initial point, the button need not be pressed a second time, interrupts in the sensor will reactivate the node to the same active mode state after the completion of another, albeit longer, time period, T. Meanwhile, the data collected from the node will be sent and stored over to the cloud-based Firebase server through the Node MCU component connected to the multicontroller that facilitates data transmission over WiFi protocols.

The cloud stored data is aggregated, meaning data from multiple nodes or over time periods is combined to form a comprehensive dataset. This aggregated dataset is then used to run a predictive learning algorithm. The results of this machine learning algorithm are sent back to the Firebase dashboard. Parallely, as well, current data On the dashboard, data visualization, aggregation, and analysis provide an interpretive layer for the raw data. The system will be designed to allow for exporting data and customized user data to a web application interface, indicating a layer of user interaction where the system developers can manipulate the data or view it in different forms. Finally, as the flowchart indicates, the final data is displayed to the user in various visual forms, allowing for an interactive and user-friendly presentation of the results.

5.2.5 State Diagram

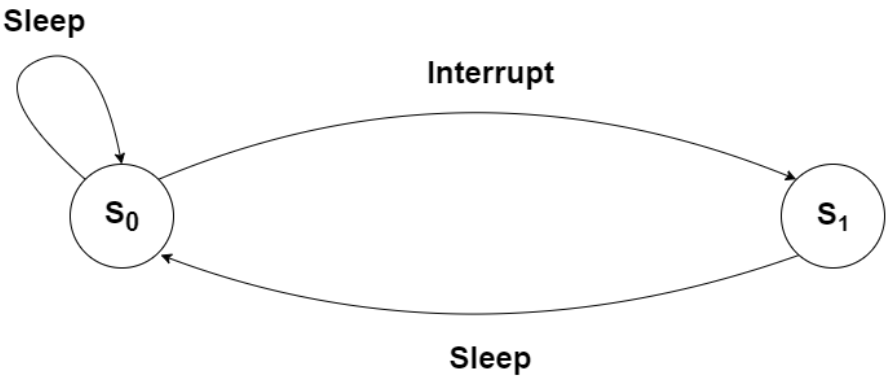


Figure 16. State Diagram

S_0: Standby state where the device draws the lowest possible power while waiting for interrupts.

S_1: On state where an interrupt happens, and measurements are taken from the sensors and sent through the network.

5.2.6 Main Resources Needed

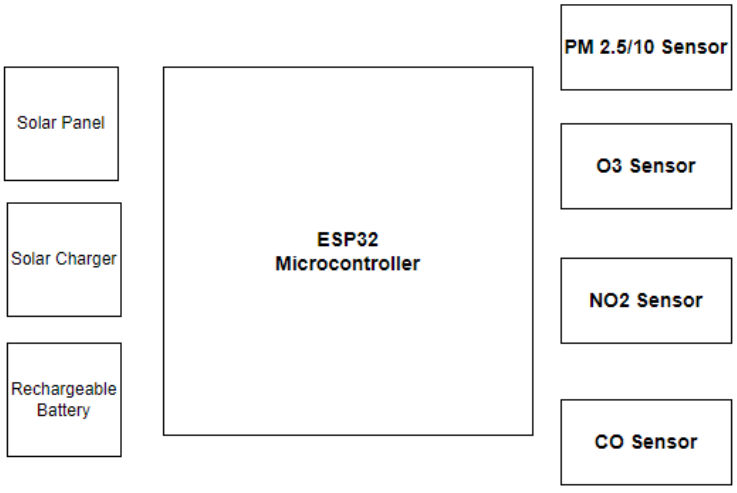


Figure 17. Hardware Resources

Primary resources include the ESP32 Microcontroller, which will be used for interfacing the sensors, and its integrated Wi-Fi module. The sensors for each target gas are

responsible for transforming analog phenomena into digital data. The Wi-Fi module integrated into the ESP32 is responsible for transmitting the TCP packets (Datagrams). Additionally, the system requires a solar panel, a solar battery charger, and a rechargeable battery to ensure uninterrupted power supply and enable energy-efficient operation.

6. Implementation

6.1 Hardware

6.1.1 Microcontroller:

Microcontroller	**Power Consumption**	**No. of GPIO**	**Cost (AED)**
Raspberry Pi 2	2 A	40	154.26
ESP32	240 mA	34	31
ESP8266	170 mA	17	25

Raspberry Pi Zero	Active mode - 38 mA Sleep mode - 2 mA	40	154
ESP-WROOM-32	Active mode: 23.88 mA Sleep mode : 8.14 uA	38	63.95
Arduino Nano	19 mA	22	94
PIC24F16KA102	Active mode: 11-18 mA Sleep mode: 0.55µA	17	13.41
STM32L082	Active mode: 12.7 mA Sleep mode: 0.82 mA	17	23
PSoC3 CY8C3246PVI-147	Active mode: 1.2 mA Sleep mode: 1 uA	25	43.71
ATmega 1284p	Active mode: 0.4 mA Standby mode: 0.1 uA	32	28.72

	Sleep mode: 0.6 uA		
Texas Instruments MSP430	Active mode - 230 uA Standby mode - 0.25 μA Off mode - 0.1 μA	10	150

Table 1. Comparison on Microcontroller Models

Our project is focused on creating a low-powered air quality monitoring system. We evaluated various microcontrollers based on power consumption and GPIO pins. The ESP-WROOM-32 was selected for its relatively low power usage, with an active mode current consumption of approximately 23.88 milliamperes, which translates to an average power consumption of 78.32 mW. With 38 GPIO pins, the ESP-WROOM-32 provides sufficient flexibility for interfacing with sensors and other peripherals. This microcontroller also offers Wi-Fi (802.11 b/g/n) and Bluetooth (v4.2 and BLE) functionality, enabling wireless data transmission and communication with other devices. It is powered by a single or dual-core 32-bit LX6 microprocessor, with a clock frequency adjustable up to 240 MHz[3]. Additionally, its cost-effective price makes it an attractive choice for our project. These features, combined with its low power consumption, make the ESP-WROOM-32 an ideal choice for our energy-efficient air quality monitoring system.

- **ESP-WROOM-32:**

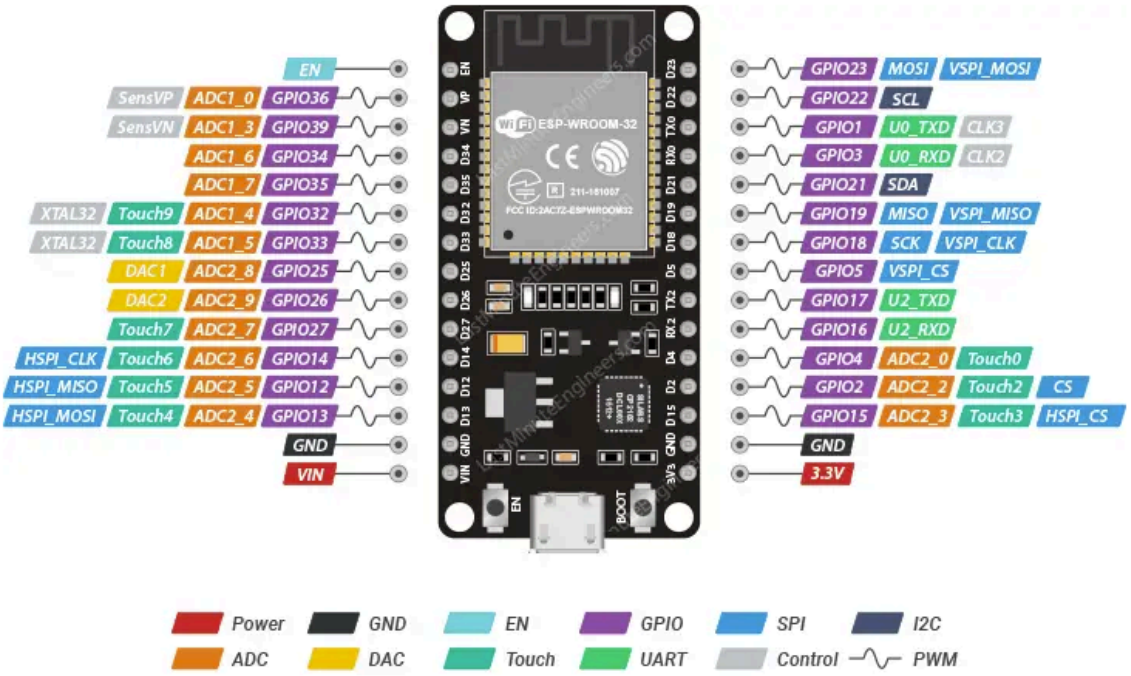


Figure 18 . ESP-WROOM-32 Microcontroller Unit

The ESP-WROOM-32, developed by Espressif Systems, is a powerful module that targets a wide variety of applications. At the core of this module is a 32-bit LX6 microprocessor from the ESP32 family, offering advanced features for IoT and wireless applications. This module embodies the philosophy of Espressif's development kits, providing an economical yet robust platform for rapid prototyping and experimentation.

A key feature of the ESP-WROOM-32 is its versatility and connectivity. It offers integrated Wi-Fi, Bluetooth, and Bluetooth Low Energy (BLE) capabilities, alongside a wide range of General Purpose Input/Output (GPIO) pins for versatile interfacing. This characteristic is particularly valuable for projects that require wireless communication and sensor integration, making the ESP-WROOM-32 an ideal choice for IoT applications or scenarios where connectivity is essential.

The module also includes integrated peripherals, such as timers, communication ports (UART, SPI, I2C), and analog-to-digital converters (ADC), further extending its capabilities. It also supports touch sensors, temperature sensing, and has a built-in hall effect sensor. These features enhance its adaptability to various project requirements.

Complementing the hardware, the development environment for the ESP-WROOM-32 typically includes the Arduino IDE or the Espressif IoT Development Framework (ESP-IDF) for software development. These development environments facilitate tasks such as writing, compiling, and debugging code for the ESP32 microcontrollers, contributing to a comprehensive development ecosystem.

An important aspect of the ESP-WROOM-32 is its low power consumption, which stands out as an ultra-low-power solution. This characteristic is particularly valuable for projects that prioritize energy efficiency, making the ESP-WROOM-32 suitable for battery-powered applications or scenarios where power conservation is critical.

In conclusion, the ESP-WROOM-32 is a dedicated module designed for use with the ESP32 microcontroller series. It provides a powerful, versatile, and energy-efficient solution for a wide range of applications, particularly those involving IoT and wireless communication.

6.1.2 Sensors:

6.1.2.1 Types of Sensor to be used:

- **MQ131 Sensor:**

Figure 19. MQ131 Sensor

The MQ131 sensor is specifically engineered for the precise measurement of ozone (O3) gas concentration in the air. Known for its high sensitivity and fast response time, this

sensor provides an analog output voltage that corresponds to the detected ozone levels. This feature facilitates easy interfacing with microcontrollers or data acquisition systems for further analysis. Its wide operating range, stability over time, and longevity make it adaptable to diverse applications requiring continuous monitoring. Notably, the MQ131 sensor's sensitivity to ozone, coupled with its low power consumption, positions it as a reliable tool for various scenarios such as environmental monitoring and industrial processes. Incorporating the MQ131 sensor offers a robust solution for real-time and accurate ozone concentration monitoring. One point to note is that while the MQ131 sensor is highly sensitive to ozone, it may also respond to other gases to a lesser extent. Therefore, it's important to consider the environment in which the sensor is deployed to ensure accurate readings. This characteristic is particularly valuable for projects that prioritize energy efficiency, making the MQ131 sensor suitable for battery-powered applications or scenarios where power conservation is critical. In conclusion, the MQ131 sensor is a dedicated module designed for use with ozone gas detection. It provides a powerful, versatile, and energy-efficient solution for a wide range of applications, particularly those involving environmental monitoring and industrial processes.

- **MICS-6814 Sensor:**

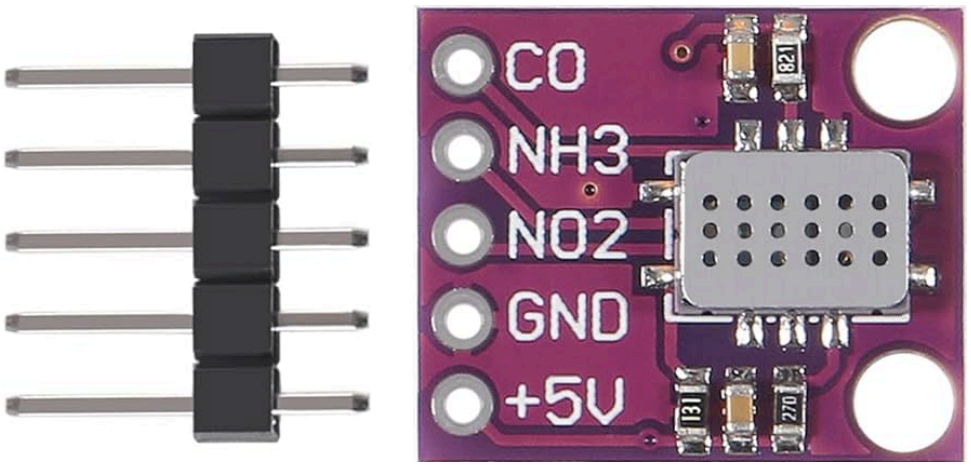


Figure 20. MICS-6814 Sensor

The MICS-6814 sensor module is a versatile multi-gas sensor designed for the simultaneous measurement of specific gasses, including nitrogen dioxide (NO2) and carbon monoxide (CO). Its capacity to detect multiple gasses concurrently, along with its compact

design, makes it particularly suitable for projects focused on monitoring air quality and safety. The sensor provides analog output signals for each detected gas, enabling seamless integration with microcontrollers or other systems for analysis. Equipped with an integrated heater for enhanced sensitivity and response time, the MICS-6814 offers a reliable solution for applications requiring precise measurements of NO2 and CO concentrations.

- **PMS5003 Sensor:**

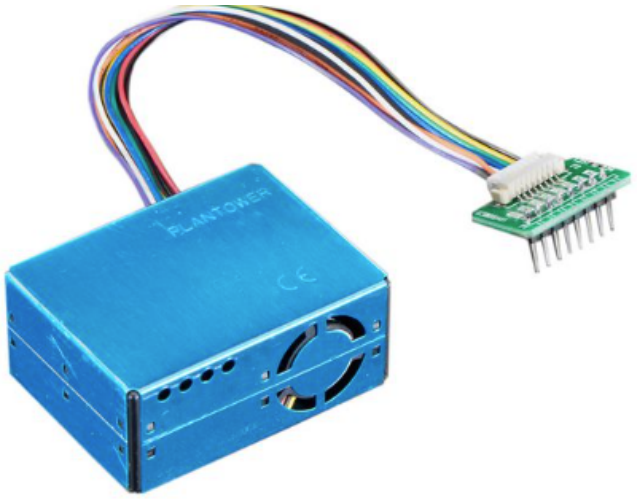


Figure 21. PMS5003 Sensor

The PMS5003 is a specialized air quality sensor engineered for the precise measurement of particulate matter concentrations in the air, focusing specifically on PM2.5 and PM_{10}. Employing laser scattering technology, this compact sensor offers accurate real-time monitoring of fine particulate matter levels. With its digital output signals, typically utilizing UART communication, the PMS5003 allows seamless interfacing with microcontrollers and digital devices. Its high sensitivity to small particulate matter makes it effective for applications demanding detailed insights into air quality, such as environmental monitoring and pollution control projects. Additionally, the sensor's low power consumption enhances its suitability for battery-operated or energy-efficient applications.

- **DHT-22 Sensor:**

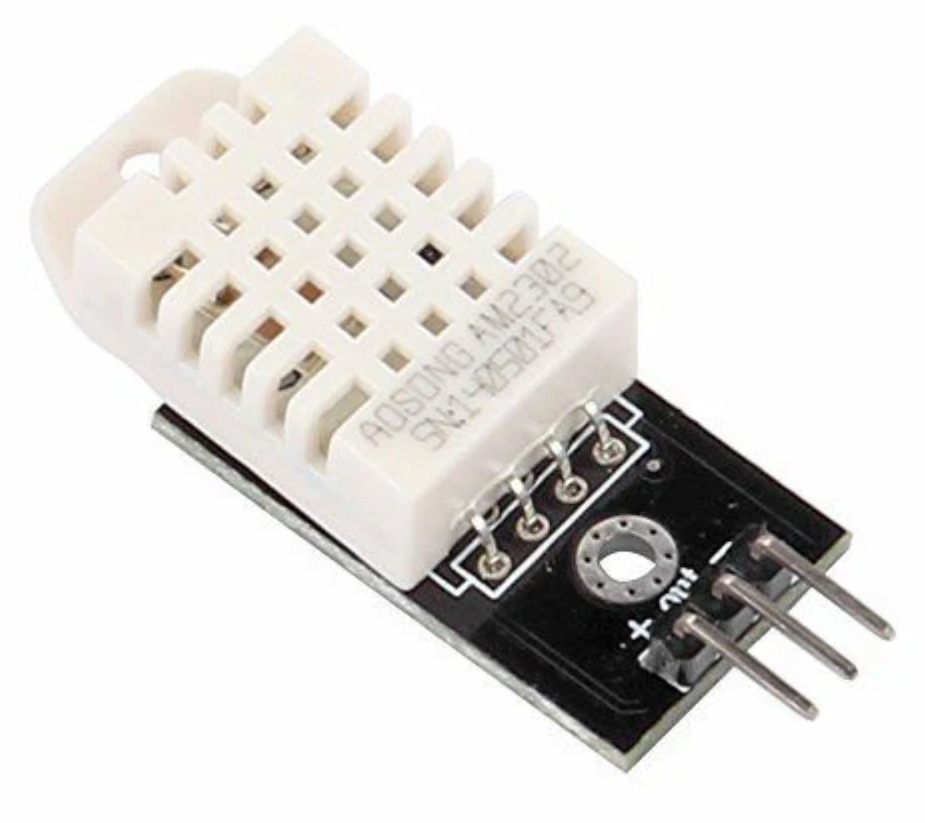


Figure 22. DHT-22 Sensor

The DHT22 sensor is a versatile and widely used digital temperature and humidity sensor. Developed by Adafruit, the DHT22 sensor is renowned for its accuracy, reliability, and ease of use in various environmental monitoring applications. At the heart of the DHT22 sensor is a calibrated digital signal output, providing accurate temperature and humidity readings with high precision. The sensor employs a capacitive humidity sensing element and a thermistor to measure temperature, ensuring consistent and reliable performance across a wide range of environmental conditions. One of the key features of the DHT22 sensor is its simplicity of integration into microcontroller-based projects. The sensor communicates with the microcontroller using a single-wire digital interface, making it easy to connect and interface with a wide range of development boards and microcontrollers. Additionally, the DHT22 sensor requires minimal external components, further simplifying the integration process. The DHT22 sensor offers impressive performance specifications, with a temperature measurement range of -40°C to 80°C and a humidity measurement range of 0% to 100% RH. The sensor's accuracy is typically within ±0.5°C for temperature and ±2% RH for humidity, making it suitable for a wide range of applications, including climate monitoring, HVAC

systems, and indoor environmental quality monitoring. Furthermore, the DHT22 sensor features a fast response time, allowing it to provide real-time temperature and humidity readings with minimal delay. This rapid response time is particularly advantageous for applications that require timely data acquisition and control. In terms of power consumption, the DHT22 sensor operates at a low current draw, making it suitable for battery-powered and energy-efficient applications. The sensor also features a wide operating voltage range, typically between 3.3V and 5V, ensuring compatibility with a variety of power sources and microcontroller platforms. Overall, the DHT22 sensor is an excellent choice for projects that require accurate and reliable temperature and humidity monitoring. Its combination of high performance, ease of use, and low power consumption makes it a popular choice among hobbyists, professionals, and researchers alike.

6.1.2.2 Sensor Analysis:

In order to analyze how efficient the sensors will be, the following figures might shed some light. These tables are acquired from the National Center of Meteorology (NCM), United Arab Emirates and selected as reference for our sensors as we discussed previously of achieving UAE's vision of environmental improvement and public health.

UAE Air Quality Index Value	Color	Index Category	Category Description
0-50	Green	Good	No risk for the population
51-100	Yellow	Moderate	Acceptable for the majority of the population
101-150	Orange	Unhealthy for sensitive groups	Sensitive individuals should avoid exposure
151-200	Red	Unhealthy	Greater proportion of the public may be affected
201-300	Purple	Very Unhealthy	Everyone may experience health effects
301-500	Maroon	Hazardous	Entire population expected to be affected

Table 2. UAE Air Quality Index [9]

Table 2 in [source] depicts the UAE Air Quality Index value which ranges from 0 to 500, and it is segmented into six categories named by descriptor words chosen to indicate the relationship between the air quality and public health. Each category has a designated color from a predetermined color-scale. Different population groups are more sensitive to the harmful effects of the different air pollutants, and to advise each group the six categories have been defined as follows:

- **Good:** Indicates that the air quality is considered satisfactory and the air pollution concentrations pose no risk for the population.
- **Moderate:** In this range, the air quality is considered acceptable. The large majority of the population will not experience any effects. However, for some pollutants there may be a moderate health concern for a very small number of unusually sensitive people. This level is set in alignment to the National Ambient Air Quality Standard.
- **Unhealthy for sensitive groups:** Exposure to ambient concentrations just above the numerical level of the National Standard is not likely to result in concern for most healthy people. This category is intended to serve as a warning so that sensitive individuals can take appropriate 'exposure avoidance' behavior.
- **Unhealthy:** In this range, exposure is associated with an increase in the number of individuals who could potentially experience effects,and includes a greater proportion of the members of the public.
- **Very unhealthy:** Everyone may experience more serious health effects when pollution concentrations are found within this range. It is at this level when health alerts are triggered.
- **Hazardous:** Corresponds to a level of pollution concentrations at which the entire population is expected to be affected. It is at this level when warnings for emergency conditions are issued.

The designation of a category is done for each pollutant based on its individual concentration level, by the means of breaking points. The breaking points act as the concentration range for which a category spans. Table 3 below presents the breaking points for each pollutant.

Sub-indices values	Ozone (O_3)		Carbon Monoxide (CO)	Sulphur Dioxide (SO_2)		Nitrogen Dioxide (NO_2)	Particulate Matter (PM_{10})	Particulate Matter ($PM_{2.5}$)
	$\mu g/m^3$		mg/m^3	$\mu g/m^3$		$\mu g/m^3$	$\mu g/m^3$	$\mu g/m^3$
	1-hour	8-hour	8-hour	1-hour	24-hour	1-hour	24-hour	24-hour

50	-	0-100	0-5	0-92	-	0-100	0-75	0-50
100	-	>100-120	>5-10	>92-350	-	>100-400	>75-150	>50-60
150	200-322	>120-167	>10-14.3	>350-485	-	>400-677	>150-250	>60-75
200	>322-400	>167-206	>14.3-17.8	>485-797	-	>677-1221	>250-350	>75-150
300	>400-792	>206-392	>17.8-35	-	>797-1583	>1221-2349	>350-420	>150-250
500	>792-1184	-	>35-58	-	>1583-2631	>2349-3853	>420-600	>250-500

Table 3. AQI and Pollutants [9]

In order for sensor analysis to be done, we will need to use the following equation which converts a sensor's ppm measurement range into µg/m^3. With this converted value, we can refer to the table with the help of the colored sections and then infer the minimum air quality range of the sensor. Through our inference, we will be able to understand the quality of the sensor to be used.

$$\mu g/m^3 = ppm \times g/mol \times 40.90 \quad (6)$$

- **NO2 Sensor (Nitrogen Dioxide Sensor - MICS-6814):**

Measures the concentration of nitrogen dioxide (NO2) in parts per billion (ppb) or parts per million (ppm) in the air. NO2 is a reddish-brown gas that is a byproduct of combustion processes, and high levels can be harmful to human respiratory health.

Using equation (6):

$$0.05\,ppm \times 46.0\,g/mol \times 40.90 = 94.07\,\mu g/m^3$$

- **CO Sensor (Carbon Monoxide Sensor - MICS-6814):**

The same sensor used for measuring NO2 will be used to measure NO2 as well. The sensor measures the concentration of carbon monoxide (CO) in parts per million (ppm) or even lower levels. CO is a colorless, odorless gas produced by incomplete combustion of fossil fuels. High levels of CO can be life-threatening, as it interferes with the body's ability to transport oxygen.

Using equation (6):

$$1\,ppm \times 28.0\,g/mol \times 0.0409 = 1.15\,mg/m^3$$

- **PM 2.5 Sensor (Particulate Matter 2.5 Sensor - PMS5003):**

Measures fine particulate matter (PM2.5), which consists of particles with a diameter of 2.5 micrometers or smaller. PM2.5 is associated with various health issues and can be expressed in micrograms per cubic meter (µg/m³). It is a key indicator of air quality.

According to the datasheet:

$$= 1\ \mu g/m^3$$

- **O3 Sensor (Ozone Sensor - MQ131):**

Measures the concentration of ozone (O3) in parts per billion (ppb) or parts per million (ppm). Ozone is an important component of the Earth's atmosphere, but ground-level ozone, a major component of smog, can be harmful to human health and is monitored for air quality assessments.

Using equation (6):

$$0.01\ ppm \times 48.0\ g/mol \times 40.90 = 19.63\ \mu g/m^3$$

6.1.3 Battery Charging System:

- **BQ24074 Solar Charger**:

We chose the BQ24074, a highly integrated power management device from Texas Instruments, for our project. It's designed to charge single-cell Li-ion or Li-polymer batteries safely and efficiently. With a wide input voltage range of 5-10V, it's compatible with various power sources, including solar panels. It can provide a charge current up to 1.5A, which is beneficial for quickly charging larger battery packs. The charger also features a built-in power path management system, ensuring that our system remains powered even when the input voltage dips below 4.5V.

- **Solar Panel:**

Our system uses a solar panel to convert sunlight into electricity. The efficiency of our solar panel is crucial as it determines how much of the sunlight it can convert into usable electricity. We selected our solar panel based on the intensity and availability of sunlight in our location, the power requirements of our system, and the charging specifications of our battery and solar charger.

- **Rechargeable 3.4V Battery:**

Our system uses a rechargeable 3.4V battery to store the electricity generated by the solar panel for later use. We chose a lithium-ion or lithium-polymer battery for their high energy density and low self-discharge rate. These batteries can be recharged multiple times, making them cost-effective and environmentally friendly. We selected our battery based on its capacity (measured in mAh), which determines how long it can power our system before needing to be recharged.

6.2 Software

6.2.1 Entities:

Entities within the software encompass data structures representing sensor readings for O3, NO2, CO, PM10, PM10, and temperature/humidity (DHT22).These structures facilitate efficient collection, storage, and transmission of air quality data. For the ESP-WROOM-32 microcontroller, these entities are defined to interface with the specific GPIO pins connected to each sensor. Additionally, entities are defined for communication interfaces, ensuring seamless interaction between the ESP-WROOM-32 MCU, Wi-Fi module, cloud system, and server. User-related entities include structures for dashboard display and website presentation, allowing for the visualization and interpretation of air quality data.

6.2.2 Functions:

6.2.2.1 Sensor Data Collection:

In order to integrate our sensor readings digitally into a software application, we will utilize the ESP-WROOM-32's development environment, which includes the Arduino IDE or the Espressif IDF (IoT Development Framework). These environments support the ESP-WROOM-32 microcontroller and provide a suite of tools for software development, including libraries and APIs for sensor interfacing and data processing. Hence, we will be using the C/C++ programming language to digitally integrate our sensor readings and manage communication with peripheral devices.

O3 Sensor (MQ131):

We will connect the MQ131 sensor to the ESP-WROOM-32 microcontroller, utilizing either analog or digital pins based on the specific requirements of our setup. Subsequently, we'll create functions dedicated to initializing the sensor, reading the analog or digital signal from the MQ131 sensor, and processing the raw data. To make sense of the data, we'll refer to the MQ131 datasheet, which outlines how to interpret the raw data and convert it into a concentration unit, such as parts per billion.

NO2 and CO Sensor (MICS-6814):

For the MICS-6814 sensor, we'll connect it to the ESP-WROOM-32 microcontroller and create functions to initialize the sensor, read analog or digital signals for both NO2 and CO, and process the raw data. To obtain concentration values, our reference point will be the sensor's datasheet, providing insights into interpreting the raw data and calibrating the sensor readings if necessary.

PM2.5/PM10 Sensor (PMS5003):

For the PMS5003 sensor, we'll establish communication via UART with the ESP-WROOM-32 microcontroller. We'll develop functions for UART communication to initialize the sensor, send commands to start data acquisition, receive data from the sensor, and subsequently parse the received data to extract PM2.5 and PM10 concentrations. Here, too, the sensor's datasheet will be instrumental in understanding communication protocols, configuring the sensor, and interpreting data format.

Temperature/Humidity Sensor (DHT22):

For the DHT22 sensor, we'll connect it to the ESP-WROOM-32 microcontroller and develop functions to initialize the sensor, read digital signals for temperature and humidity, and process the raw data. The datasheet provided for the DHT22 sensor will guide us in interpreting the raw data and converting it into temperature and humidity values.

Here is an example of how the code for the sensor readings can be structured:

```
// Inclusion of necessary libraries and headers
// Function to initialize sensors
```

```
void initializeSensors()
{
   // Initialize sensor connections and configurations
}

// Functions to read concentrations for each sensor
float readO3Concentration() { /* ... */ }
float readNO2Concentration() { /* ... */ }
float readCOConcentration() { /* ... */ }
float readPM25Concentration() { /* ... */ }
float readPM10Concentration() { /* ... */ }

int main()
{
   initializeSensors();

   while (1)
   {
      // Read sensor data
      float o3 = readO3Concentration();
      float so2 = readSO2Concentration();
      float no2 = readNO2Concentration();
      float co = readCOConcentration();
      float pm25 = readPM25Concentration();
      float pm10 = readPM10Concentration();

      // Process and use sensor data as needed
      // Add delay or implement a suitable data collection frequency
   }

   return 0;
}
```

6.2.2.2 Data Transmission:

To enable the ESP-WROOM-32 microcontroller (MCU) to communicate with the Wi-Fi module and transmit sensor readings through TCP, a series of coordinated steps are involved. First and foremost, the hardware connections must be established, linking the relevant pins of the ESP-WROOM-32 MCU with those of the Wi-Fi module. This typically entails establishing a Serial Peripheral Interface (SPI) connection, along with connections for control pins such as Chip Select (CS), Reset, and Interrupt. Once the hardware is set up, the initialization phase commences, initializing SPI communication between the ESP-WROOM-32 MCU and the Wi-Fi module. Espressif provides Wi-Fi libraries that abstract the low-level details of communication, simplifying the interaction between the MCU and the Wi-Fi module.

Following initialization, the Wi-Fi module must be configured to connect to the desired Wi-Fi network. This involves providing the SSID (Wi-Fi network name) and passphrase, if applicable. Subsequently, the TCP/IP stack provided by the Wi-Fi module is utilized to establish a TCP connection, ensuring reliable and ordered data transmission. Meanwhile, the ESP-WROOM-32 MCU collects sensor readings, often acquired from connected analog or digital sensors. These readings are then formatted into a suitable data structure for transmission, typically converted into a format like JSON for ease of parsing on the receiving end.

With the data formatted, the Wi-Fi module is employed to send the sensor data over the established TCP connection to the Firebase Realtime Database. Firebase provides a cloud-based database service that allows for real-time data storage and synchronization across multiple clients. By storing the sensor data in the Firebase Realtime Database, it becomes accessible for further processing, analysis, and visualization. The reliance on TCP ensures the reliable and ordered delivery of the transmitted data. To bolster reliability, error handling and acknowledgment mechanisms are implemented. This involves checking for acknowledgments (ACKs) from the receiving end and retransmitting data if necessary.

6.2.2.3 Cloud Processing:

Cloud processing plays a pivotal role in managing the influx of air quality data from multiple Internet of Things (IoT) devices. For this purpose, we have opted for Firebase, a versatile and scalable cloud platform that seamlessly integrates with our IoT architecture. The data flow within our system begins with the ESP-WROOM-32 collecting sensor readings for O_3, NO_2, CO, and $PM_{2.5}$/PM_{10}. The low power consumption of the ESP-WROOM-32 ensures energy efficiency during data collection. Subsequently, this data is transmitted to the cloud through a Wi-Fi module, where Firebase's Realtime Database efficiently stores the incoming sensor data. The NoSQL nature of Firebase facilitates flexible and efficient data storage. The integration with Firebase extends beyond mere data storage. Cloud functions in Firebase are employed to trigger real-time updates on a dashboard, providing developers with a visual representation of the air quality index (AQI) in different locations. This live monitoring capability empowers developers to make informed decisions based on current environmental conditions.

As an added layer of sophistication, our project incorporates machine learning (ML) models for predictive analysis of AQI. Firebase Cloud Functions seamlessly integrate these models into the cloud infrastructure, allowing for historical data analysis and the prediction of future air quality conditions. The results of these analyses are displayed on the dashboard, providing developers with insights for proactive decision-making. User accessibility is a key aspect of our project. Processed and analyzed data, along with real-time information, is dynamically integrated into a user-friendly website. Users, including those with health conditions related to the target gasses, can access the website to make informed decisions about their activities based on current and predicted air quality conditions.

6.2.2.4 Dashboard Display:

The primary objective of the dashboard segment in our project is to provide an intuitive and interactive platform for displaying real-time air quality data. The goal is to make air quality data comprehensible and actionable for users, especially those within the university campus, by visualizing the data in an accessible format. This platform aims to empower users with the ability to make informed decisions about their health and outdoor activities based on the latest AQI readings. The dashboard not only serves as a central hub for current air quality data but also offers predictive insights into future air quality trends, enabling users to plan ahead. Emphasis has been placed on ensuring that the data is presented in a way that is easily interpretable by a diverse audience, including students, faculty, and campus visitors.

Design and Implementation

The user interface is strategically designed to provide immediate visual feedback; the homepage promptly presents AQI nodes placed around the campus, each with a distinct color indicative of the air quality level — from 'Good' green to 'Hazardous' purple. This visual immediacy empowers users to assess the air quality with just a glance. Upon visiting the site, users are prompted to interact with these nodes to access detailed environmental data for their selected location.

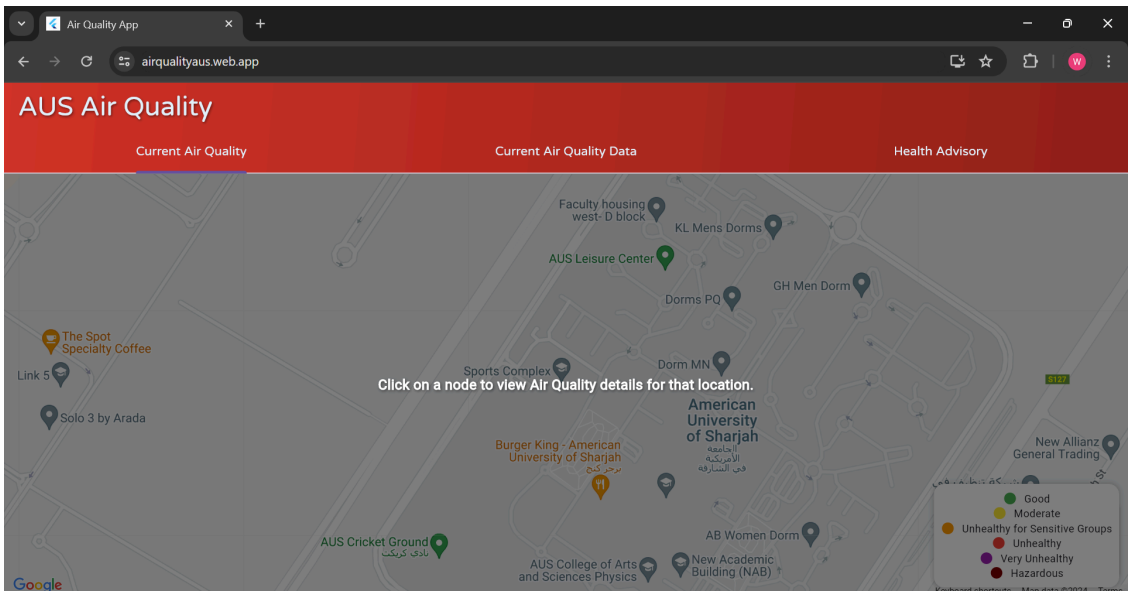


Figure 23: AUS Air Quality Website initial prompt for the user to click on the node according to the preferred location

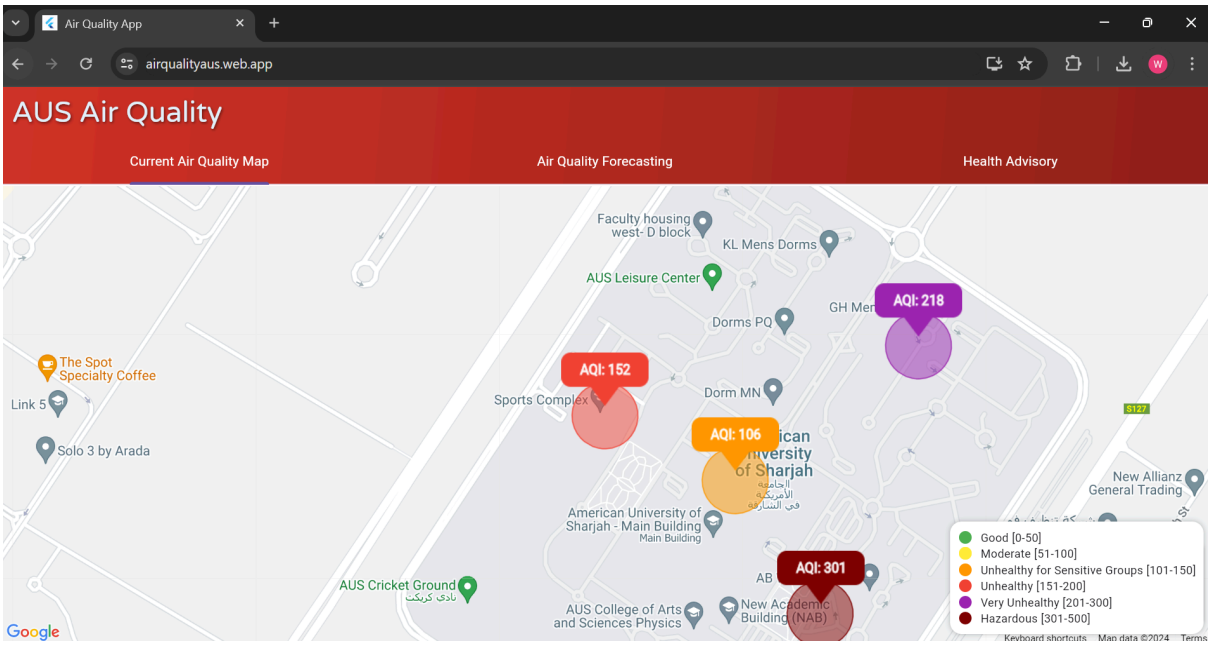


Figure 24: AUS Air Quality Website Map

The second tab of the dashboard is particularly dynamic, offering up-to-date readings on the AQI value, pollutant subindexes, and essential weather metrics like temperature and humidity. Each data point is refreshed every 10 minutes, ensuring that users receive the most current information. An integral feature of the dashboard is the time-stamped assurance of data recency, which fosters user trust in the system's readings.

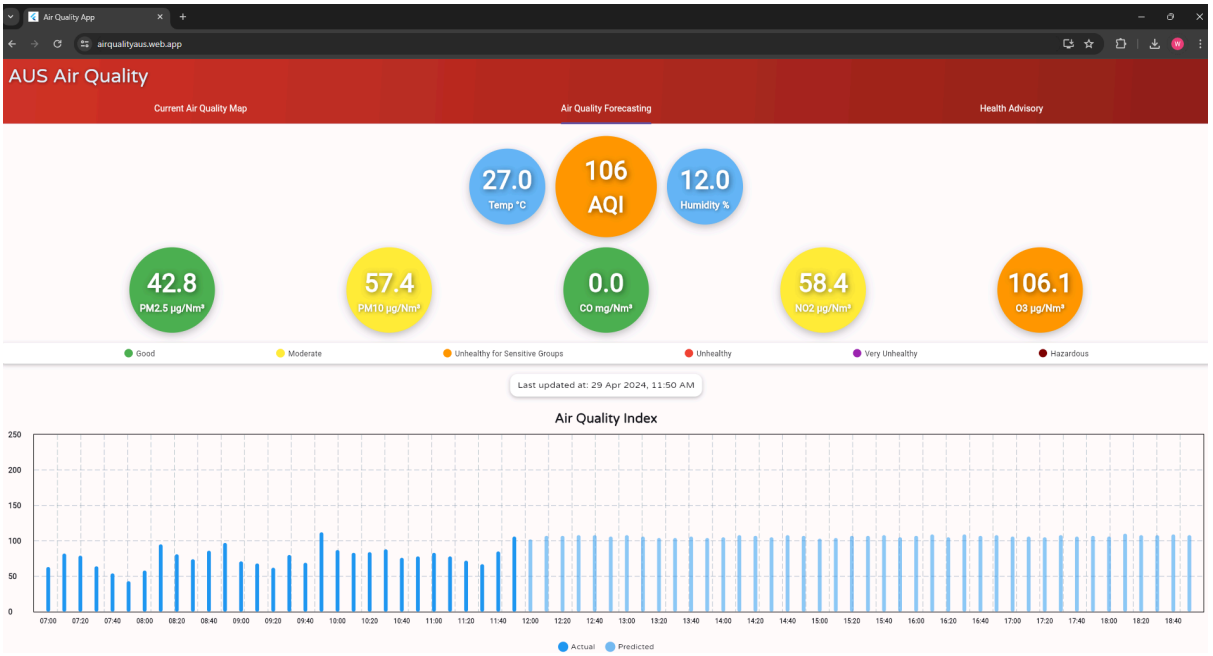


Figure 25: AUS Air Quality Website Current Air Quality Data and Predicted Data

Below the real-time data, the dashboard transitions into a comprehensive visual narrative with a 24-hour graphical representation of the AQI trends. This graph is pivotal in that it doesn't just reflect past and present air quality but also extends into predictive analytics, offering forecasts beyond the latest reading. Through these features, the dashboard not only informs but also anticipates, guiding users in making informed decisions about their immediate and forthcoming activities.

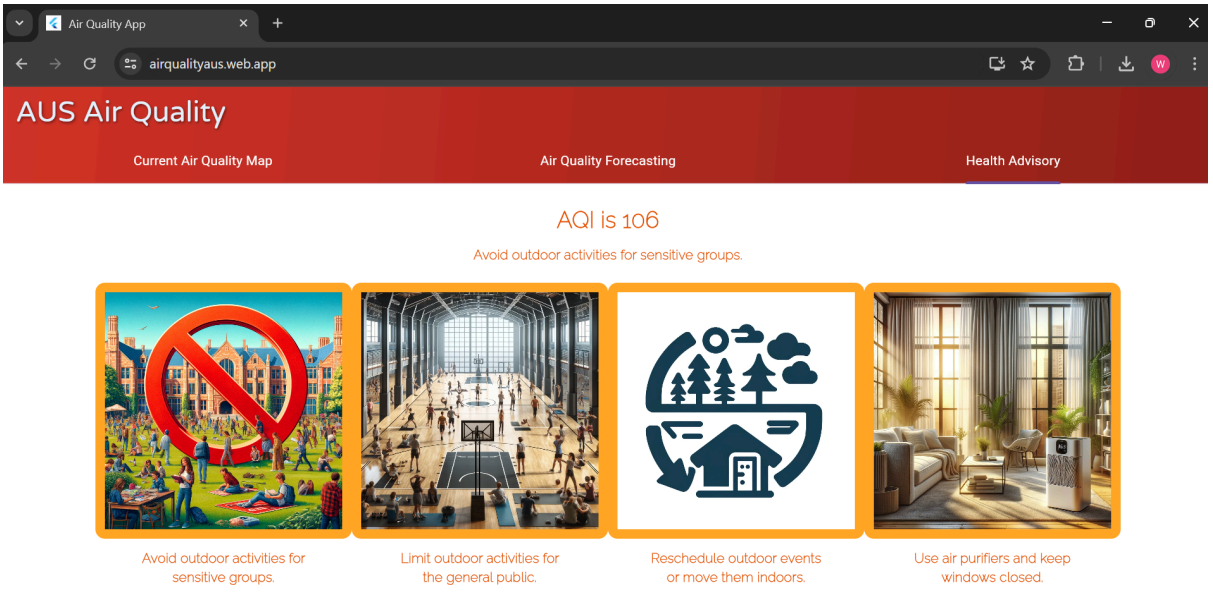


Figure 26: AUS Air Quality Website Health Advisory Tab

These updates also extend to the Health Advisory tab, where the health advice is dynamically adjusted based on the new AQI readings. This tab changes to reflect health recommendations appropriate for one of two AQI ranges, ensuring that users receive timely and relevant health guidance based on the latest air quality data.

Technology Stack

The technology stack for the AQI dashboard was selected with performance, scalability, and real-time data handling in mind. Flutter, an open-source UI software development kit created by Google, was chosen for its ability to build natively compiled applications for mobile, web, and desktop from a single codebase, facilitating rapid development and ease of maintenance. The Firebase Realtime Database is utilized as the backend service, providing a cloud-hosted NoSQL database that allows for storing and syncing data between users in real-time. This choice enables the dashboard to reflect live updates of air quality readings promptly and reliably. The synergy between Flutter and Firebase provides a robust infrastructure capable of handling high-frequency data updates and ensuring that the dashboard's users always have access to the most recent air quality information.

Firebase Realtime Database Architecture

The backbone of our AQI dashboard's real-time functionality is the well-structured Firebase Realtime Database. We have established a node for each AQI station on campus, labelled AQI1 through AQI7, facilitating precise monitoring at various critical points. Inside each node, data is meticulously organized in a chronological hierarchy by date, then further broken down into time intervals, capturing snapshots of air quality at regular ten-minute intervals throughout the day.

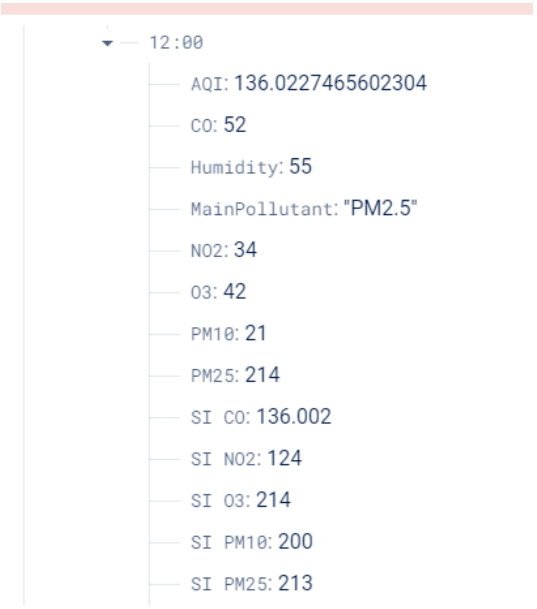


Figure 27: Structure of AQI2 node

Each node within this structure is a repository of rich data. Take, for example, AQI2; it encapsulates a wealth of information including AQI values, individual pollutant concentrations like CO, NO2, O3, PM10, PM2.5, and pertinent environmental parameters such as humidity and temperature. This multidimensional dataset enables not just the visualization of current air quality but also the derivation of sub-indexes that inform the overall AQI calculation.

Another layer to this architecture is the predictive data, labeled under ML_AQI1 to ML_AQI7 nodes. This data, generated by running machine learning models, is seamlessly integrated into the database, providing forecasts that extend the utility of the dashboard beyond the present conditions. The forecasts anticipate air quality trends and enable users to plan for future activities with greater confidence.

The provided code defines a StatefulWidget called AQIWidget in Flutter, which is responsible for displaying the Air Quality Index (AQI) and related information. The widget retrieves AQI data from a Firebase Realtime Database and listens for changes in the data using a StreamBuilder. It fetches the most recent date and time from the database and calculates the average values for different pollutants (NO2, CO, O3, PM2.5, and PM10) based on the last six data points. Using these average values, the code computes the subindices for each pollutant using predefined calculation functions.

The overall AQI value is then determined by selecting the maximum subindex among all pollutants. The code also identifies the pollutant responsible for the maximum subindex, which is considered the primary pollutant contributing to the AQI value. Both the AQI value and the subindices are written back to the Firebase Realtime Database for the most recent date and time.

```
import 'package:aus_air_quality/widgets/newaqimap.dart';
import 'package:flutter/material.dart';
import 'package:firebase_database/firebase_database.dart';
import 'aqi_calculator.dart';

class AQIWidget extends StatefulWidget {
  final String aqiNode;

  const AQIWidget({required this.aqiNode});

  @override
  _AQIWidgetState createState() => _AQIWidgetState();
}

class _AQIWidgetState extends State<MLAQIWidget> {
  late DatabaseReference _mlAqiRef;

  @override
  void initState() {
    super.initState();
    _mlAqiRef = FirebaseDatabase.instance.ref('ML_' + widget.aqiNode);
  }

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<DatabaseEvent>(
      stream: _mlAqiRef.onValue,
      builder: (context, snapshot) {
        if (snapshot.hasError) {
          return Center(child: Text('Error fetching data'));
        }
        if (!snapshot.hasData || snapshot.data!.snapshot.value == null) {
          return Center(child: CircularProgressIndicator());
        }
```

```
        Map<dynamic, dynamic> aqiData =
            snapshot.data!.snapshot.value as Map<dynamic, dynamic>;
        String mostRecentDate = aqiData.keys.last;
        Map<dynamic, dynamic> dateData =
            aqiData[mostRecentDate] as Map<dynamic, dynamic>;
        List<String> sortedTimes =
            dateData.keys.cast<String>().toList()..sort((a, b) =>
b.compareTo(a));

        for (String time in sortedTimes) {
          double averageNO2 = _getAveragePollutantValue(dateData, time,
'NO2');
          double averageCO = _getAveragePollutantValue(dateData, time, 'CO');
          double averageO3 = _getAveragePollutantValue(dateData, time, 'O3');
          double averagePM25 = _getAveragePollutantValue(dateData, time,
'PM25');
          double averagePM10 = _getAveragePollutantValue(dateData, time,
'PM10');

          double no2Subindex = calculateNO2Subindex(averageNO2);
          double coSubindex = calculateCOSubindex(averageCO);
          double o3Subindex = calculateO3Subindex(averageO3);
          double pm25Subindex = calculatePM25Subindex(averagePM25);
          double pm10Subindex = calculatePM10Subindex(averagePM10);

          double aqi = _calculateAQI(no2Subindex, coSubindex, o3Subindex,
pm25Subindex, pm10Subindex);
          String pollutant = _getAQIPollutant(no2Subindex, coSubindex,
o3Subindex, pm25Subindex, pm10Subindex);

          _writeAQIToDatabase(mostRecentDate, time, aqi, pollutant);
          _writeSubindicesToDatabase(mostRecentDate, time, coSubindex,
no2Subindex, o3Subindex, pm25Subindex, pm10Subindex);
        }

        return Center(child: Text('All data processed for $mostRecentDate'));
      },
    );
  }

  double _getAveragePollutantValue(Map<dynamic, dynamic> dateData, String
time, String pollutant) {
```

```
  final timeData = dateData[time] as Map<dynamic, dynamic>;
  return (timeData[pollutant] as num?)?.toDouble() ?? 0.0;
}

double _calculateAQI(double no2Subindex, double coSubindex,
    double o3Subindex, double pm25Subindex, double pm10Subindex) {
  double maxSubindex = [
    no2Subindex,
    coSubindex,
    o3Subindex,
    pm25Subindex,
    pm10Subindex
  ].reduce((a, b) => a > b ? a : b);
  return maxSubindex;
}

String _getAQIPollutant(double no2Subindex, double coSubindex,
    double o3Subindex, double pm25Subindex, double pm10Subindex) {
  double maxSubindex = [
    no2Subindex,
    coSubindex,
    o3Subindex,
    pm25Subindex,
    pm10Subindex
  ].reduce((a, b) => a > b ? a : b);

  if (maxSubindex == no2Subindex) return 'NO2';
  if (maxSubindex == coSubindex) return 'CO';
  if (maxSubindex == o3Subindex) return 'O3';
  if (maxSubindex == pm25Subindex) return 'PM2.5';
  if (maxSubindex == pm10Subindex) return 'PM10';

  return 'Unknown';
}

void _writeAQIToDatabase(
    String date, String time, double aqi, String pollutant) {
  _mlAqiRef.child(date).child(time).update({
    'AQI': aqi,
  });
}
```

```
  void _writeSubindicesToDatabase(
      String date, String time, double si_co, double si_no2, double si_o3,
double si_pm25, double si_pm10) {
   _mlAqiRef.child(date).child(time).update({
      'SI CO': si_co,
      'SI NO2': si_no2,
      'SI O3': si_o3,
      'SI PM25': si_pm25,
      'SI PM10': si_pm10,
    });
  }

}
```

AQI Map Functionality and Design

Design and Implementation

The AQIMapScreen widget is a pivotal element of the air quality dashboard, designed to provide a real-time, interactive map interface that displays air quality indices (AQI) across various nodes located around the campus. This widget integrates Google Maps through the GoogleMapsFlutter plugin, offering users a familiar and intuitive map experience enhanced with custom data overlays.

Real-Time Data Integration

Central to the functionality of AQIMapScreen is its ability to handle real-time data. This is achieved through Firebase's Realtime Database integration, where the widget subscribes to specific database references for each AQI node (e.g., FirebaseDatabase.instance.ref().child('AQI1')). By using StreamBuilder, the widget listens for updates in real-time, ensuring that any changes in AQI values are immediately reflected on the map with updated marker colors and labels.

Interactive Markers and Custom Visualization

Each AQI node on the map is represented by a marker, which is color-coded to indicate the current AQI level, providing a quick visual assessment of air quality. These markers are not static; they are interactive, allowing users to tap on any marker to get a detailed view of the air quality at that location. The implementation includes custom methods like _addAQI1Marker and _addAQI2Marker, which dynamically generate markers based on the latest AQI data fetched. Additionally, each marker is surrounded by circles of varying radii, painted in semi-transparent layers to visually represent the area potentially affected by the measured air quality levels.

Enhancing User Experience with Instructional Messages

When a user first interacts with the AQIMapScreen, an instructional message is displayed, guiding them on how to use the map effectively. This message fades after a brief period, ensuring it provides necessary guidance without cluttering the interface. The timer for this instructional overlay is managed through the _showInstructionMessageTimer method, which uses Future.delayed to control visibility based on user interaction and time elapsed.

Integration with Other Dashboard Components

Selecting a marker on the AQIMapScreen triggers a series of events within the application. This interaction is managed through the _handleAQINodeSelection method, which updates the state to reflect the selected node and potentially navigates to other tabs or views within the app, such as detailed pollutant breakdowns or health advisories. This seamless integration is crucial for providing a cohesive user experience, where data exploration feels intuitive and connected across different components of the dashboard.

Below is the code for the Map showcasing AQI nodes in our website:

```
import 'package:firebase_database/firebase_database.dart';
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:label_marker/label_marker.dart';

// The AQIMapScreen widget is a stateful widget that displays a Google Map
with AQI data
```

```
class AQIMapScreen extends StatefulWidget {
  // Callback function to handle AQI node selection
  final Function(String) onAQINodeSelected;
  // TabController to control the tab navigation
  final TabController tabController;

  // Constructor for the AQIMapScreen widget
  const AQIMapScreen({
    Key? key,
    required this.onAQINodeSelected,
    required this.tabController,
  }) : super(key: key);

  @override
  _AQIMapScreenState createState() => _AQIMapScreenState();
}

// The _AQIMapScreenState class is the state for the AQIMapScreen widget
class _AQIMapScreenState extends State<AQIMapScreen> {
  // Initialize Firebase database references for each AQI node
  final DatabaseReference _databaseReferenceAQI1 =
      FirebaseDatabase.instance.ref().child('AQI1');
  final DatabaseReference _databaseReferenceAQI2 =
      FirebaseDatabase.instance.ref().child('AQI2');
  final DatabaseReference _databaseReferenceAQI3 =
      FirebaseDatabase.instance.ref().child('AQI3');
  final DatabaseReference _databaseReferenceAQI4 =
      FirebaseDatabase.instance.ref().child('AQI4');

  // Variables to store AQI values for each node
  int? _aqi1, _aqi2, _aqi3, _aqi4;

  // Google Maps controller
  GoogleMapController? _mapController;

  // Sets to store markers and circles on the map
  final Set<Marker> _markers = {};
  final Set<Circle> _circles = {};

  // Default location and locations for each AQI node
  final LatLng _defaultLocation = const LatLng(25.312581, 55.488835);
  final LatLng _aqi1Location = const LatLng(25.312581, 55.488835);
```

```
  final LatLng _aqi2Location = const LatLng(25.311260, 55.491774);
  final LatLng _aqi3Location = const LatLng(25.308551, 55.493693);
  final LatLng _aqi4Location = const LatLng(25.314004, 55.495914);

  @override
  void initState() {
    super.initState();
    // Fetch the latest AQI values for each node when the widget is
initialized
    _fetchLatestAQI1Value();
    _fetchLatestAQI2Value();
    _fetchLatestAQI3Value();
    _fetchLatestAQI4Value();
  }

  // Fetch the latest AQI value for AQI1 node from the Firebase database
  void _fetchLatestAQI1Value() {
    _databaseReferenceAQI1.onValue.listen((event) {
      if (event.snapshot.value != null) {
        // Extract the AQI data from the database snapshot
        final aqiData = event.snapshot.value as Map<dynamic, dynamic>? ?? {};
        final mostRecentDate = aqiData.keys.last;
        final dateData = Map<dynamic, dynamic>.from(aqiData[mostRecentDate] as
Map);
        final mostRecentTime = dateData.keys.last;
        final lastEntryData = Map<dynamic,
dynamic>.from(dateData[mostRecentTime] as Map);
        final aqi1Value = (lastEntryData['AQI']?.toDouble() ?? 0.0).round();

        // Update the state with the fetched AQI value and add the
corresponding marker and circles
        setState(() {
          _aqi1 = aqi1Value;
          _addAQI1Marker(_aqi1Location, _aqi1);
          _addAQICircles1(_aqi1Location, _aqi1);
        });
      }
    });
  }

  // Fetch the latest AQI value for AQI2 node from the Firebase database
  void _fetchLatestAQI2Value() {
```

```
    _databaseReferenceAQI2.onValue.listen((event) {
      if (event.snapshot.value != null) {
        // Extract the AQI data from the database snapshot
        final aqiData = event.snapshot.value as Map<dynamic, dynamic>? ?? {};
        final mostRecentDate = aqiData.keys.last;
        final dateData = Map<dynamic, dynamic>.from(aqiData[mostRecentDate] as
Map);
        final mostRecentTime = dateData.keys.last;
        final lastEntryData = Map<dynamic,
dynamic>.from(dateData[mostRecentTime] as Map);
        final aqi2Value = (lastEntryData['AQI']?.toDouble() ?? 0.0).round();

        // Update the state with the fetched AQI value and add the
corresponding marker and circles
        setState(() {
          _aqi2 = aqi2Value;
          _addAQI2Marker(_aqi2Location, _aqi2);
          _addAQICircles2(_aqi2Location, _aqi2);
        });
      }
    });
  }

  // Fetch the latest AQI value for AQI3 node from the Firebase database
  void _fetchLatestAQI3Value() {
    _databaseReferenceAQI3.onValue.listen((event) {
      if (event.snapshot.value != null) {
        // Extract the AQI data from the database snapshot
        final aqiData = event.snapshot.value as Map<dynamic, dynamic>? ?? {};
        final mostRecentDate = aqiData.keys.last;
        final dateData = Map<dynamic, dynamic>.from(aqiData[mostRecentDate] as
Map);
        final mostRecentTime = dateData.keys.last;
        final lastEntryData = Map<dynamic,
dynamic>.from(dateData[mostRecentTime] as Map);
        final aqi3Value = (lastEntryData['AQI']?.toDouble() ?? 0.0).round();

        // Update the state with the fetched AQI value and add the
corresponding marker and circles
        setState(() {
          _aqi3 = aqi3Value;
          _addAQI3Marker(_aqi3Location, _aqi3);
```

```
          _addAQICircles3(_aqi3Location, _aqi3);
        });
      }
    });
  }

  // Fetch the latest AQI value for AQI4 node from the Firebase database
  void _fetchLatestAQI4Value() {
    _databaseReferenceAQI4.onValue.listen((event) {
      if (event.snapshot.value != null) {
        // Extract the AQI data from the database snapshot
        final aqiData = event.snapshot.value as Map<dynamic, dynamic>? ?? {};
        final mostRecentDate = aqiData.keys.last;
        final dateData = Map<dynamic, dynamic>.from(aqiData[mostRecentDate] as
Map);
        final mostRecentTime = dateData.keys.last;
        final lastEntryData = Map<dynamic,
dynamic>.from(dateData[mostRecentTime] as Map);
        final aqi4Value = (lastEntryData['AQI']?.toDouble() ?? 0.0).round();

        // Update the state with the fetched AQI value and add the
corresponding marker and circles
        setState(() {
          _aqi4 = aqi4Value;
          _addAQI4Marker(_aqi4Location, _aqi4);
          _addAQICircles4(_aqi4Location, _aqi4);
        });
      }
    });
  }

  // Add AQI circles for AQI1 node
  void _addAQICircles1(LatLng position, int? aqiValue) {
    _addAQICircles(position, aqiValue);
  }

  // Add AQI circles for AQI2 node
  void _addAQICircles2(LatLng position, int? aqiValue) {
    _addAQICircles(position, aqiValue);
  }

  // Add AQI circles for AQI3 node
```

```
void _addAQICircles3(LatLng position, int? aqiValue) {
  _addAQICircles(position, aqiValue);
}

// Add AQI circles for AQI4 node
void _addAQICircles4(LatLng position, int? aqiValue) {
  _addAQICircles(position, aqiValue);
}

// Add AQI circles to the map based on the AQI value and position
void _addAQICircles(LatLng position, int? aqiValue) {
  // Define the radii for the circles
  final List<double> radii = [75];
  // Get the color based on the AQI value
  final aqiColor = getAQIColor(aqiValue);
  // Iterate over each radius and create a circle
  for (var radius in radii) {
    final circle = Circle(
      circleId: CircleId('aqi_${aqiValue}_circle_$radius'),
      center: position,
      radius: radius,
      fillColor: aqiColor.withOpacity(0.5),
      strokeColor: aqiColor,
      strokeWidth: 1,
    );
    // Update the state by adding the circle to the set of circles
    setState(() {
      _circles.add(circle);
    });
  }
}

// Add AQI marker for AQI1 node
void _addAQI1Marker(LatLng position, int? aqiValue) {
  _addMarker(position, aqiValue, 'AQI1');
}

// Add AQI marker for AQI2 node
void _addAQI2Marker(LatLng position, int? aqiValue) {
  _addMarker(position, aqiValue, 'AQI2');
}
```

```
// Add AQI marker for AQI3 node
void _addAQI3Marker(LatLng position, int? aqiValue) {
  _addMarker(position, aqiValue, 'AQI3');
}

// Add AQI marker for AQI4 node
void _addAQI4Marker(LatLng position, int? aqiValue) {
  _addMarker(position, aqiValue, 'AQI4');
}

// Add AQI marker to the map based on the AQI value, position, and node ID
void _addMarker(LatLng position, int? aqiValue, String nodeId) {
  // Generate a unique marker ID based on the AQI value and position
  final markerId = MarkerId('aqi_${aqiValue}_${position.toString()}');
  // Get the color based on the AQI value
  final circleColor = getAQIColor(aqiValue);
  // Add the marker to the set of markers using the label_marker package
  _markers.addLabelMarker(
    LabelMarker(
      label: 'AQI: $aqiValue',
      markerId: markerId,
      position: position,
      backgroundColor: circleColor,
      textStyle: TextStyle(
        color: Colors.white,
        fontSize: 14,
        fontWeight: FontWeight.bold,
      ),
      onTap: () {
        _onMarkerTapped(nodeId);
      },
    ),
  ).then((value) => setState(() {}));
  // Show the marker info window
  _mapController?.showMarkerInfoWindow(markerId);
}

// Handle marker tap event
void _onMarkerTapped(String nodeId) {
  // Invoke the callback function with the selected node ID
  widget.onAQINodeSelected(nodeId);
  // Delay execution by 1 second
```

```
    Future.delayed(const Duration(seconds: 1), () {
      // Check if the current tab index is less than the total number of tabs
      if (widget.tabController.index < widget.tabController.length - 1) {
        // Animate to the next tab
        widget.tabController.animateTo(widget.tabController.index + 1);
      }
    });
  }

  // Get color based on AQI value
  Color getAQIColor(int? aqiValue) {
    if (aqiValue == null) return Colors.grey;
    if (aqiValue <= 50) return Colors.green;
    else if (aqiValue <= 100) return Colors.yellow;
    else if (aqiValue <= 150) return Colors.orange;
    else if (aqiValue <= 200) return Colors.red;
    else if (aqiValue <= 300) return Colors.purple;
    else return maroonColor;  // maroonColor for very high AQI values
  }

  @override
  Widget build(BuildContext context) {
    return Stack(
      children: <Widget>[
        // Google Map widget
        GoogleMap(
          initialCameraPosition: CameraPosition(
            target: _defaultLocation,
            zoom: 16.0,
          ),
          onMapCreated: (GoogleMapController controller) {
            // Store the map controller when the map is created
            _mapController = controller;
          },
          markers: _markers.toSet(),
          circles: _circles.toSet(),
        ),
        // Positioned widget to display the legend at the bottom right corner
        Positioned(
          right: 10,
          bottom: 10,
          child: _buildLegend(),
```

```
        ),
      ],
    );
  }

  // Build the legend widget
  Widget _buildLegend() {
    return Container(
      padding: EdgeInsets.all(8),
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(12),
        boxShadow: [BoxShadow(color: Colors.black26, blurRadius: 4)],
      ),
      child: Column(
        mainAxisSize: MainAxisSize.min,
        crossAxisAlignment: CrossAxisAlignment.start, // Align children to the
left
        children: <Widget>[
          _buildLegendItem(Colors.green, 'Good [0-50]'),
          _buildLegendItem(Colors.yellow, 'Moderate [51-100]'),
          _buildLegendItem(Colors.orange, 'Unhealthy for Sensitive Groups
[101-150]'),
          _buildLegendItem(Colors.red, 'Unhealthy [151-200]'),
          _buildLegendItem(Colors.purple, 'Very Unhealthy [201-300]'),
          _buildLegendItem(maroonColor, 'Hazardous [301-500]'),
        ],
      ),
    );
  }

// Build a single legend item with color and text
  Widget _buildLegendItem(Color color, String text) {
    return Row(
      mainAxisSize: MainAxisSize.min, // Limit the Row to its minimum size
      children: <Widget>[
        // Display a colored circle icon
        Icon(Icons.circle, color: color, size: 16),
        const SizedBox(width: 8),
        // Display the legend text
        Text(
          text,
```

```
          style: const TextStyle(fontSize: 12, fontWeight: FontWeight.w200),
        ),
      ],
    );
  }

  @override
  void dispose() {
    // Cancel the database listeners when the widget is disposed to avoid
memory leaks
    _databaseReferenceAQI1.onValue.drain();
    _databaseReferenceAQI2.onValue.drain();
    _databaseReferenceAQI3.onValue.drain();
    _databaseReferenceAQI4.onValue.drain();
    super.dispose();
  }

  // Maroon color for very high AQI values
  static final Color maroonColor = const Color(0xFF800000);
}
```

AQI Calculation Methodology

Explanation of AQI Calculation

The AQI calculation within our dashboard utilizes a series of methods to determine the air quality index from individual pollutant concentrations. These calculations are performed using established Emirati Air Quality Index methods which involve converting pollutant concentrations into subindex values to calculate AQI values using breakpoint formulas. For each pollutant like NO2, CO, O3, PM10, and PM2.5, specific breakpoint tables are defined in aqi_calculator.dart. The index for each pollutant is calculated by locating the concentration within these tables and applying linear interpolation to determine the AQI value. Below is the aqi_calculator.dart code.

```
// Calculates the NO2 subindex based on average concentration.
double calculateNO2Subindex(double averageNO2) {
  // Breakpoints and index ranges for NO2, based on environmental standards.
  const breakPoints = [
    {'BPlo': 0, 'BPhi': 100, 'Ilo': 0, 'Ihi': 50},
```

```
    {'BPlo': 101, 'BPhi': 400, 'Ilo': 51, 'Ihi': 100},
    {'BPlo': 401, 'BPhi': 677, 'Ilo': 101, 'Ihi': 150},
    {'BPlo': 678, 'BPhi': 1221, 'Ilo': 151, 'Ihi': 200},
    {'BPlo': 1222, 'BPhi': 2349, 'Ilo': 201, 'Ihi': 300},
    {'BPlo': 2350, 'BPhi': 3853, 'Ilo': 301, 'Ihi': 500}
  ];

  // Iterates over the breakpoints to find the correct range for the given
averageNO2.
  for (var bp in breakPoints) {
    if (averageNO2 >= bp['BPlo']!.toDouble() && averageNO2 <=
bp['BPhi']!.toDouble()) {
      // Calculate index using linear interpolation between defined
breakpoints.
      double BPlo = bp['BPlo']!.toDouble();
      double BPhi = bp['BPhi']!.toDouble();
      double Ilo = bp['Ilo']!.toDouble();
      double Ihi = bp['Ihi']!.toDouble();
      return (((Ihi - Ilo) / (BPhi - BPlo)) * (averageNO2 - BPlo)) + Ilo;
    }
  }

  return 0.0; // Returns 0 if no matching range is found.
}

// Calculates the CO subindex based on average concentration.
double calculateCOSubindex(double averageCO) {
  // Breakpoints and index ranges for CO, similar to NO2.
  const breakPoints = [
    {'BPlo': 0, 'BPhi': 4, 'Ilo': 0, 'Ihi': 50},
    {'BPlo': 5.5, 'BPhi': 10.4, 'Ilo': 51, 'Ihi': 100},
    {'BPlo': 10.5, 'BPhi': 14.4, 'Ilo': 101, 'Ihi': 150},
    {'BPlo': 14.5, 'BPhi': 17.9, 'Ilo': 151, 'Ihi': 200},
    {'BPlo': 18.0, 'BPhi': 35.4, 'Ilo': 201, 'Ihi': 300},
    {'BPlo': 35.5, 'BPhi': 58.4, 'Ilo': 301, 'Ihi': 500}
  ];

  for (var bp in breakPoints) {
    if (averageCO >= bp['BPlo']!.toDouble() && averageCO <=
bp['BPhi']!.toDouble()) {
      // Linear interpolation formula applied to CO data.
      double BPlo = bp['BPlo']!.toDouble();
```

```
      double BPhi = bp['BPhi']!.toDouble();
      double Ilo = bp['Ilo']!.toDouble();
      double Ihi = bp['Ihi']!.toDouble();
      return (((Ihi - Ilo) / (BPhi - BPlo)) * (averageCO - BPlo)) + Ilo;
    }
  }

  return 0.0; // Default return value if CO level doesn't fit any range.
}

// Calculates the O3 subindex for ozone based on its average concentration.
double calculateO3Subindex(double averageO3) {
  // Ozone breakpoints differ due to its unique impact compared to other
pollutants.
  const breakPoints = [
    {'BPlo': 0, 'BPhi': 100, 'Ilo': 0, 'Ihi': 50},
    {'BPlo': 101, 'BPhi': 199, 'Ilo': 51, 'Ihi': 100},
    {'BPlo': 200, 'BPhi': 322, 'Ilo': 101, 'Ihi': 150},
    {'BPlo': 323, 'BPhi': 400, 'Ilo': 151, 'Ihi': 200},
    {'BPlo': 401, 'BPhi': 792, 'Ilo': 201, 'Ihi': 300},
    {'BPlo': 793, 'BPhi': 1184, 'Ilo': 201, 'Ihi': 500},
  ];

  for (var bp in breakPoints) {
    if (averageO3 >= bp['BPlo']!.toDouble() && averageO3 <=
bp['BPhi']!.toDouble()) {
      // O3 index calculation using the specified interpolation formula.
      double BPlo = bp['BPlo']!.toDouble();
      double BPhi = bp['BPhi']!.toDouble();
      double Ilo = bp['Ilo']!.toDouble();
      double Ihi = bp['Ihi']!.toDouble();
      return (((Ihi - Ilo) / (BPhi - BPlo)) * (averageO3 - BPlo)) + Ilo;
    }
  }

  return 0.0; // If no range is matched, returns 0.
}

// Calculates PM2.5 subindex, a fine particulate matter, using its average
concentration.
double calculatePM25Subindex(double averagePM25) {
  // Specific breakpoints for PM2.5 due to its health implications.
```

```
  const breakPoints = [
    {'BPlo': 0, 'BPhi': 50.4, 'Ilo': 0, 'Ihi': 50},
    {'BPlo': 50.5, 'BPhi': 60.4, 'Ilo': 51, 'Ihi': 100},
    {'BPlo': 60.5, 'BPhi': 75.4, 'Ilo': 101, 'Ihi': 150},
    {'BPlo': 75.5, 'BPhi': 150.4, 'Ilo': 151, 'Ihi': 200},
    {'BPlo': 150.5, 'BPhi': 250.4, 'Ilo': 201, 'Ihi': 300},
    {'BPlo': 250.5, 'BPhi': 500.4, 'Ilo': 301, 'Ihi': 500},
  ];

  for (var bp in breakPoints) {
    if (averagePM25 >= bp['BPlo']!.toDouble() && averagePM25 <=
bp['BPhi']!.toDouble()) {
      // PM2.5 index calculation similar to other pollutants.
      double BPlo = bp['BPlo']!.toDouble();
      double BPhi = bp['BPhi']!.toDouble();
      double Ilo = bp['Ilo']!.toDouble();
      double Ihi = bp['Ihi']!.toDouble();
      return (((Ihi - Ilo) / (BPhi - BPlo)) * (averagePM25 - BPlo)) + Ilo;
    }
  }

  return 0.0; // Returns 0 if outside any specified range.
}

// Calculates the PM10 subindex using breakpoints for coarser particulate
matter.
double calculatePM10Subindex(double averagePM10) {
  // Breakpoints for PM10, impacting health differently than PM2.5.
  const breakPoints = [
    {'BPlo': 0, 'BPhi': 75, 'Ilo': 0, 'Ihi': 50},
    {'BPlo': 76, 'BPhi': 150, 'Ilo': 51, 'Ihi': 100},
    {'BPlo': 151, 'BPhi': 250, 'Ilo': 101, 'Ihi': 150},
    {'BPlo': 251, 'BPhi': 350, 'Ilo': 151, 'Ihi': 200},
    {'BPlo': 351, 'BPhi': 420, 'Ilo': 201, 'Ihi': 300},
    {'BPlo': 421, 'BPhi': 600, 'Ilo': 301, 'Ihi': 500}
  ];

  for (var bp in breakPoints) {
    if (averagePM10 >= bp['BPlo']!.toDouble() && averagePM10 <=
bp['BPhi']!.toDouble()) {
      // PM10 index calculation uses linear interpolation to find specific
index value.
```

```
      double BPlo = bp['BPlo']!.toDouble();
      double BPhi = bp['BPhi']!.toDouble();
      double Ilo = bp['Ilo']!.toDouble();
      double Ihi = bp['Ihi']!.toDouble();
      return (((Ihi - Ilo) / (BPhi - BPlo)) * (averagePM10 - BPlo)) + Ilo;
    }
  }

  return 0.0; // Default to 0 if no matching range is found.
}
```

Data Visualization

Techniques for Displaying Data

Our dashboard leverages several techniques for visualizing air quality data effectively:

Interactive Maps: Utilizing GoogleMapsFlutter, the AQIMapScreen displays real-time AQI values at various campus locations with color-coded markers that users can interact with for more detailed information.
Graphs and Charts: The BarChartCard widget illustrates AQI trends and forecasts using bar charts, showing both historical and predicted data, which helps in visualizing how air quality changes over time.

User Interaction with Data
Interactive elements are embedded throughout the dashboard to enhance user engagement. Users can tap on map markers to view specific AQI details or interact with graphs to explore different time points. Tooltips and expandable chart details provide further insights into the data presented.

Health Advisory Tab
Logic for Generating Health Advisories
Health advisories are dynamically generated based on the current AQI level. These advisories inform users about potential health impacts and suggested actions to take at different AQI levels. The logic in the HealthAdvisory widget assesses the AQI and uses predefined health

messages and activity recommendations that correspond to different AQI categories, from 'Good' to 'Hazardous'.

Customization Based on AQI Levels

The advisories are customized not just in content but also in presentation, with color coding that matches the AQI visualization elsewhere in the app. This consistent use of color helps reinforce the severity level of the air quality and makes the advisories intuitive and immediately understandable.

Current Subindices of Pollutants

We chose to show the subindices of each pollutant instead of the current pollutant values because subindex values provide a standardized way of representing the air quality levels for different pollutants, taking into account their specific concentration ranges and health impacts. This allows for a consistent comparison and interpretation of air quality across different pollutants. Subindex values are calculated based on the health effects associated with different concentration levels of each pollutant, making it easier to understand and communicate the potential health impacts of the air quality. Moreover, the Air Quality Index (AQI) is calculated by selecting the highest subindex value among all the pollutants, providing a straightforward way to determine the overall AQI and communicate the air quality level to the public simply and understandably. Subindex values also provide a more meaningful and intuitive way of communicating air quality to the public by categorizing air quality into different levels such as "Good," "Moderate," "Unhealthy," etc. Furthermore, subindex values are often linked to regulatory standards and guidelines for air quality, allowing for easier comparison of air quality levels against these standards. Lastly, subindex values allow for historical comparison and trend analysis of air quality over time, enabling the tracking of changes in air quality levels and identification of patterns or improvements in air quality management.

For instance, the below code snippet defines a Flutter StatefulWidget called PollutantsRowWidget that displays various pollutant levels, including AQI, temperature, humidity, PM2.5, PM10, CO, NO2, and O3. The widget retrieves the data from a Firebase Realtime Database and displays it in a visually appealing manner using circular widgets (AQICircle). The main widget listens for changes in the database and updates the UI accordingly.

The PollutantsRowWidget displays three rows of circular widgets. The first row shows the temperature, AQI, and humidity values. The second row displays the subindices for PM2.5, PM10, CO, NO2, and O3 pollutants. Each circular widget displays the corresponding pollutant name, concentration, and unit of measurement.
The widget also includes a legend that explains the color coding used for different AQI ranges (good, moderate, unhealthy for sensitive groups, unhealthy, very unhealthy, and hazardous). Additionally, it displays the last update time for the data using the LastUpdateWidget.

```
class PollutantsRowWidget extends StatefulWidget { // A StatefulWidget that displays a row of pollutant information
  final String aqiNode; // The path to the AQI node in the Firebase Realtime Database


  const PollutantsRowWidget({
    super.key,
    required this.aqiNode, // The aqiNode parameter is required when creating an instance of this widget
  });


  @override
  State<PollutantsRowWidget> createState() => _PollutantsRowWidgetState();
}


class _PollutantsRowWidgetState extends State<PollutantsRowWidget> { // The state class that manages the state of the PollutantsRowWidget
  late DatabaseReference _aqiRef; // A reference to the AQI node in the Firebase Realtime Database


  @override
  void initState() {
    super.initState();
    _aqiRef = FirebaseDatabase.instance.ref(widget.aqiNode); // Initializes the database reference using the aqiNode value passed to the widget
  }
```

```
  @override
  Widget build(BuildContext context) {
    return StreamBuilder<DatabaseEvent>( // A StreamBuilder that listens for
changes in the AQI node
      stream: _aqiRef.onValue, // Listens to the onValue stream of the AQI
node for changes
      builder: (context, snapshot) {
        if (snapshot.hasError) {
          return const Center(child: Text('Error fetching data')); // Displays
an error message if there's an error fetching data
        }
        if (!snapshot.hasData || snapshot.data!.snapshot.value == null) {
          return const Center(child: Text('Loading...')); // Displays a
loading message while waiting for data
        }


        // Extracts the AQI data from the snapshot
        Map<dynamic, dynamic> aqiData = snapshot.data!.snapshot.value as
Map<dynamic, dynamic>;


        // Finds the most recent date for which data is available
        String mostRecentDate = aqiData.keys.last;


        // Finds the most recent time and data entry for the most recent date
        Map<dynamic, dynamic> dateData = aqiData[mostRecentDate] as
Map<dynamic, dynamic>;
        String mostRecentTime = dateData.keys.last;
        Map<dynamic, dynamic> lastEntryData = dateData[mostRecentTime];


        // Extracts the AQI value from the last data entry
        double aqiValue = lastEntryData['AQI']?.toDouble() ?? 0;


        return Column(
          children: [
            // First row displaying Temperature, AQI, and Humidity circles
            Row(
```

```
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                // Temperature circle on the left
                AQICircle(
                  pollutantName: "Temp", // Displays "Temp" as the pollutant
name
                  pollutantConcentration:
lastEntryData['Temperature']?.toDouble() ?? 27, // Displays the temperature
value or a default value of 27
                  size: 120, // Size of the circle
                  circleColor: Colors.blue[300], // Custom color for the
circle
                  unit: '°C', // Unit for temperature
                ),
                const SizedBox(width: 16), // Spacing between circles
                // AQI circle in the center
                AQICircle(
                  value: aqiValue, // Displays the AQI value
                  size: 160, // Size of the circle
                ),
                const SizedBox(width: 16), // Spacing between circles
                // Humidity circle on the right
                AQICircle(
                  pollutantName: "Humidity", // Displays "Humidity" as the
pollutant name
                  pollutantConcentration:
lastEntryData['Humidity']?.toDouble() ?? 12, // Displays the humidity value or
a default value of 12
                  size: 120, // Size of the circle
                  circleColor: Colors.blue[300], // Custom color for the
circle
                  unit: '%', // Unit for humidity
                ),
              ],
            ),
            const SizedBox(height: 16), // Vertical spacing
            // Second row displaying circles for PM2.5, PM10, CO, NO2, and O3
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                AQICircle(
```

```
                  pollutantName: "PM2.5", // Displays "PM2.5" as the pollutant
name
                  pollutantConcentration: lastEntryData['SI PM25']?.toDouble()
?? 0, // Displays the PM2.5 subindex value or a default value of 0
                  size: 135, // Size of the circle
                  unit: 'µg/Nm³', // Unit for PM2.5
                ),
                AQICircle(
                  pollutantName: "PM10", // Displays "PM10" as the pollutant
name
                  pollutantConcentration: lastEntryData['SI PM10']?.toDouble()
?? 0, // Displays the PM10 subindex value or a default value of 0
                  size: 135, // Size of the circle
                  unit: 'µg/Nm³', // Unit for PM10
                ),
                AQICircle(
                  pollutantName: "CO", // Displays "CO" as the pollutant name
                  pollutantConcentration: lastEntryData['SI CO']?.toDouble()
?? 0, // Displays the CO subindex value or a default value of 0
                  size: 135, // Size of the circle
                  unit: 'mg/Nm³', // Unit for CO
                ),
                AQICircle(
                  pollutantName: "NO2", // Displays "NO2" as the pollutant
name
                  pollutantConcentration: lastEntryData['SI NO2']?.toDouble()
?? 0, // Displays the NO2 subindex value or a default value of 0
                  size: 135, // Size of the circle
                  unit: 'µg/Nm³', // Unit for NO2
                ),
                AQICircle(
                  pollutantName: "O3", // Displays "O3" as the pollutant name
                  pollutantConcentration: lastEntryData['SI O3']?.toDouble()
?? 0, // Displays the O3 subindex value or a default value of 0
                  size: 135, // Size of the circle
                  unit: 'µg/Nm³', // Unit for O3
                ),
              ],
            ),
            const SizedBox(height: 16), // Vertical spacing
            _buildLegend(), // Adds the legend widget
            const SizedBox(height: 16), // Vertical spacing
```

```
            LastUpdateWidget(lastUpdateDateTime: '$mostRecentDate
$mostRecentTime'), // Displays the last update date and time
          ],
        );
      },
    );
  }

  }
```

Below code is the main.dart file of our website interface:

This code represents the main entry point and the root widget of a Flutter application designed to display air quality information. It initializes the Firebase app, including specific configurations for web platforms. The application's user interface is built using the Material Design components provided by Flutter. The MyApp widget serves as the root widget and sets up the initial theme and home page (AirQualityPage) of the application. The AirQualityPage is a stateful widget that manages the display of various air quality-related screens using a TabController and TabBarView. The screens include an AQI map, pollutant graphs, and a multifunctional AQI widget. The AirQualityPage also handles the selection of different AQI nodes (locations) and updates the displayed content accordingly. It initially shows an instruction message that prompts users to click on a node to view air quality details for that location. The instruction message is automatically hidden after 8 seconds using a timer. The app's user interface consists of an AppBar with a gradient color effect and tabs for navigating between different screens. The body of the Scaffold is a Stack containing the TabBarView and an overlay for displaying the instruction message.

Overall, this code sets up the structure and navigation for a Flutter application focused on presenting air quality data, including an AQI map, pollutant graphs, and other relevant information based on the selected location or AQI node.

```
import 'package:flutter/foundation.dart'; // Provides Flutter tools for
determining platform-specific features.
import 'package:flutter/material.dart'; // Material Design components for
Flutter.
import 'package:firebase_core/firebase_core.dart'; // Firebase plugin for
initializing Firebase.
import 'package:aus_air_quality/aqi/AQI_multifunc_widget.dart'; // Custom
widget for multifunctional AQI display.
```

```
import 'package:aus_air_quality/widgets/newaqimap.dart'; // Widget for
displaying the AQI map.
import 'package:aus_air_quality/widgets/pollutants_graph_widget.dart'; //
Widget for displaying graphs of pollutants.

// Entry point of the application.
void main() async {
  // Ensures proper initialization of Flutter bindings before executing the
app.
  WidgetsFlutterBinding.ensureInitialized();

  // Initialization for Firebase specifically tailored for web platforms.
  if (kIsWeb) {
    await Firebase.initializeApp(
      options: const FirebaseOptions(
        apiKey: "AIzaSyCucMN2_A3kwS8S6xJSb-AtzIZ7HB2JBjY",
        authDomain: "ausairquality-2aa07.firebaseapp.com",
        databaseURL:
"https://ausairquality-2aa07-default-rtdb.asia-southeast1.firebasedatabase.app
",
        projectId: "ausairquality-2aa07",
        storageBucket: "ausairquality-2aa07.appspot.com",
        messagingSenderId: "723795756675",
        appId: "1:723795756675:web:411057a25c31abfb9dbdb6",
        measurementId: "G-PLJL5K1MHW",
      ),
    );
  }

  // General Firebase initialization for all platforms.
  await Firebase.initializeApp();
  runApp(const MyApp());
}

// Root widget of the application.
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Air Quality App', // Application title.
```

```
      theme: ThemeData(
        primarySwatch: Colors.blue, // Primary color theme of the app.
      ),
      home: const AirQualityPage(), // Sets the home page of the app.
    );
  }
}

// Stateful widget that handles the display of the air quality pages.
class AirQualityPage extends StatefulWidget {
  const AirQualityPage({Key? key}) : super(key: key);

  @override
  _AirQualityPageState createState() => _AirQualityPageState();
}

// State class for AirQualityPage.
class _AirQualityPageState extends State<AirQualityPage> with
SingleTickerProviderStateMixin {
  late TabController _tabController; // Manages the index and animation of
tabs.
  String _selectedAQINode = 'AQI1'; // Default selected AQI node.
  bool _showInstructionMessage = true; // Controls visibility of the
instruction message.

  // Function to handle the selection of different AQI nodes when the user
clicks.
  void _handleAQINodeSelection(String selectedNode) {
    setState(() {
      _selectedAQINode = selectedNode;
    });
  }

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 3, vsync: this); // Initializes the
TabController.
    _showInstructionMessageTimer(); // Starts a timer to hide the instruction
message.
  }
```

```
  // Timer to auto-hide the instruction message after 8 seconds.
  void _showInstructionMessageTimer() {
    Future.delayed(const Duration(seconds: 8), () {
      setState(() {
        _showInstructionMessage = false; // Updates the visibility state.
      });
    });
  }

  @override
  void dispose() {
    _tabController.dispose(); // Disposes the TabController when the widget is
disposed.
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        flexibleSpace: Container(
          decoration: const BoxDecoration(
            gradient: LinearGradient(
              colors: [ // Gradient color effect for the AppBar.
                Color.fromARGB(255, 210, 51, 40),
                Color.fromARGB(255, 145, 33, 25),
              ],
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
            ),
          ),
        ),
        title: Text(
          'AUS Air Quality', // AppBar title.
          style: TextStyle(
            color: Colors.blue[50],
            fontFamily: 'Varela',
            fontSize: 30,
            fontWeight: FontWeight.w400,
            shadows: [
              Shadow(
                offset: const Offset(1.0, 1.0),
```

```
                blurRadius: 2.0,
                color: Colors.black.withOpacity(0.5),
              ),
            ],
          ),
        ),
        bottom: TabBar(
          controller: _tabController, // Tabs within the website
          tabs: [
            Tab(text: 'Current Air Quality'),
            Tab(text: 'Air Quality Forecasting'),
            Tab(text: 'Health Advisory'),
          ],
        ),
      ),
      body: Stack(
        children: [
          TabBarView(
            controller: _tabController,
            children: [
              AQIMapScreen(
                onAQINodeSelected: _handleAQINodeSelection,
                tabController: _tabController,
              ),
              PollutantAndGraphWidget(aqiNode: _selectedAQINode),
              AQIWidget(aqiNode: _selectedAQINode),
            ],
          ),
          if (_showInstructionMessage) // Displays the initial instruction
message.
            Container(
              color: Colors.black.withOpacity(0.5), // Semi-transparent
overlay.
              child: Center(
                child: Text(
                  'Click on a node to view Air Quality details for that
location.',
                  style: TextStyle(
                    fontFamily: 'Roboto',
                    color: Colors.white,
                    fontSize: 16.0,
                    fontWeight: FontWeight.bold,
```

```
                    shadows: [
                      Shadow(
                        offset: const Offset(1.0, 1.0),
                        blurRadius: 2.0,
                        color: Colors.black.withOpacity(0.5),
                      ),
                    ],
                  ),
                  textAlign: TextAlign.center,
                ),
              ),
            ),
          ],
        ),
      );
  }
}
```

Predictive Analysis

Overview of the Predictive Model:
Our predictive model integrates with the dashboard to forecast future AQI levels. This model processes historical air quality data using machine learning techniques to predict how air quality will change at each node. Predictions are updated regularly and stored in Firebase under the ML_AQI nodes.

6.2.3 Interfaces:

6.2.3.1 ESP-WROOM-32 to Wi-Fi Interface:

The ESP-WROOM-32 MCU interfaces with the Wi-Fi module for seamless communication and transmission of air quality sensor readings. The hardware connections involve linking specific pins of the ESP-WROOM-32 MCU to those of the Wi-Fi module, establishing a Serial Peripheral Interface (SPI) connection. This includes connections for essential control pins such as Chip Select (CS), Reset, and Interrupt. The initialization phase initializes SPI communication between the ESP-WROOM-32 MCU and the Wi-Fi module. The ESP32's development environment, including the Arduino IDE or the Espressif IDF (IoT Development Framework), provides the necessary tools for this initialization. Once

initialized, the Wi-Fi module is configured to connect to the desired Wi-Fi network, requiring the SSID and passphrase. The TCP/IP stack provided by the Wi-Fi module is utilized to establish a TCP connection, ensuring reliable and ordered transmission of sensor data. The ESP-WROOM-32 MCU collects sensor readings, which are formatted into a suitable structure, often in JSON format, for transmission. The Wi-Fi module then sends this data over the established TCP connection, with error handling and acknowledgment mechanisms in place for reliability. Power management strategies are implemented to optimize power consumption, ensuring efficient use of power resources in both the ESP-WROOM-32 MCU and the Wi-Fi module.

6.2.3.2 Cloud-Server Interface:

The cloud-server interface manages the flow of air quality data from the ESP-WROOM-32 MCU to the Firebase cloud system and subsequently to the server hosting the developer dashboard. The data collection starts with the ESP-WROOM-32 gathering sensor readings for O_3, NO_2,CO, and $PM_{2.5}$/PM_{10}. The low power consumption of the ESP-WROOM-32 MCU ensures energy efficiency during this data collection phase. This data is then transmitted to the Firebase cloud through the Wi-Fi module, utilizing Firebase's Realtime Database for efficient storage. Firebase's NoSQL nature facilitates flexible and efficient data storage. Beyond storage, Firebase Cloud Functions play a vital role in triggering real-time updates on the developer dashboard. These functions enable a visual representation of the air quality index (AQI) in different locations, providing developers with live monitoring capabilities. Machine learning (ML) models for predictive analysis of AQI are seamlessly integrated into Firebase Cloud Functions, allowing for historical data analysis and the prediction of future air quality conditions. Processed and analyzed data, along with real-time information, is dynamically integrated into a user-friendly website, creating a comprehensive interface between the cloud system and the server hosting the dashboard.

6.2.3.3 Database-Cloud Storage Interface:

The dashboard-cloud storage interface facilitates the interaction between the developed dashboard and the cloud functions that interact with the model. Google Cloud functions were employed that triggered the transfer of the updated dataset from the Firebase realtime database each time new values were added to the database from any of the monitoring stations. The dataset for each station would then be transferred to Google Cloud Storage (GCS) buckets where they were stored under hierarchical folders in the JSON format.

6.2.3.4 Cloud Storage-Predictive Model Interface:

The cloud storage-prediction model interface represents the transfer of data from the GCS buckets to Google Colab notebooks that simultaneously run in the backend and hosts the predictive model used to predict forecasted values for each data point. Every 10 minutes, the function downloads the dataset from each station, converts it from JSON String to a pandas dataframe object. Then, it will compare the final row of this new dataset with that of the dataset from the previous download. If the final rows are not the same, this means that the dataset has been updated with a new set of values from a station. This new set is used as the current value in the predictive model while the past data points are used in the model's training process. A set of forecasts are generated from each new data point. Using a separate function, the forecasted values for each data point are sent to the firebase realtime, each time creating a new node that contains the values.

6.2.3.5 Dashboard-Website Interface:

The dashboard-website interface ensures a dynamic presentation of analyzed and visualized air quality data to end users. Leveraging modern front-end frameworks such as Flutter, the dashboard's processed and analyzed data is seamlessly integrated into a user-friendly website. Google Maps API enhances the interface by providing a geographic distribution of harmful target gasses. A sign-up/login page allows users to personalize their experiences, with Firebase Authentication ensuring secure user authentication and Firebase Firestore storing user information, including health data. Firebase Cloud Functions for ML model analysis, coupled with Firebase Firestore, dynamically display gas concentration information based on the user's health data. Google Maps API further enhances the user experience by pinpointing gas concentrations on a map, using Firebase Realtime Database or Firestore to store and retrieve real-time sensor data. The website presentation acts as a vital interface, providing users with accessible and actionable insights into air quality conditions.

6.3 Integration

6.3.1 Integration of Hardware Components:

.The connection between the ESP-WROOM-32 MCU and sensors involves linking each sensor (MQ131, MICS-6814, PMS5003) to the ESP-WROOM-32 MCU using either analog or digital pins based on the specifications of each sensor. Functions are developed to

read analog or digital signals from these sensors, with the interpretation and calibration guided by the respective datasheets. These sensor functions are then integrated into the main loop of the program to enable continuous data collection.

For the Wi-Fi module, UART connections are established between the ESP-WROOM-32 MCU and the Wi-Fi module. UART communication is initialized, and the Wi-Fi module is configured to connect to the Wi-Fi network. Functions for data transmission are developed to convert sensor readings into a suitable format (e.g., JSON) for reliable UART transmission.

Additionally, a solar panel is incorporated into the system to harness sunlight and convert it into electrical energy. The solar panel is connected to a power management system, which includes the BQ24074 Solar Charger from Texas Instruments. This charger is responsible for safely and efficiently charging the system's rechargeable 3.4V lithium-ion or lithium-polymer battery using the energy generated by the solar panel. Power management strategies are implemented to optimize energy consumption, ensuring efficient use of solar energy and prolonging the battery life.

6.3.2 Integration of Software Components:

The software implementation begins with the utilization of ESP32 development environments, including the Arduino IDE or the Espressif IDF (IoT Development Framework), for coding in C++, facilitating the digital integration of sensor readings. Specific protocols outlined in each sensor's datasheet are followed to interpret raw data and convert it into concentration units.

Data transmission involves using the ESP-WROOM-32 MCU's UART connections for reliable data transmission with the Wi-Fi module. Error handling and acknowledgment mechanisms are implemented to ensure the integrity of transmitted data. Power management strategies are applied to optimize power consumption in both the ESP-WROOM-32 MCU and the Wi-Fi module.

For cloud processing, Firebase Realtime Database is implemented for efficient storage of sensor data. Firebase Cloud Functions are utilized to trigger real-time updates on the

developer dashboard, and ML models are integrated into these functions for historical data analysis and future predictions.

The dashboard is developed using Flutter, integrating Google Maps API for visualizing air quality data. Serverless functions (Google Colab or AWS Lambda) are set up for hosting ML models and serving air quality forecasts. Periodic updates are implemented for real-time information on the dashboard.

Website presentation involves creating a user-friendly website using Flutter, integrating data visualization libraries for interactive graphs. The Google Maps API is used for the geographic distribution of harmful gasses. Firebase Authentication ensures secure user access, and Firestore is employed for storing user information. Firebase Cloud Functions for ML model analysis are integrated into the website.

6.3.3 Interfaces:

The ESP-WROOM-32 to Wi-Fi interface involves establishing UART connections between the ESP-WROOM-32 MCU and the Wi-Fi module. UART communication is initialized, and the Wi-Fi module is configured for Wi-Fi connectivity. The cloud-server interface manages the data flow from the ESP-WROOM-32 MCU to Firebase Realtime Database. Firebase Cloud Functions are employed to trigger real-time updates on the developer dashboard. The dashboard-ML interface integrates ML models into Firebase Cloud Functions, displaying the ML model results on the developer dashboard. The dashboard-website interface ensures the presentation of processed and analyzed data on the user-friendly website. Data visualization libraries and the Google Maps API are utilized for interactive displays.

6.3.4 Development Process:

For the front-end framework and tools, Flutter is used for dashboard and website development. Visual Studio Code serves as the code editor, Git for version control, and Github Actions for continuous integration. Google Cloud Functions and Storage will be used for continuously storing updated data and Google Colab will be used to host, train and run the model. Anaconda Spyder, Jupyter Notebook and Google Colabs Notebooks will all be used to work on the python and JSON development with terms of the predictive model. The iterative development process follows an approach starting with a minimal viable product (MVP).

Regular code reviews, user feedback incorporation, and task tracking using GitHub Issues ensure a structured and organized development cycle.

6.4 Data Pre-Processing

6.4.1 Data Cleaning and Imputation

Addressing missing data values is a crucial step in the data preprocessing phase preceding model construction. Missing data points can significantly impact analysis and modeling outcomes during the model's training process, resulting in erroneous or inaccurate forecasts. Likewise, instances of zero values, particularly when the average value of a feature is considerably high can introduce notable complications because they falsely represent data distribution patterns during the training process and therefore can lead to faulty forecasts.

While one strategy may be to skip or omit such data points, the potential data loss from the decreased sample size and increased wastage of other valid samples can be significant. Subsequently, imputation has been employed. Imputation employs strategies that estimate values for selected data points based on some arbitrary statistical method.

For this paper, The K-Nearest Neighbors (KNN) Imputer algorithm was implemented. KNN Imputation fills in missing values based on the nearest neighbors around those values, determined by a specified parameter, k. In regression analysis, it works by taking the average value of the k-nearest points around the missing value in the feature space. Distance between points is carried out by calculating the Euclidean distance between them as shown in the formula below.

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \quad (7)$$

where d represents the distances, p_i and q_i represent the values of the ith feature of two points p and q respectively, n in this case represents the dimensionality of the feature space

6.4.2 Data Scaling and Standardization

Since the dataset is varied, orders of magnitude for each feature also vary, slowing down training speed and the time or quality of model convergence (where the model iterates during the training process to find the optimal parameters that work for the dataset). Additionally, features with larger scales pose a risk of deafening the effects of features with smaller scales unfairly skewing the parameters and predictions. This is especially important for some of the distance-based or gradient descent algorithms used in this paper including PCA, Linear and Non-linear Support Vectors, and KNN Regressors among others. Although its effect is minimal in others such as Decision Tree or Random Forests Regressors which make decisions on the relative ordering of the input feature rather than their absolute features.

While there are several types of feature scaling strategies. Data Standardization or Z-score normalization was selected for all feature scaling implementation in the program. Standardization scales the features such that each feature has a mean of 0 and a standard deviation of 1. This transformation preserves the shape of the original distribution while also being more easily interpretable as a result of the nature of the transformation.

$$z = \frac{x - \mu}{\sigma} \tag{8}$$

where x is the original feature value, μ is the mean of the feature across all samples, σ is the standard deviation of the feature across all samples and z is the standardized value, or the output

Notably, after model training, the dataset values were destandardized using the same mean and standard deviation of their corresponding features to retrieve the original values for metric evaluation. The following formula was used:

$$x = z \cdot \sigma + \mu \tag{9}$$

where z is the standardized value to be destandardized, μ is the mean of the feature across all samples, σis the standard deviation of the feature across all samples and x is the destandardized value, or the output

6.4.3 Feature Extraction and Selection

6.4.3.1 Principal Component Analysis

If the input features are interdependent on the others, the features may often contain redundant or overlapping information. Therefore, if this dataset gets too large, it will increase computational complexity and inefficiency. To mitigate this, Principal Component Analysis (PCA) is employed. PCA is a dimensionality reduction method that works for multivariate dataset. In PCA, the original features are transformed into a new set of linearly orthogonal (uncorrelated) variables called Principal Components (PCs). This is performed by identifying patterns in the feature space where data varies the most and subsequently maximizing retention of the original information from the variables through the least amount of components as possible. As a result, the PCs can be ordered by the amount of variance they explain in the dataset with the initial PCs explaining the most variance and subsequent PCs progressively less so.

6.4.3.2 Bartlett's Test

Once the features have been standardized, two tests are conducted to verify if the dataset is suitable for PCA or not. The first of these is the Bartlett Test which evaluates the intercorrelation or the correlation structure between the input variables. It does this by testing the null hypothesis that the correlation matrix of the feature space is equal to its identity matrix. Correlation matrices are developed, comprising the pairwise correlations between all pairs in the feature space. This matrix would have a size n x n with n representing the number of total features. The correlation matrix represents the linear relationships between variables in the dataset. A coefficient of 0 or close to it between two variables would indicate no linear relationship and therefore linear correlation. On the other hand, a coefficient closer to 1 indicates a strong positive relationship and a coefficient closer to -1 indicates a strong negative relation. Strong relations, either positive or negative, indicate a strong possibility of multicollinearity which warrants the need for PCA.

The identity matrix, of the same size as the computed correlation matrix, is compared against it, where the differences are calculated between corresponding elements in the matrices. The test statistic, a chi square value, is then calculated with the sum of these differences squared. A p-value is obtained by comparing the test statistic against a value

acquired by using the cumulative distribution function of the chi-square distribution with a degrees of freedom value with the formula below.

$$dof = \frac{n\,x\,(n-1)}{2} \quad (10)$$

where dof is the degree of freedom and n is the number of variables

If this p-value is less than a significance value of 0.05, then the null hypothesis is rejected. This suggests that there are significant correlations between the features. However, if the p-value is larger, it suggests that the matrices are equal and that there is no significant intercorrelation. In this case, the variables are relatively independent of each other and PCA would not be able to capture any meaningful patterns in a lesser number of variables.

6.4.3.3 Condition Number

The second test involves gauging the prevalence of multicollinearity in the input feature space. The condition number is a numerical indicator that verifies how sensitive the output of a function is to small changes in the input data. Consequently, a high condition number suggests that the features are highly correlated with each other or that feature space is multicollinear. This would make multiple input features in the dataset transmit redundant information to the models and can increase noise in the dataset, making it more difficult for the model to capture intricate patterns in each feature. As a rule of thumb, a condition number higher than 5 is generally considered to indicate some prevalence of multicollinearity that would require feature extraction.

6.4.3.4 Scree and Cumulative Explained Variance Plots

Another vital step in the feature extraction process is to determine the number of features to be retained versus the number of features to be eliminated. While PCA prioritizes the first few components, the choice of which components are statistically significant enough to be employed in the model training process can be guided by a Scree Plot. Scree Plots displays the eigenvalue of each pc in descending order, representing the amount of variance explained by each component and its statistical significance. Typically, the plot shows a noticeable drop in eigenvalues at some point, known as the “elbow” of the curve. This drop indicates the point at which adding additional components provides diminishing returns with regards to explained variance of the dataset.

6.5 Machine Learning Models

The approach to forecasting involves the prediction of the individual gas concentrations used to calculate AQI levels. Predicted AQI levels will subsequently be computed from the values of the predicted variables that are required for the AQI calculation. There are 5 variables that are required for AQI calculation which are CO, NO2, O3, PM 2.5, and PM 10. Each of these variables will act as the target feature for the predictive model. The following section underlines the multiple, diverse models employed for training and testing. The aim of this section of the implementation is to gauge the predictive performance and model convergence of each and therefore find the most optimal model based on their metrics performance, computation time, and ease of implementation and training. Other input features including other gas variables were used to further aid in these predictions. These predicted gas concentrations would then be used to forecast corresponding future AQI values for the particular location of the station.

6.5.1 Regularization Models

In MLR, the aim is to minimize the sum of squared errors between observed and target variables. However, MLR does not impose constraints on the magnitude of the coefficients assigned to each variable. As a result, the model can assign very small or even very large values to predictors, regardless of how significant the predictor is in explaining the variance in the target feature. This lack of constraint poses the risk of the model leading to overfitting when it becomes too complex, resulting in poor performances to unseen, new data. Regularization models such as Lasso and Ridge regression are extensions of MLR that introduce certain regularization penalties to both control model complexity and prevent overfitting.

6.5.1.1 Lasso Regression

Lasso, short for Least Absolute Shrinkage and Selection Operator, Regression is a type of linear regression that introduces an L1 regularization or Lasso penalty term to a standard LR function. The primary purpose of this penalty is to introduce sparsity to the coefficient estimates by penalizing the sum of the absolute value of coefficients. The equation for the L1 penalty term is shown in the equation below.

$$L1\ Term\ =\ \lambda \sum_{j=1}^{p} \left|\beta_j\right| \qquad (11)$$

where λ is the regularization parameter that controls strength of penalty, and p is num of predictor variables while β represents the coefficient term of each variable numbered by j

$$SSE\ =\ \sum_{i=1}^{n} (y_i - \widehat{y_i})^2 \qquad (12)$$

where n is the number of samples, y_i is the actual value for the i^{th} observation and $\widehat{y_i}$ is the predicted value for the same

The aim of the algorithm is to minimize the combined loss function which comprises the SSE, or the Sum of Squared Errors, an error for standard LR, and the L1 penalty term. This is performed during an optimization process where the coefficients are iteratively updated to minimize the loss function. Since the L1 penalty term is linearly proportional to the absolute values of the coefficients, smaller coefficients contribute less to the penalty term compared to relatively larger coefficients. Therefore, as the optimization process progresses, the term has the effect of shrinking smaller coefficients either to exact zero or small values close to it. As a result, the algorithm effectively commits feature selection, retaining a subset of the predictors that contribute more in the forecasts for the target feature. In this context, the regularization parameter controls the balance between two objective functions: fitting the model to the data well by minimizing the SSE but also keeping the model simple by minimizing the L1 penalty and removing less important predictors.

$$Loss\ =\ SSE\ +\ \lambda \sum_{j=1}^{p} \left|\beta_j\right| \qquad (13)$$

Since Lasso regression automatically performs feature selection, it improves upon MLR by mitigating issues related to large numbers of predictors, including the prevalence of multicollinearity and computational costs. Additionally, with redundant predictors effectively removed, Lasso regression helps simplify the model mitigating risks of overfitting by reducing likelihood of noise and outliers found in the dataset.

6.5.1.2 Ridge Regression

Ridge regression introduces an L2 regularization penalty to the LR function. Unlike lasso regression, ridge regression penalizes the sum of the squared coefficients. This leads to

coefficient shrinkage, however unlike in lasso, none of the coefficients are eliminated entirely by turning their coefficients to zero. This allows for all predictors to be retained however the shrinkage to a value close to zero effectively reduces the variability of the predictors and makes them more stable across different subsets of the dataset, when compared to MLR models with no regularization.

6.5.2 Decision Tree Regression

Decision Tree regression (DTR) is a non-parametric supervised machine learning method used for regression tasks. In this context, non-parametric refers to models which, unlike traditional regression models, neither make assumptions about the nature of the relationship between the input and target features nor assume a predefined functional form for the same. Instead, as in the case of DTR, the model recursively partitions the feature space into increasingly smaller subsets selecting input variables and split points that best separates the dataset. Split points here refer to values along the selected input variable at which a dataset can be split into multiple smaller subsets and are based on some statistical criteria such as Mean squared error (MSE) or Gini impurity among others.

The algorithm performs the partition by identifying a particular input variable and split point that maximizes the homogeneity of the consequent subsets with respect to the target variable. The aim is to end up with subsets of data that are as homogeneous as possible with regards to the target variable, allowing the algorithm to capture patterns and relationships in the dataset by fitting simple models within each subset. The partitions continue until they meet a certain threshold (e.g. maximum tree depth, maximum number of samples per leaf node, etc) with the final nodes, also called leaf nodes, in the resulting tree structure used to predict a value for the target variable.

Compared to LR models, DTR models can effectively function as non-linear models based on the patterns captured in the feature space. Additionally, they are able to do so without the need to specify complex functional forms allowing DTR models to be flexible based on the complexity of the dataset itself. Furthermore, at least relative to LR models, DTR are more robust to non-zero or non-null outliers, since the algorithm partitions the feature space based on relative ordering of feature values rather than their absolute magnitudes. While this still makes them susceptible to outliers disproportionately affecting

model performance if they differ significantly from the rest of the data, DTR performs better than most regression methods in this regard. Finally, since they can be easily parallelized across multiple cores or processors, the partitions can be performed independently for each internal or decision node, making DTR relatively more suitable for large datasets.

However, DTR models are difficult to control. The lack of definite functions to run the dataset through means that DTR models may be excessively dependent on the multiple hyperparameters that define its execution. While GSCV can aid in determining the best possible hyperparameter values, these values are limited by the range of values provided. Exhaustive trial-and-error methods would need to be employed to determine the best possible hyperparameters however this can be time-consuming and computationally costly. Additionally, these strategies would need to be potentially employed multiple times when the dataset is updated. Larger datasets as well can increase computational costs and time during hyperparameter tuning and may result in more complex or deeper decision tree structures that can lead to potential overfitting when the model effectively memorizes the dataset than learn patterns from it.

While single decision trees are able to produce relatively robust predictions against outliers or extreme values, particularly noisy or biased subsets of data at individual internal nodes may produce predictions that are not representative of the underlying data distribution. Additionally, low variability in the dataset may make it difficult for the regressor to identify meaningful splits in each subset due to its limited spread. Since the final prediction output represents an average of the predicted values from each leaf node, faulty prediction values may steer the final predicted output further away from the true value. Moreover, single decision trees are more prone to overfitting when they work on large datasets since this usually requires deeper tree structures that increases model complexity and risks the model memorizing the training data rather than capturing useful patterns.

6.5.3 Random Forest Regression

Random Forest Regression or RFR attempts to address a few of DTR's complications by employing a combination of decision trees. RFR belongs to the ensemble learning family which refers to ML techniques that combine predictions of multiple individual ML models to produce a more accurate and robust prediction. In the case of RFR, this involves the combination of multiple decision trees to produce the final output prediction. Each tree has to

be trained on a random subset of the training data and uses a random subset of the input features. RFR performs this using a technique called bootstrap aggregation (or bagging) which creates multiple random training subsets from the original dataset, with replacement. This means that some samples may appear across the different datasets multiple times and that each subset is subsequently random and diverse, displaying a variety of combinations of patterns and distributions and the different aspects of the original dataset for the model to learn from. Each tree is subsequently trained on one of these samples to produce multiple predictions for the same output data point/s. These prediction values are then averaged to produce a single output value.

6.5.4 Extreme Gradient Boosting

Similar to RFR, Extreme Gradient Boosting Regression or XGBR belongs to the family of ensemble learning methods and performs through a combination of decision trees. However, while RFR builds trees independently and then averages their results, XGBR builds trees sequentially, with each tree learning to correct the errors of the previous ones. The errors are determined by training each tree to predict the residuals, or the differences between the actual target values and the predicted of the ensemble so far. In each iteration, the subsequent tree's predictions are added to the ensemble, gradually reducing the overall error.

Additionally, XGBR implements safeguards against potential complications. For instance, XGBR adopts regularization parameters to prevent overfitting. Two of the most notable include lambda, the L2 or Ridge regularization term (see 6.5.1.2 for Ridge regression), and gamma, a hyperparameter that controls excessive tree growth if the loss reduction from the split is less than it. However, compared to RFR and previously mentioned models, XGBR is also much computationally more complex, resource-greedy. Moreover, its varied number of hyperparameters, which it is incredibly sensitive to determine its performance and quality, require significant tuning and iterations through GSCV.

6.6 Deep Learning Model Components and Elements

The DL model consists of various, diverse sets of components and elements. These are some of the

6.6.1 Activation Functions

Rectified Linear Unit (ReLu): involves a simple thresholding operation (eq 14) that adds non-linearity to the neural network. If any input is less than zero, the ReLU function returns zero. Otherwise, it returns the input as is. This introduces sparsity to the network since only a fraction of neurons may be active at one time, which can speed up execution time, and decrease redundancy.

$$f(x) = max(0, x) \quad (14)$$

6.6.2 Optimizers

Adjust or tune network hyperparameters, including weights and biases, during the training process in order to minimize the model's loss function and improve its performance The primary goal of optimizers is to find the optimal set of parameters that minimize the difference between the predicted outputs of the model and the actual targets in the training data.

Adaptive Moment Estimation (Adam): Used especially in hybrid regression models like CNN-BiLSTM

Stochastic Gradient Descent (SGD): In each epoch during training, SGD randomly selects a sample of the training data and computes the gradient of the network's loss function based on the values of the hyperparameters initially set. Some of these parameters include initial learning rate, learning rate decay, momentum and implementing Nesterov accelerated gradient (NAG). These parameters are upgraded with each epoch based on the computed gradient, with the direction of the upgrades that reduces loss for the samples.

6.6.3 Kernel Initializers

Uniform: Initializes weights of neural network layers, including Dense layers, from a uniform continuous probability distribution. Weights are usually assigned randomly but within a specific range.

He_Normal: also known as Kalming Intialization. commonly used for initializing the weights of neural network layers, particularly in deep networks. He_Normal initialization draws the initial weights from a Gaussian distribution with zero mean and variance proportional to the number of input units to the layer. This initialization method is particularly effective for layers that use rectified linear activation functions (ReLU), as it helps prevent the gradients from vanishing or exploding during training.

6.6.4 Wrappers

TimeDistributed: Crucial help during time forecasting. This wrapper around , typically convolutional or dense layers, allows the application of the layer to every timestep of the input data independently. For example, in a CNN-BiLSTM model, the TimeDistributed wrapper can be used to apply a convolutional layer to each time step of the input sequence independently. This enables the model to capture spatial features at each time step and incorporate them into the sequential processing. TimeDistributed is useful for tasks where the temporal dynamics of the data are important, such as sequence prediction or time series forecasting, and where applying the same operation across all time steps is beneficial.

6.7 Deep Learning Models

The DL-based models in this paper will consist of a variety of layers and other critical functions, this section will provide a detailed view of the model architecture in the implementation phase

6.7.1. Single Layer Perceptron

This is the simplest and most basic of the DL models therefore its architecture is also relatively basic consisting of only a single layer. Following is the description of this layer:

Input/Output Layer - Dense Layer with 1 neuron:

Will act as both the input and output layer. Since this is a fully connected layer, the number of neurons will be equal to the number of input features. Each of 5 features will be fully mapped to the single neuron. The neuron will assign weights to each feature and compute the prediction or forecast value based on the magnitude of the weights. The layer is

wrapped in a TimeDistributed wrapper which allows the model to apply each timestep independently and capture features specific to that timestep. Therefore, it is also able to capture some temporal dependencies and patterns from the dataset.

Optimizers: SGD with a learning rate of 0.1, lr decay rate of 0.001, momentum of 0.5 and enabling NAG

6.7.2. Multiple Layer Perceptron

MLP builds on the SLP model by adding layers between the input and output called hidden layers. The following architecture represents a simple MLP architecture

Input Layer: Dense Layer with 10 neurons

All input features will be mapped to each of the 10 neurons which will assign weights to them. The layer was wrapped with a TimeDistributed wrapper and has a Uniform kernel initializer added.

Hidden Layer 1: ReLu

Hidden Layer 2: Dense Layer with 10 neurons

Hidden Layer 3: ReLu

Output Layer: Dense Layer with 1 neuron

Optimizers: SGD with a learning rate of 0.1, lr decay rate of 0.001, momentum of 0.5 and enabling NAG

7. Project Management Plan

Name	Start Date	End Date	Resources
Project Initiation	Sep 04, 2023	Oct 26, 2023	
Problem Identification & Idea Development	Sep 04, 2023	Sep 16, 2023	Ahmed,Dwayne,Joel,Warda
Problem Statement & Proposed Solution	Sep 17, 2023	Sep 22, 2023	Ahmed,Dwayne,Joel,Warda
Design Objectives & Limitations	Sep 23, 2023	Sep 28, 2023	Joel,Warda
Problem Statement, Design objectives	Sep 27, 2023	Sep 27, 2023	Joel,Warda
Project Plan	Oct 03, 2023	Oct 05, 2023	Ahmed,Dwayne
Literature Review	Sep 28, 2023	Oct 26, 2023	Ahmed,Dwayne,Joel,Warda
Report Part I	Oct 09, 2023	Oct 12, 2023	
Introduction	Oct 09, 2023	Oct 12, 2023	Warda
Problem Statement, Design Objectives	Oct 09, 2023	Oct 09, 2023	Dwayne,Joel,Warda
Literature Review (Writing and Organization)	Oct 09, 2023	Oct 12, 2023	Ahmed,Dwayne,Joel,Warda
Report Part II	Oct 13, 2023	Oct 20, 2023	
Literature Review (Writing and Organization)	Oct 13, 2023	Oct 19, 2023	Joel,Warda
Requirements	Oct 13, 2023	Oct 19, 2023	Ahmed,Dwayne
Cost Estimate	Oct 13, 2023	Oct 19, 2023	Ahmed,Joel,Dwayne,Warda
Updated Project Plan	Oct 16, 2023	Oct 19, 2023	Dwayne,Warda
Project Impact	Oct 17, 2023	Oct 20, 2023	Joel,Warda
Report Part III	Oct 20, 2023	Nov 02, 2023	
System Hardware/Software	Oct 20, 2023	Nov 02, 2023	Ahmed,Joel
System Architecture	Oct 25, 2023	Nov 02, 2023	Dwayne,Warda
System Detailed Design	Oct 25, 2023	Nov 02, 2023	Ahmed,Dwayne,Joel,Warda
Report Part IV	Nov 03, 2023	Nov 12, 2023	
Verification & Validation Plan	Nov 03, 2023	Nov 12, 2023	Ahmed,Dwayne
Testing Plan	Nov 05, 2023	Nov 07, 2023	Ahmed,Dwayne,Joel
Result Analysis & Standards	Nov 03, 2023	Nov 12, 2023	Ahmed,Dwayne,Joel,Warda
Final Report Draft Submission	Nov 03, 2023	Nov 12, 2023	Joel
Final Report Submission Preparation	Nov 13, 2023	Nov 16, 2023	Ahmed,Dwayne,Joel,Warda
Final Report Submission	Nov 17, 2023	Nov 17, 2023	Joel
Presentation Preparation	Nov 18, 2023	Nov 21, 2023	Ahmed,Dwayne,Joel,Warda
Final Presentation	Nov 22, 2023	Nov 22, 2023	Ahmed,Dwayne,Joel,Warda

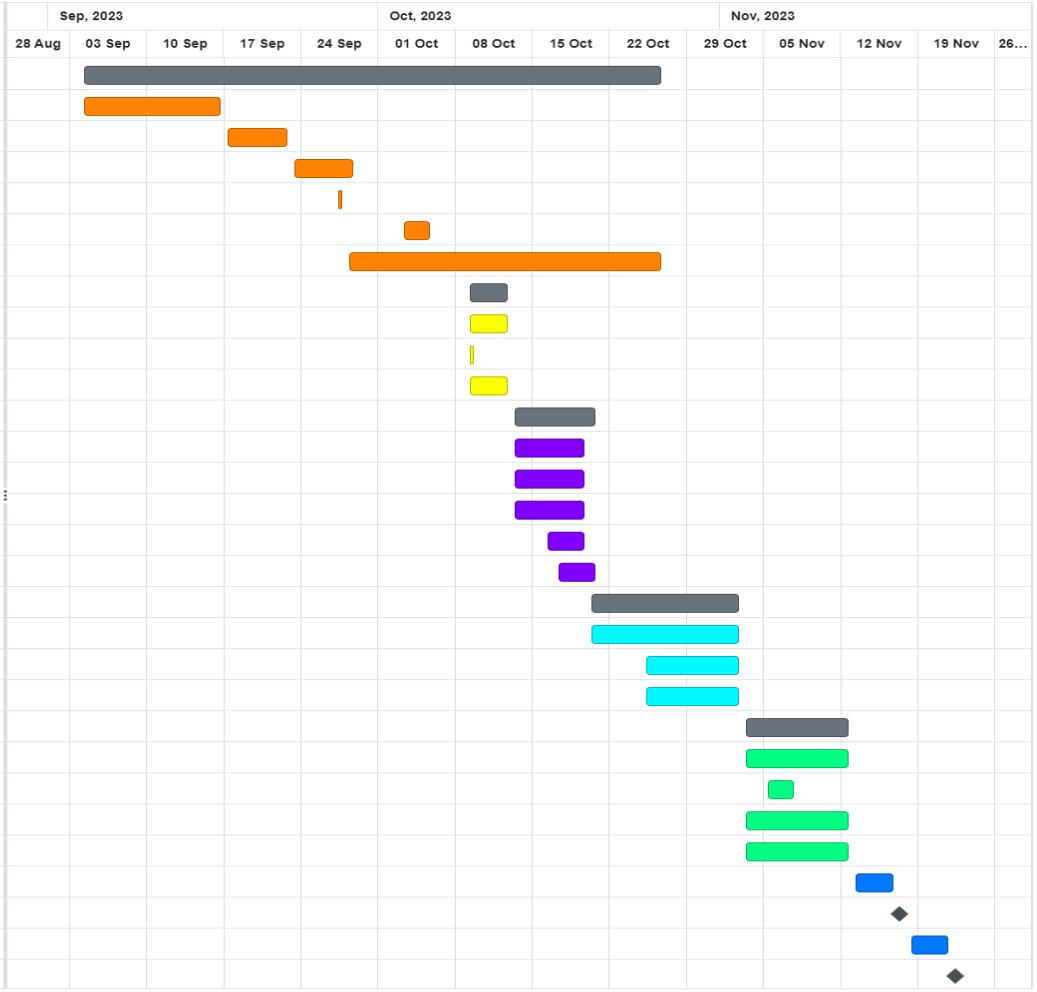
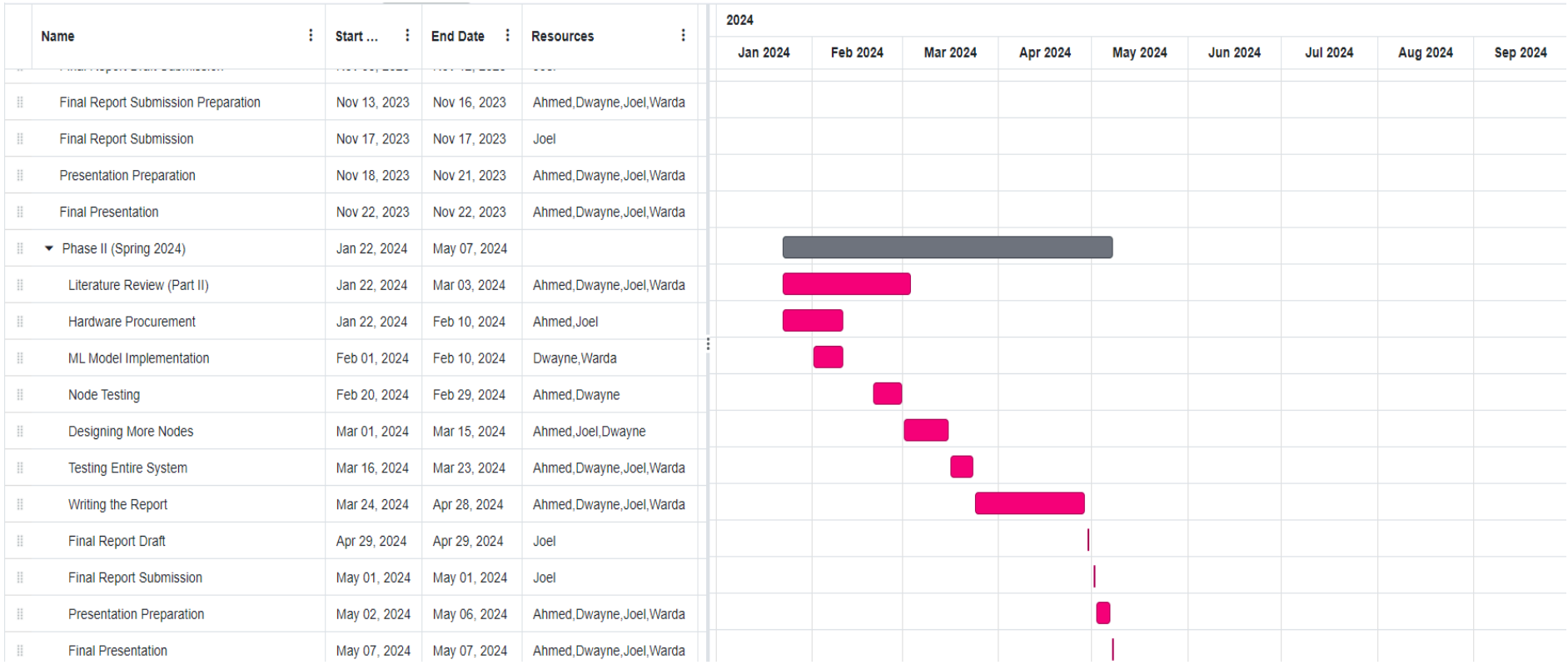


Figure 28. Phase I of the project (Fall 2023)



Name	Start ...	End Date	Resources
Final Report Submission Preparation	Nov 13, 2023	Nov 16, 2023	Ahmed,Dwayne,Joel,Warda
Final Report Submission	Nov 17, 2023	Nov 17, 2023	Joel
Presentation Preparation	Nov 18, 2023	Nov 21, 2023	Ahmed,Dwayne,Joel,Warda
Final Presentation	Nov 22, 2023	Nov 22, 2023	Ahmed,Dwayne,Joel,Warda
Phase II (Spring 2024)	Jan 22, 2024	May 07, 2024	
Literature Review (Part II)	Jan 22, 2024	Mar 03, 2024	Ahmed,Dwayne,Joel,Warda
Hardware Procurement	Jan 22, 2024	Feb 10, 2024	Ahmed,Joel
ML Model Implementation	Feb 01, 2024	Feb 10, 2024	Dwayne,Warda
Node Testing	Feb 20, 2024	Feb 29, 2024	Ahmed,Dwayne
Designing More Nodes	Mar 01, 2024	Mar 15, 2024	Ahmed,Joel,Dwayne
Testing Entire System	Mar 16, 2024	Mar 23, 2024	Ahmed,Dwayne,Joel,Warda
Writing the Report	Mar 24, 2024	Apr 28, 2024	Ahmed,Dwayne,Joel,Warda
Final Report Draft	Apr 29, 2024	Apr 29, 2024	Joel
Final Report Submission	May 01, 2024	May 01, 2024	Joel
Presentation Preparation	May 02, 2024	May 06, 2024	Ahmed,Dwayne,Joel,Warda
Final Presentation	May 07, 2024	May 07, 2024	Ahmed,Dwayne,Joel,Warda

Figure 29. Phase II of the project (Spring 2024)

8. Validation, verification, and Performance Analysis Plan

Identifying the metrics and methodologies employed to evaluate the performance of our project design, especially our machine learning model, is crucial. It enables us to gauge its standing among similar cutting-edge implementations and to make any changes and improvements to our model where and when required.

Our testing approach will involve a combination of both white-box and black-box testing. For reference, white-box testing involves assessing the internal design, logic, and structure, including code, of our project. Consequently, black-box testing involves the evaluation of the system functionality without a thorough knowledge of the internal logic.

8.1 System Hardware and Software

Verification:

1. Hardware Verification: We ensure that all the sensors (MICS 6814, DHT22, PMS5003, MQ131) and the ESP32 MCU are connected correctly. We check the wiring and the power supply. We use a multimeter to verify the voltage and current levels. This ensures that the hardware setup is correct and the sensors are powered properly.
2. Software Verification: We check the code running on the ESP32. We ensure that it's correctly reading data from the sensors and sending it to Firebase. We do this by unit testing - we simulate sensor data and check if it's correctly read and sent. This ensures that the software is functioning as expected.

Validation:

1. Hardware Validation: Once we've verified the hardware, we validate it by checking if it's able to operate in the environmental conditions of our university campus. This ensures that the hardware is robust and can withstand real-world conditions.
2. Software Validation: We validate the software by deploying it and checking if it's able to send sensor data to Firebase under real-world conditions.

Testing Plan:

1. Functional Testing: We check if the system is measuring the target gases, temperature, and humidity correctly. We also test if the data is correctly sent to Firebase and if the ML model is providing accurate predictions. This ensures that the system is functioning as intended.
2. Non-Functional Testing: This includes performance testing (how the system performs under load) and endurance testing (how it performs over a long period of time). This ensures that the system is robust, durable, and secure.

8.1.1 White-box testing:

The sensors will be tested using their software routines to verify the proper functioning of each sensor. This process using the sensor's program will involve testing their basic functionalities including power settings, as well as, their ability to read and transmit data to the microcontroller accurately and promptly.

White-box Testing for Sensors:

1. MICS-6814 Gas Sensor:

Objective: We validate the proper functioning of the MICS-6814 gas sensor and ensure accurate data transmission to the microcontroller.

Initialization Test:

We write a test script to initialize the MICS-6814 sensor and set appropriate power settings.We then verify that the sensor responds correctly to initialization commands.

```
#include <MICS6814.h>

MICS6814 micsSensor;

void setup() {

  Serial.begin(9600);

  micsSensor.begin();

}

void loop() {

  // Read gas concentration values
```

```
  float CO = micsSensor.getGas(CO_CHANNEL);

  float NO2 = micsSensor.getGas(NO2_CHANNEL);

  float NH3 = micsSensor.getGas(NH3_CHANNEL);


  // Print sensor readings

  Serial.print("CO: ");

  Serial.println(CO);

  Serial.print("NO2: ");

  Serial.println(NO2);

  }
```

2. DHT22 Temperature and Humidity Sensor:

Objective: Verify proper functioning of the DHT22 sensor for temperature and humidity measurements.

Initialization Test:

```
// Include the library for the DHT22 sensor

#include <DHT.h>


// Define the pin for the DHT22 sensor

#define DHTPIN 4

#define DHTTYPE DHT22


// Create an instance of the DHT22 sensor

DHT dht(DHTPIN, DHTTYPE);


void setup() {

  // Begin serial communication with a baud rate of 115200
```

```
  Serial.begin(115200);
  // Initialize the DHT22 sensor
  dht.begin();
}

void loop() {
  // Read temperature and humidity values
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Print sensor readings
  Serial.print("Humidity: ");
  Serial.println(humidity);
  Serial.print("Temperature: ");
  Serial.println(temperature);
}
```

3. PMS5003 Particulate Matter Sensor:

Objective: Validate proper functioning of the PMS5003 sensor for measuring particulate matter (PM2.5 and PM10).

Initialization Test:

```
// Include the library for the PMS5003 sensor
#include <PMS.h>
// Create an instance of the PMS5003 sensor
PMS pms(Serial1);
void setup() {
  // Begin serial communication with a baud rate of 115200
```

```
  Serial.begin(115200);
  Serial1.begin(115200);
}
void loop() {
  // Read particulate matter (PM2.5 and PM10) values
  if (pms.readUntil(data)) {
    float pm25 = data.PM_AE_UG_2_5();
    float pm10 = data.PM_AE_UG_10_0();
    // Print sensor readings
    Serial.print("PM2.5: ");
    Serial.println(pm25);
    Serial.print("PM10: ");
    Serial.println(pm10);
  }
}
```

4. MQ131 Ozone Sensor:

Objective: We verify proper functioning of the MQ131 ozone sensor and ensure accurate data transmission to the microcontroller.

Initialization Test:

```
// Include the library for the MQ131 ozone sensor
#include <MQ.h>

// Define the pin for the MQ131 ozone sensor
#define MQ131_PIN A0
// Create an instance of the MQ131 ozone sensor
MQ mq131(MQ_PIN, MQ131);
```

```
void setup() {
  // Begin serial communication with a baud rate of 115200
  Serial.begin(115200);
}
void loop() {
  // Read ozone concentration value
  float ozone = mq131.readOzone();
  // Print sensor reading
  Serial.print("Ozone: ");
  Serial.println(ozone);
}
```

By conducting white-box testing using these Arduino code examples, we can verify the proper functioning of each sensor and validate their accuracy for our air quality monitoring system.

8.1.2 Black-box testing:

Additionally, each sensor type will be separately tested to gauge the accuracy of sensor readings. Each sensor will be paired with an identical counterpart that will be tested in the same controlled environment to ensure consistency in test results and for a predetermined period of time. This can be compared statistically with the help of simple standard deviation tests. The threshold and accuracy of the standard deviation values will be determined through the scope of sensor data output and the repeatability of these tests.

Black-box Testing for Sensors:

1. MICS-6814 Gas Sensor:

Objective: The objective of black-box testing for the MICS-6814 gas sensor is to validate the accuracy and consistency of gas concentration readings in different environmental conditions.

Controlled Environment Testing:
In a controlled environment with known gas concentrations, we placed the MICS-6814 sensor alongside an identical counterpart. Gas concentration readings were recorded from

both sensors over a predetermined period. Comparisons between the two sensors were made to assess the consistency and accuracy of readings.

Validation: Statistical analysis, such as standard deviation tests, were employed to compare sensor data outputs and determine accuracy thresholds.

2. DHT22 Temperature and Humidity Sensor:

The goal of black-box testing for the DHT22 temperature and humidity sensor is to verify the accuracy of temperature and humidity readings under varying environmental conditions.

Environmental Testing:
The DHT22 sensor and an identical counterpart were exposed to different temperature and humidity levels in a controlled environment. Temperature and humidity readings were recorded from both sensors over time. Data analysis was conducted to assess the consistency and accuracy of readings across different conditions.

Validation: Sensor readings were compared against calibrated reference instruments to validate accuracy and assess deviation from expected values.

3. PMS5003 Particulate Matter Sensor:

Objective: The objective of black-box testing for the PMS5003 particulate matter sensor is to validate the accuracy of PM2.5 and PM10 readings in various air quality conditions.

Field Testing:
Deployments of the PMS5003 sensor and an identical counterpart were made in different locations with varying levels of air pollution. PM2.5 and PM10 concentration readings were monitored from both sensors over an extended period. Readings were compared to assess consistency and accuracy across different environments.

Validation: Collected data was analyzed and compared against reference instruments or established air quality standards to validate accuracy.

4. MQ131 Ozone Sensor:

Objective: The goal of black-box testing for the MQ131 ozone sensor is to verify the accuracy of ozone concentration readings in different atmospheric conditions.

Outdoor Testing:
Installations of the MQ131 sensor and an identical counterpart were made in outdoor locations with varying ozone levels. Ozone concentration readings were monitored from both sensors over time. Sensor performance was evaluated under different weather conditions and pollution levels.

Validation: Sensor readings were compared against established ozone concentration levels and validated for accuracy through statistical analysis.

By conducting black-box testing under various conditions, we ensure the accuracy and reliability of sensor readings for our air quality monitoring system. This validation process is crucial for providing meaningful data for predictive analysis and decision-making

8.2 Predictive Model Metrics

8.2.1 Root Mean Square Error (RMSE)

RMSE measures the average magnitude of the errors between the predicted and actual values. It is the square root of the average of the squared differences between predicted and actual values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[P_i - O_i]^2} \quad (15)$$

where n = total number of data, P_i = ML predicted values, and O_i = observed values

It is a common metric used in ML algorithms, especially in time-series forecasting models, similar to ours. RMSE is particularly suitable for gauging accuracy in regression tasks, including in predicting continuously-fed values such as in the case of air quality index measurements. A lower RMSE implies higher accuracy in the model indicated by close aligning of predicted and actual values.

8.2.2 Mean Absolute Error (MAE)

MAE is a fundamental metric in assessing the accuracy of forecasting models. Similar to RMSE, the calculation involves taking the mean of the absolute values of the difference between predicted and actual values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|P_i - O_i\right| \quad (16)$$

where n = total number of data, P_i = ML predicted values, and O_i = observed values

However, while there are similarities between RMSE and MAE, the latter is less sensitive to outliers since it does not involve squaring the differences. The comparison of both matrices provides an interpretable quantifier and comparison of the impact of larger deviations and outliers in our algorithmic output.

8.2.3 Correlation Coefficient (R)

R value measures the strength and direction of a linear relationship between the both values.

$$r = \frac{\sum_{i=1}^{n}(P_i - \bar{P})(O_i - \bar{O})}{\sqrt{\sum_{i=1}^{n}(P_i - \bar{P})^2 \sum_{i=1}^{n}(O_i - \bar{O})^2}} \quad (17)$$

R values range from -1 to 1, where -1 indicates a perfect negative linear relationship, 1 indicates a perfect positive linear relationship, and 0 indicates no linear relationship. Values in between both these extremes determine the linearity of both variables, in either direction.

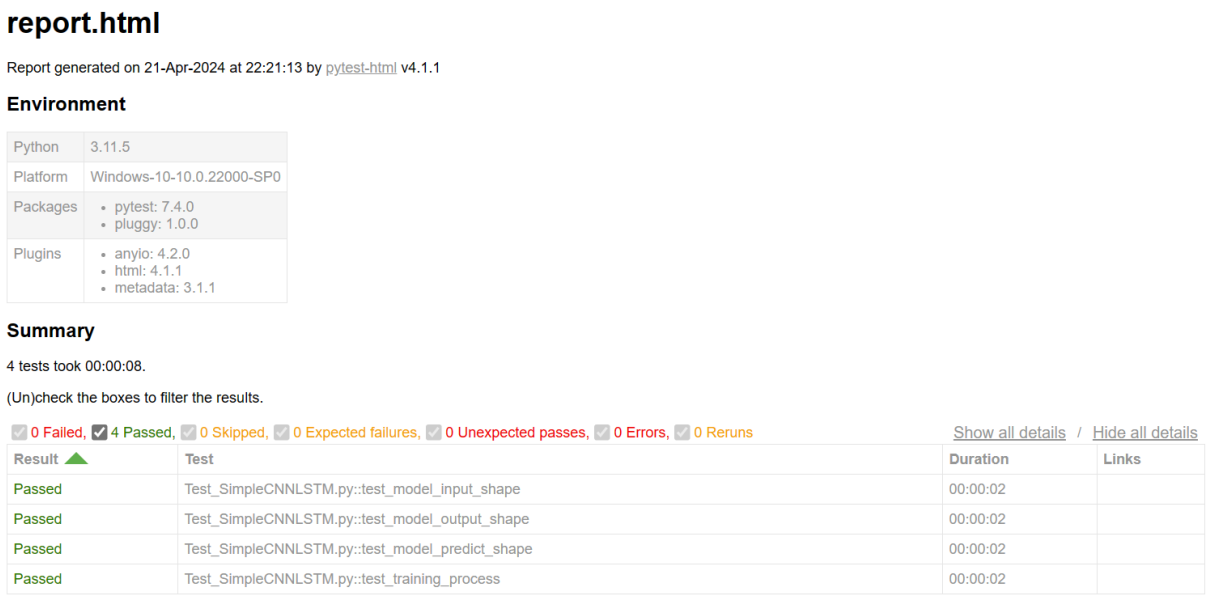
8.2.4 Coefficient of Determination (R^2)

R-squared measures the proportion of variance in the dependent variable explained by the independent variable and represents the goodness of fit of a regression model, quantifying how well the independent variable explains the variance on the dependent side. R-squared values range from 0 to 1, where 1 indicates a perfect fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(O_i - \widehat{O})^2}{\sum_{i=1}^{n}(O_i - \bar{O})^2} \quad (18)$$

8.2.5 Predictive Model Unit Testing

Unit testing was conducted on the Deep Learning model for four tests which include testing the shapes of the input and output tensors and testing the prediction function with a random sample dataset used as input with a batch size of 10. The final test validates the behavior of the deep learning model during the training process by ensuring that the loss decreases over epochs. The following depicts the results from the three unit tests.



report.html

Report generated on 21-Apr-2024 at 22:21:13 by pytest-html v4.1.1

Environment

Python	3.11.5
Platform	Windows-10-10.0.22000-SP0
Packages	• pytest: 7.4.0 • pluggy: 1.0.0
Plugins	• anyio: 4.2.0 • html: 4.1.1 • metadata: 3.1.1

Summary

4 tests took 00:00:08.

(Un)check the boxes to filter the results.

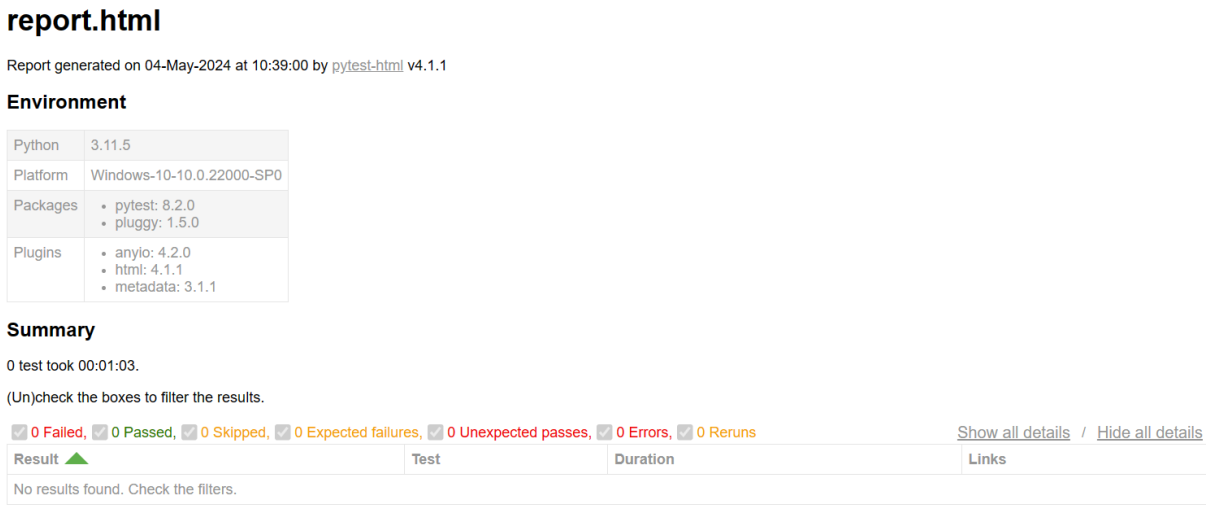
0 Failed, 4 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

Result	Test	Duration	Links
Passed	Test_SimpleCNNLSTM.py::test_model_input_shape	00:00:02	
Passed	Test_SimpleCNNLSTM.py::test_model_output_shape	00:00:02	
Passed	Test_SimpleCNNLSTM.py::test_model_predict_shape	00:00:02	
Passed	Test_SimpleCNNLSTM.py::test_training_process	00:00:02	

Figure 30: Unit Test Reports for CNN-BiLSTM Architecture

8.2.5 Predictive Model Integration Testing

Integration testing was similarly conducted which tested between components and layers in the model to ensure that they integrate and work seamlessly together.



report.html

Report generated on 04-May-2024 at 10:39:00 by pytest-html v4.1.1

Environment

Python	3.11.5
Platform	Windows-10-10.0.22000-SP0
Packages	• pytest: 8.2.0 • pluggy: 1.5.0
Plugins	• anyio: 4.2.0 • html: 4.1.1 • metadata: 3.1.1

Summary

0 test took 00:01:03.

(Un)check the boxes to filter the results.

0 Failed, 0 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

Result	Test	Duration	Links
No results found. Check the filters.			

Figure 31: Integration Test Reports for CNN-BiLSTM Architecture

8.3 Wi-Fi Testing

Our team conducted an intensive research of the AUS campus for deciding on which specific locations can be considered suitable for installing the air quality measuring nodes. Our research focuses on where strong Wi-Fi connectivity is possible around the campus so that each node can connect to the cloud through their respective Wi-Fi modules to send their sensor readings.

Below is the map of campus locations which have been preferred for installation of each node:

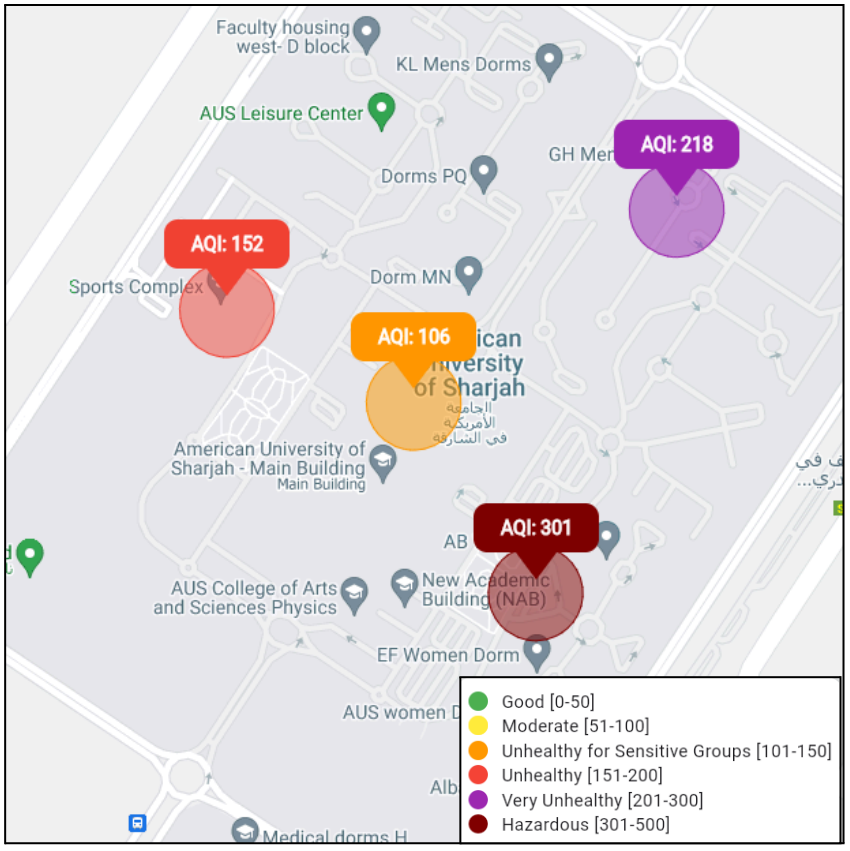


Figure 32. Nodes Placement in AUS Campus

In the provided snapshot of your website, the placement of nodes is shown. The locations for these nodes were strategically selected based on two main criteria. Firstly, the availability of a WiFi connection was considered to ensure reliable data transmission. Secondly, the nodes were intentionally distributed across different areas of the campus, rather than clustering

them together in a single location. This spacing allows for the collection of comprehensive data that covers a wider range of locations, providing a more representative overview of the entire campus.

Each node is responsible for collecting real-time measurements of various pollutants, including CO, O3, PM2.5, PM10, and NO2. These collected values are then transmitted to the Firebase Realtime Database for storage and further processing.

The Flutter application retrieves the pollutant values from the database and performs calculations to determine the current Air Quality Index (AQI) and the subindices for each pollutant. The calculated information is then displayed on the Website for users to access and view. Additionally, the Firebase Realtime Database stores these calculated values, enabling data visualization and analysis.

Based on the current pollutant values, a predictive model is employed to forecast the values for the upcoming two hours. These predicted values are also uploaded to the Firebase Realtime Database and displayed on the website, providing users with an insight into the anticipated air quality conditions in the near future.

Through our system, users can access real-time air quality information, view the current AQI and pollutant subindices, and gain an understanding of the predicted air quality for the next two hours on the Website. The strategic placement of nodes, the utilization of the Firebase Realtime Database, and the integration of a predictive model contribute to a comprehensive and informative air quality monitoring solution for the campus.

8.4 Dashboard

As mentioned our website is hosted on Firebase, and we have used Flutter to develop its functionality. The system operates by having each node transmit pollutant values to Firebase's Realtime Database every 10 minutes. Our Flutter application is designed to connect to this database, where it calculates the current Air Quality Index (AQI) and the main pollutant. Below are figures showing the structure of our data.

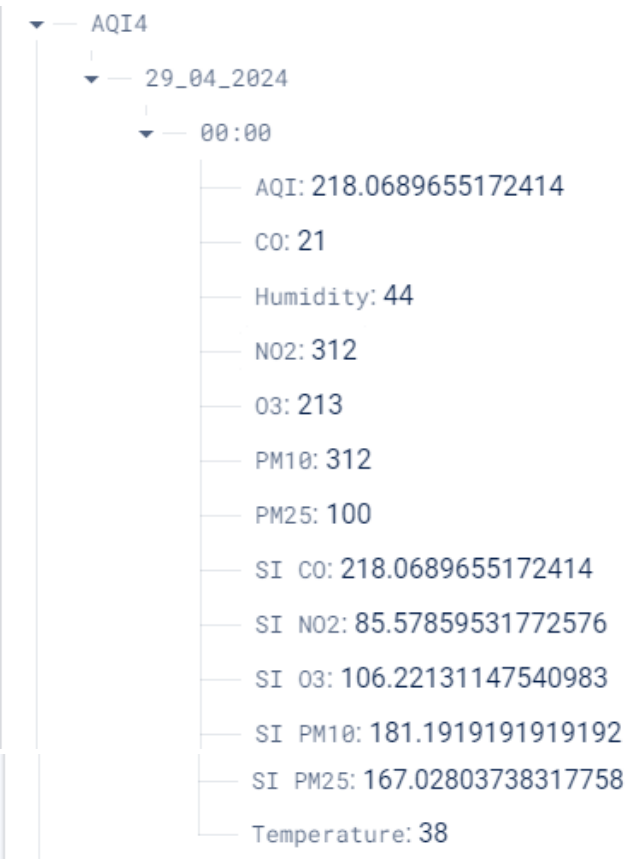


Figure 33: Data in Firebase Realtime Database

The below figure shows how mock data is uploaded to the Firebase Realtime Database to simulate node input values during the development phase.

Figure 34: Adding new pollutant, temperature, and humidity data

The process of calculating the AQI from the incoming pollutant readings is efficient: once the data is uploaded, the AQI is computed and updated in the Realtime Database instantaneously, as depicted in the figures.

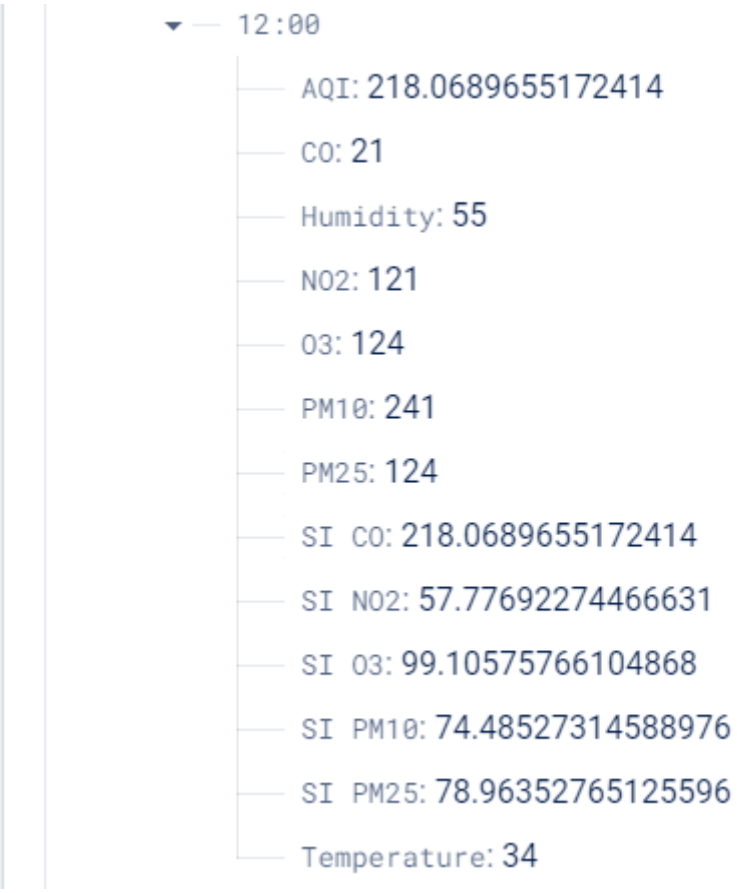


Figure 35: AQI, Pollutant subindex values updated based on the new pollutant values

The updated AQI immediately triggers changes in the user interface. For instance, the bar graph displaying both actual and predicted AQI values is updated. Additionally, information such as the pollutant concentration, temperature, and humidity levels are updated simultaneously.

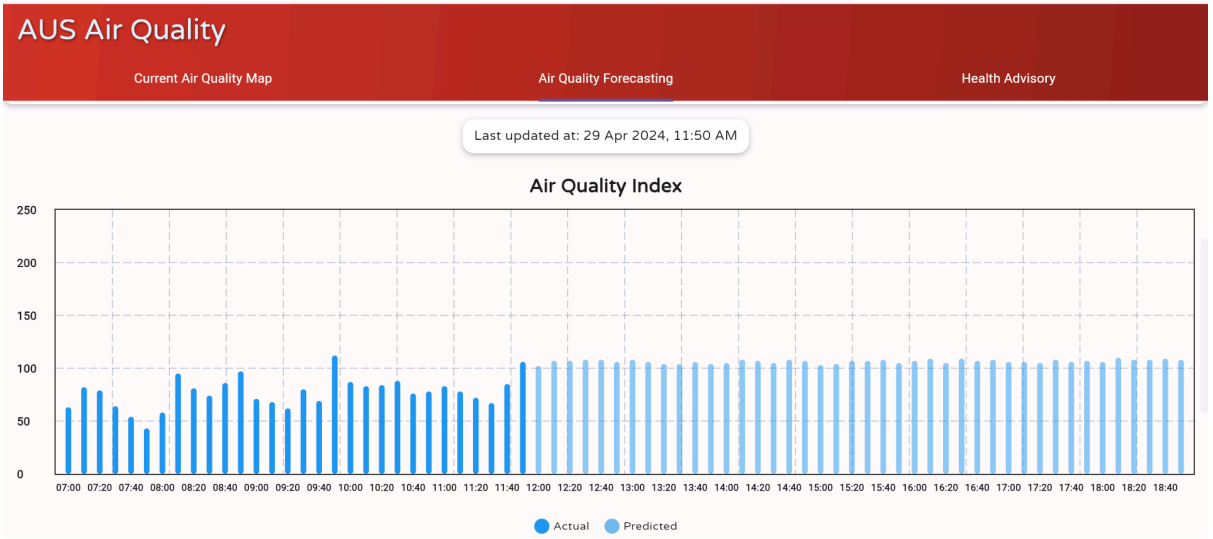


Figure 36: AQI Graph before the update of the new Pollutant values

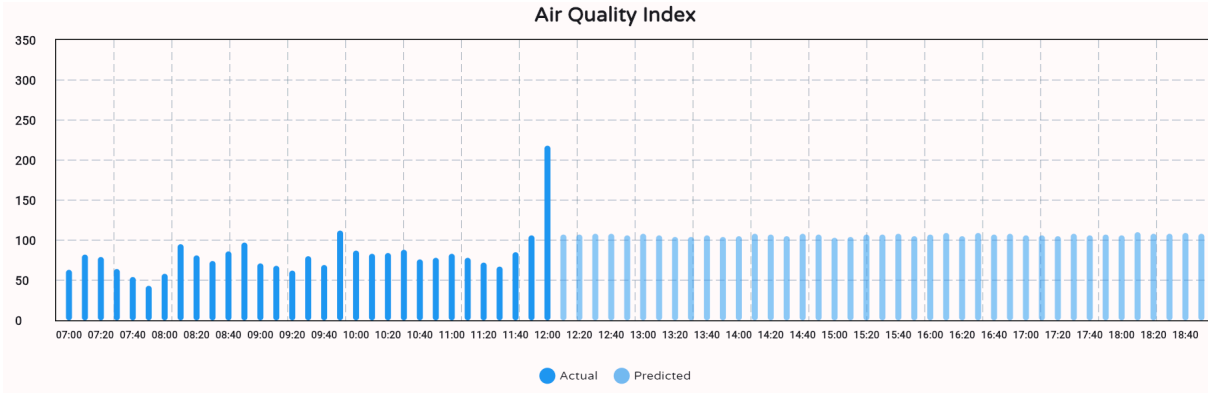


Figure 37: AQI Graph after the update of the new Pollutant values

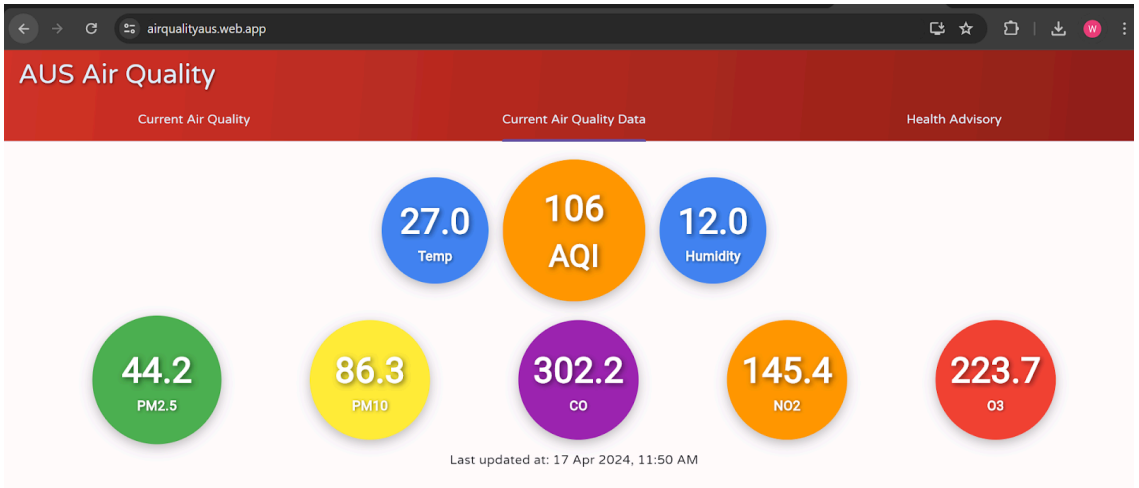


Figure 38: AQI, Pollutants Concentrations, Temperature and Humidity before the update of the new Pollutant values

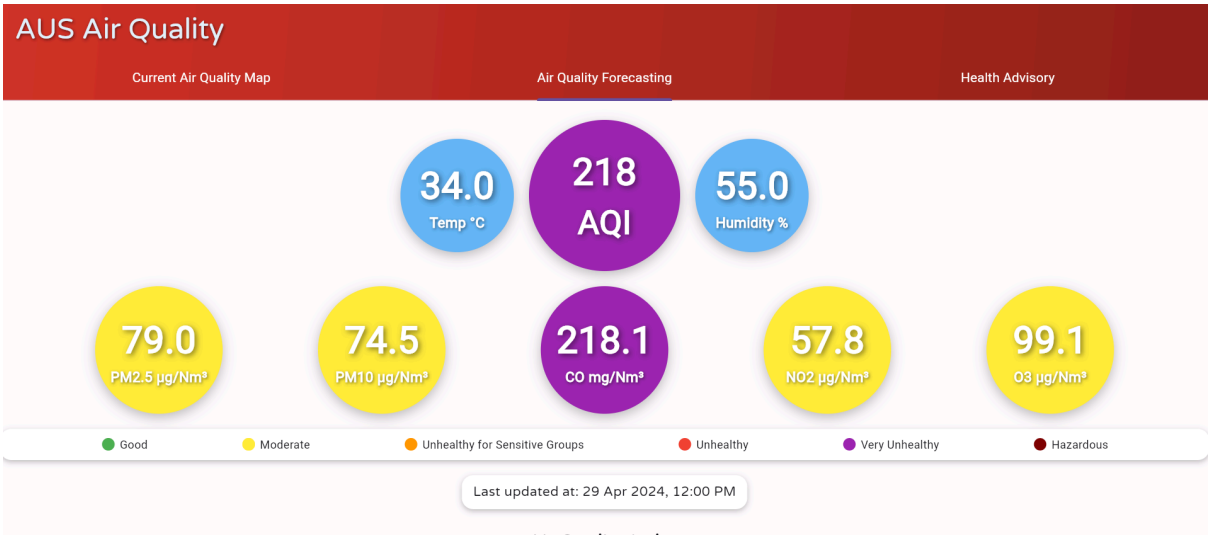


Figure 39: AQI, Pollutants Concentrations, Temperature and Humidity after the update of the new Pollutant values

These updates also extend to the Health Advisory tab, where the health advice is dynamically adjusted based on the new AQI readings. This tab changes to reflect health recommendations appropriate for one of two AQI ranges, ensuring that users receive timely and relevant health guidance based on the latest air quality data.

Figure 40: Health Advisory Tab before the update of the new Pollutant values

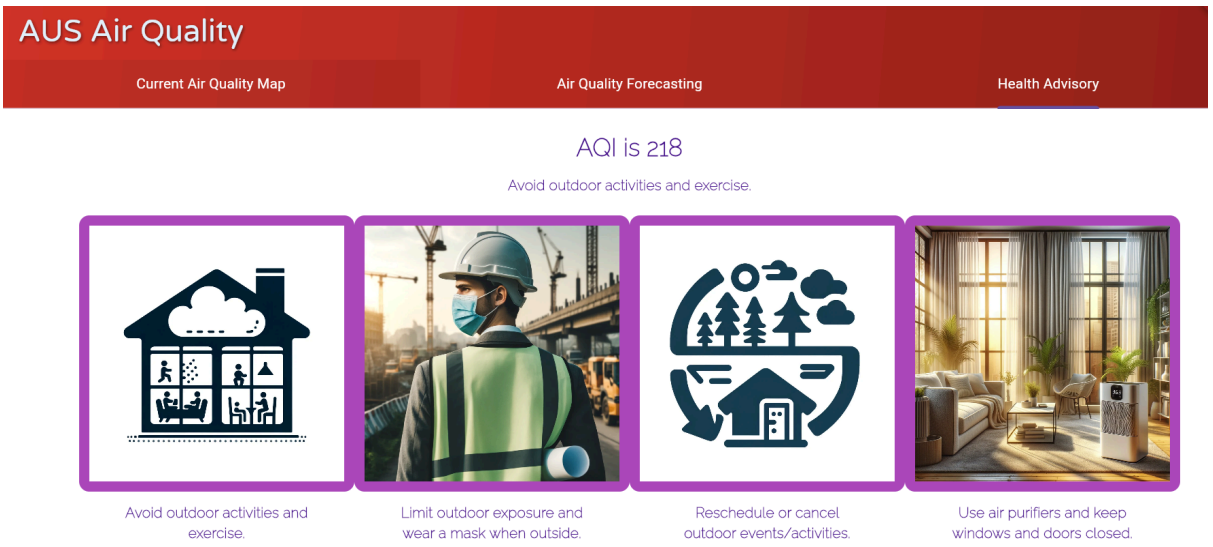


Figure 41: Health Advisory Tab after the update of the new Pollutant values

9. Results and Analysis

9.1 Dataset and Data PreProcessing

The dataset used for this analysis is sourced data from one of the monitoring stations extracted in JSON format then converted to a pandas dataframe. The figures below provide a statistical description of the dataset.

	CO	NO2	O3	PM10	PM2_5
count	1008.000000	1008.000000	1008.000000	1008.000000	1008.000000
mean	75.532113	1.478948	97.507282	108.915020	44.951895
std	8.282185	0.874813	39.349986	44.954533	18.947739
min	60.010000	0.000000	4.170000	32.270000	13.100000
25%	67.897500	0.710000	54.825000	75.452500	27.200000
50%	76.345000	1.500000	116.565000	96.640000	46.735000
75%	82.640000	2.250000	129.655000	161.140000	60.925000
max	89.960000	3.000000	139.940000	196.590000	84.460000

Figure 42: Statistical description of training/testing dataset

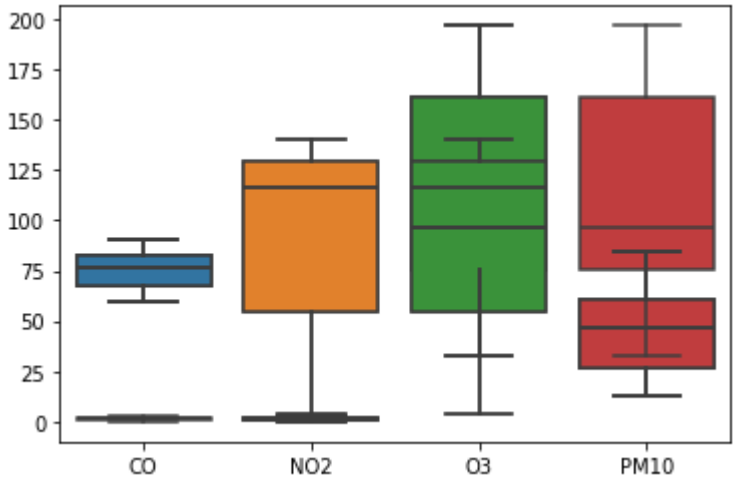


Figure 43: Box plot of the each feature

9.1.1 KNN Imputation and Standardization

The dataset was exhaustively searched for missing or zero values. The table below shows the results from this search.

Total Number of Missing Values	5	Total Number of Zero Values	10
Percentage of Missing values	0.11%	Percentage of Zero values	1.98 %

Table 4: Description of the missing and zero values in dataset for imputation

Since both missing and zero values exist within the dataset, KNN Imputation is used to fill in the values. Considering the size of the dataset samples, k was chosen at 3 that represented the most optimal value for the number of neighbors that can be used for

imputation. The data is subsequently standardized taking into account the mean and standard deviation of each feature to calculate according to the formula x.

9.1.1 Principal Component Analysis and Scree Plots

Figure 32 shows a heatmap of the correlation matrix generated from the dataset after standardization. The correlation matrix reveals the linear relationships between two variables in the dataset. The range of colors represents the strength and direction of each relationship. According to the figure, CO has a relatively strong negative correlation with O3 and a relatively weaker positive correlation with the other variables. Similarly, NO2 has a strong negative correlation with O3 and weaker relations with the others. O3 has negative relations with all the variables and a slightly more negative correlation with CO. PM 2.5 and PM 10 have near perfect correlation indicating almost complete linear dependence. Additionally, both features have a strong positive correlation with CO, weaker positive correlation with NO2 and a negative correlation with O3 like all other variables.

Since the Bartlett test requires that the correlation matrix should be equal to an identity matrix of the same size, this suggests that the null hypothesis will likely not be accepted. Nevertheless, the p-value obtained from the Bartlett test is 0.01 from a chi square or test statistic value of 886.538. Since the p-value is less than significance level of 0.05, the null hypothesis is rejected, indicating the prevalence of multicollinearity in the dataset.

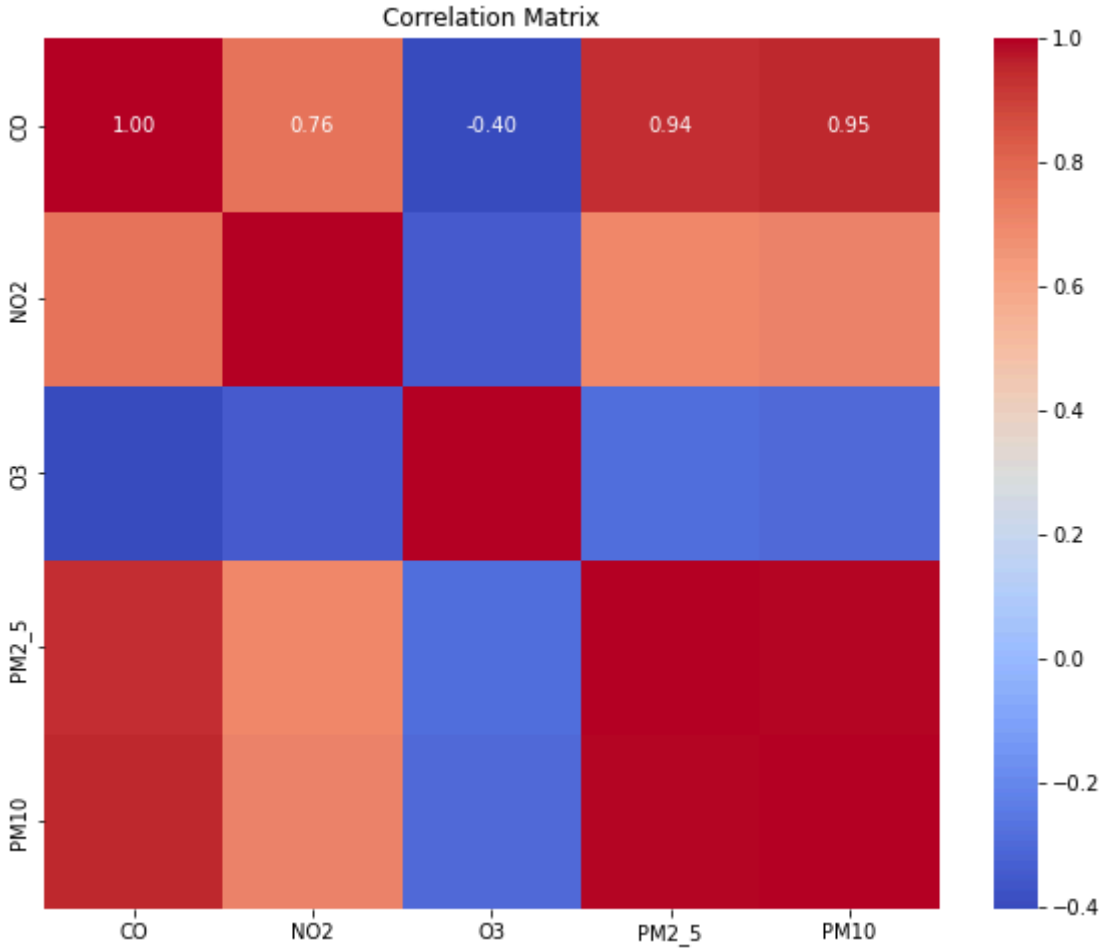
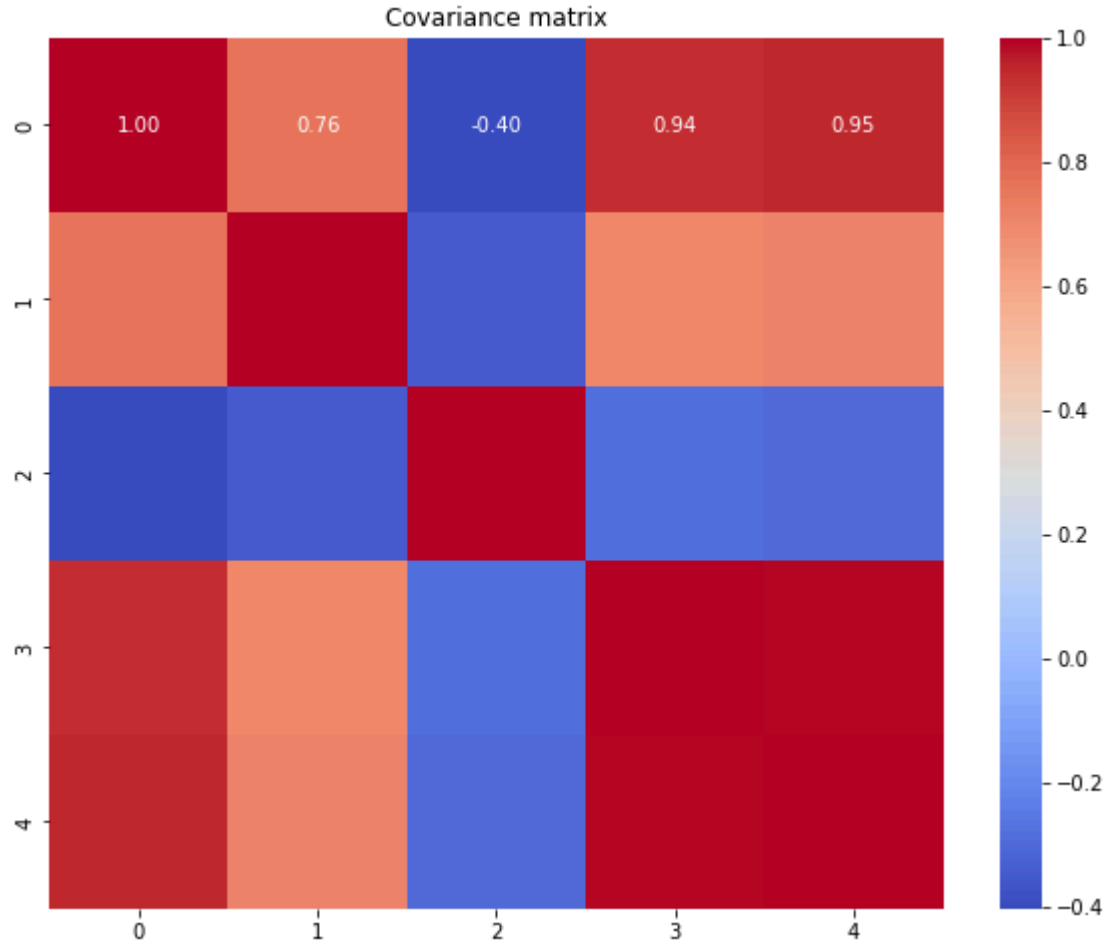


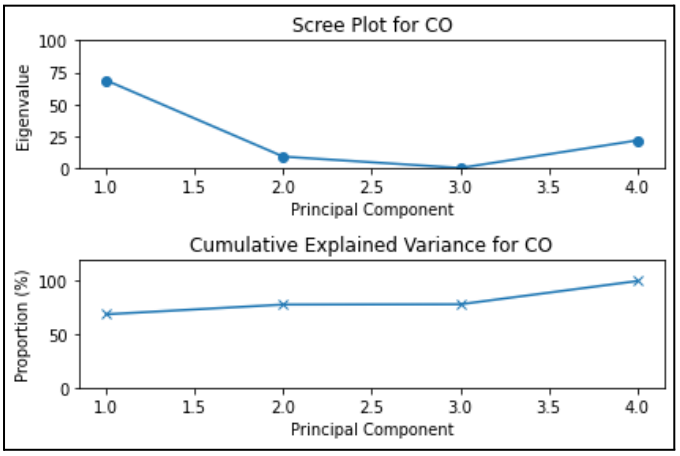
Figure 44: Correlation Matrix of the dataset

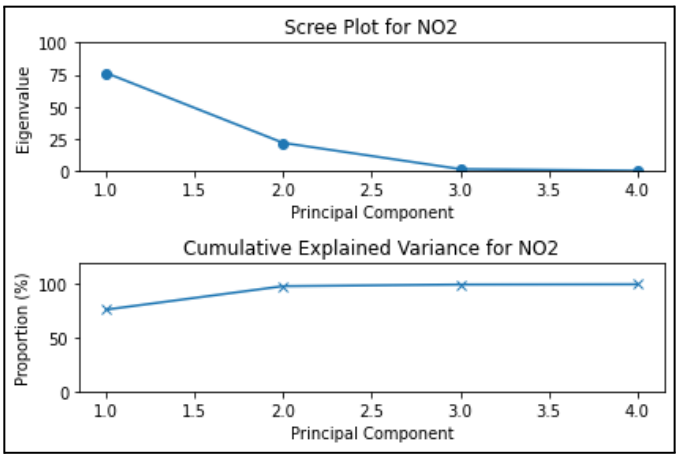
With the null hypothesis of the Bartlett's test rejected, PCA can be performed on the dataset. Subsequently, the covariance matrix of the input features are computed for each target gas. Figures x to x show heatmaps for covariance matrices for all 5 target gasses. Once this is calculated, PCA decomposes each matrix to its eigenvectors and eigenvalues.

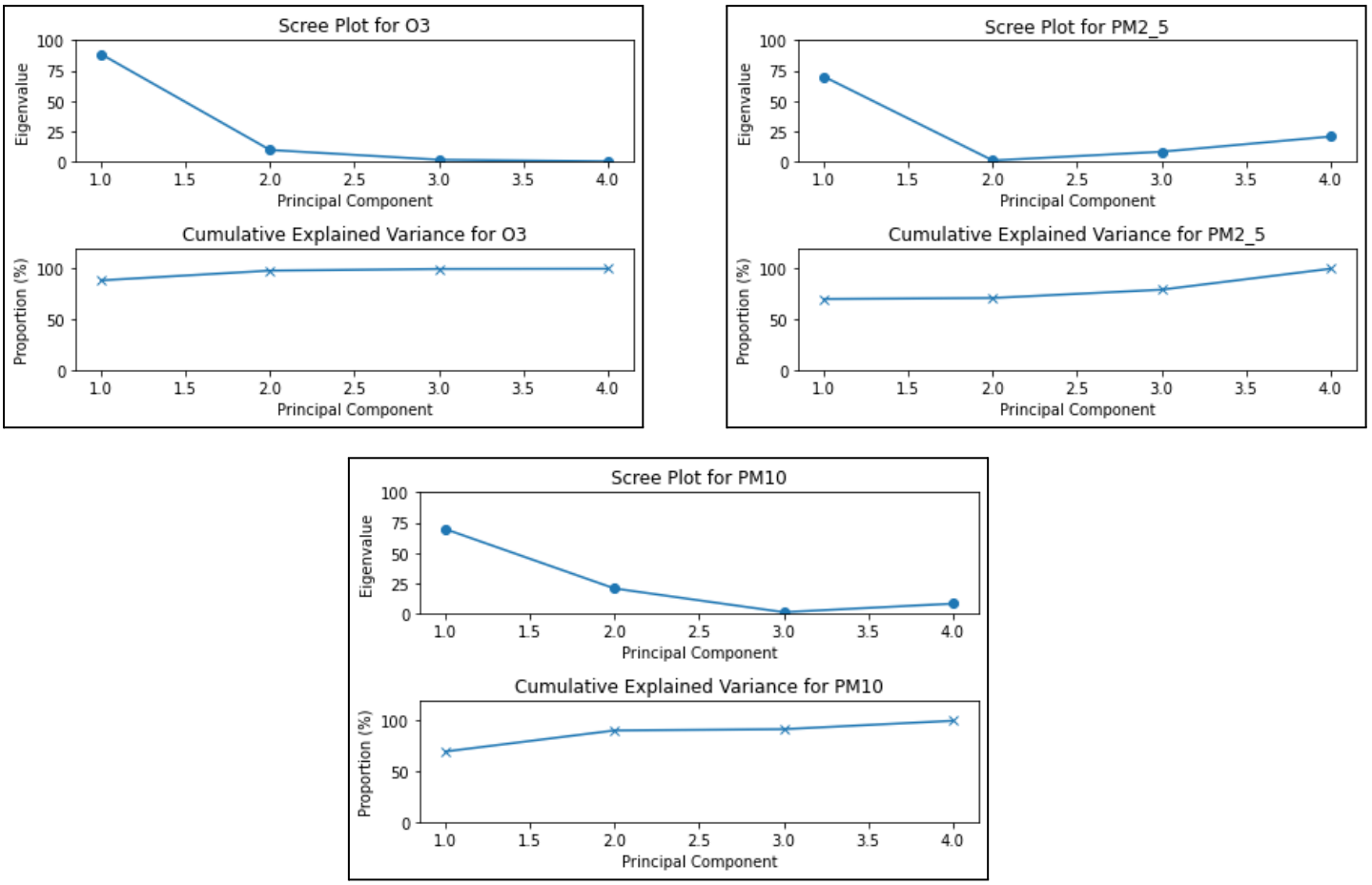


Figures 45: Covariance matrix for all target features

After PCA, we are left with principal components that now number the same as the number of input features, however, their eigenvalue ranks may vary. Taking the Scree Plot and the Cumulative Explained Variance depict which components have the larger eigenvalue and the largest contribution to the total explained variance.







Figures 46a to 46e (Clockwise from upper-left): Scree Plots and Cumulative Explained Variance curves for each target feature.

As shown in the graphs and as discussed earlier, the optimal number of components selected can be based on a point on the Scree Plot (and to some degree in CEV graphs) called the knee or elbow point where the curve takes a steep fall in eigenvalue for a range of later components. As shown in figure 34, it indicates 3 principal components show the best variance in the dataset without having to use up the remaining rather redundant features. Additionally, the CEV curve, which accumulates the eigenvalue for each component over the sum of all its components, shows 3 components to represent or explain at least 90% of the total variance in the dataset.

9.2 Model Performance and Analysis

The predictive performance of each model was evaluated on the performance metrics. In each of them, the model predicted new data points for each target gas for a total period of one day. Training past data involved samples over a period of 5 days and after every prediction of a data point, the observed values of that sample was concatenated to the training and testing data to be used by the model in its next prediction. This strategy employed is identical to how our predictive model will act upon the raw data points in our system.

Figures depict comparisons of each metric and of each target gas with the other models with the aim of providing a clear and concise evaluation of the models' performance.

Out of all variables, CO has shown to be the best performer with the models with an average R-score value at 99.53%, 98.43% for R-squared for the best working models. A likely cause for this may be due to its relationship with other variables. A look at figure 32, or the correlation matrix makes it clear that CO shares strong and positive linear relationships with most other variables which allows for the models to easily capture both linear but also spatiotemporal patterns in the CO target feature space.

DTR performed the weakest for non-DL models on all metrics for the CO and PM 10, and to some extent, PM 2.5 variables. This discrepancy was especially pronounced for the R-squared score with recorded values as low as 30.69% in CO, although similar disparities were observed across most other models, albeit to a lesser degree. While the MAE values for all other models averaged at 149.6, DTR noted a significantly higher value at 1508.24, a magnitude 10 times the difference. Results were identical with regards to RMSE where the average value without DTR was at 214.8 while DTR itself stood at a notably higher 2013.6. These trends were similar for PM 10 and PM 2.5, although MAE and RMSE values were closer to the trend in NO2 and performed slightly better than some models for O3.

The patterns from the metric values indicate that DTR failed to properly explain the proportion of the variance in most target variables when compared to the other models. The exceptionally higher MAE and RMSE scores further emphasize its poor predictive ability. While factors for this relatively poor performance can be multifold and complex, there are potential reasons that could be pointed out. As mentioned previously, while DTR models are

incredibly flexible for continuous datasets because they don't impose a functional form, they are also incredibly sensitive to small changes in the data. As shown in the boxplot in figure 31, PM 2.5, PM 10 and in particular CO, have relatively small interquartile ranges and therefore little variability or distribution in each feature. The smaller range of values limits how the DTR splits the input feature space since the split functions are based on the distributions or patterns found in their corresponding target variables and can lead to potential overfitting. Comparatively, DTR performs smoothly with outliers or extreme values which might explain the relatively improved performance for O3 with its large distribution of data, again depicted in the box plot in figure 31.

Another possible explanation for the varying performance of certain gas variables could be related to the quality of the hyperparameter tuning. It's plausible that certain hyperparameters were better tuned or optimized for some variables but not as effectively calibrated for others. While one solution to this may be to conduct more trial-and-error strategies and/or increase the range of the hyperparameters, these would likely be more computationally costly and time-consuming and are best not recommended.

On the other hand, as was theoretically expected, RFR outperformed DTR in all metrics and for all target features. RFR represents a combination of decision trees. Therefore, while it is computationally more expensive than DTR, which works with a single tree, it effectively handles issues prevalent in DT regression by further dividing and randomizing the subsets through multiple trees. This approach enhances the model's ability to capture complex relationships within the data and mitigate overfitting, resulting in improved predictive performance. Additionally, RFR significantly outperformed DTR for those variables (i.e. CO, PM 2.5, and PM 10) where the latter exceptionally underperformed for all metrics and consistently outperformed all models that were not based on deep learning techniques on multiple metrics. For example, for the R-score values for NO2, it narrowly outperformed (96.77%) both CNN (93.56%) and LSTM (96.56%) but fell just short of CNN-BiLSTM (97.21%). This indicates that the randomized interaction of variables within this model exhibits superior generalizability and greater accuracy in predicting the target features.

Notably, although it was expected to outperform RFR, XGBR slightly underperformed in almost all metrics and for all variables. Since XGBR was expected to improve on the shortcomings of RFR as the latter was for DTR, this underperformance with

XGBR might be indicative of two reasons. Firstly, the most apparent reason could be potential suboptimal hyperparameter tuning. XGBR is more sensitive to the quality of hyperparameters compared to RFR or DTR. Consequently, suboptimal parameter values risk have reduced the potential benefits that could have been accrued from XGBR. Secondly, the evidence from the correlation matrix in figure 32 shows that the variables have strong linear dependencies between them. While XGBR improves on RFR in multiple ways, it depends heavily on weak DT learners early on. If these learners fail to capture linear relationships, the XGBR would struggle to capture or leverage these in later sequences.

Multiple Linear, Lasso, Ridge and Simple Vector regressions, as shown in the graphs, performed generally on a similar level on all metrics and with all target features. Lasso and Ridge effectively act as extensions of MLR that add regularization penalties to improve its predictive performance and to handle multicollinearity and risks of overfitting. One justification for the lack of discernible discrepancies between the models, might be that the input features are not highly correlated. However, results from both the correlation and covariance matrices (figs x to x) and from the Bartlett test have shown the likely prevalence of multicollinearity in all input features. Instead, it seems likely that PCA, performed on the standardized dataset and before model training, and the subsequent feature selection may have had a significant role in eliminating at least sufficient intercorrelation in the feature space that Lasso and Ridge have negligible improvements over MLR. Overall, while all three linear regression methods performed better than DTR, their performance was generally overshadowed by other models.

KNNR depicted similar performances as the linear regression models. Its only significant performance was for O3. While it may be difficult to interpret the exact reason for this without further testing and analysis, it might be because O3 displays a more localized relationship with its variables, with underlying patterns including patterns and relationships within subsets or regions of data. This is plausible because KNNR is a locally weighted algorithm that makes predictions based on the neighbors of the test data point with the assumption that variables in a more localized area in the dataset i.e. in subsets, may better explain the variations in the target features.

For the DL models, both SLP and MLP performed the worst. This is likely a result of their simplistic architecture. Both SLP and MLP may struggle to handle the datasets, failing

against the ML models with regards to capturing linear relationships and being unable to perform against more complex DL models in capturing more non-linear dependencies.

While CNN and LSTM performed the best out of most models in all metrics and features, CNN-BiLSTM provided the most optimal performance for all metrics and gasses. In some metrics and for some variables where either CNN or LSTM would fall short, the hybrid model would perform on par with the best of the two and consistently so for all the metrics and for all variables. For example in figure 47, the hybrid model would perform better than CNN and identical to LSTM, both being the best performing of all models. On the other hand, for figure 49, LSTM would fall short in terms of performance while CNN-BiLSTM performed on par or even slightly overperformed compared to CNN, which was similarly the best among the other models.

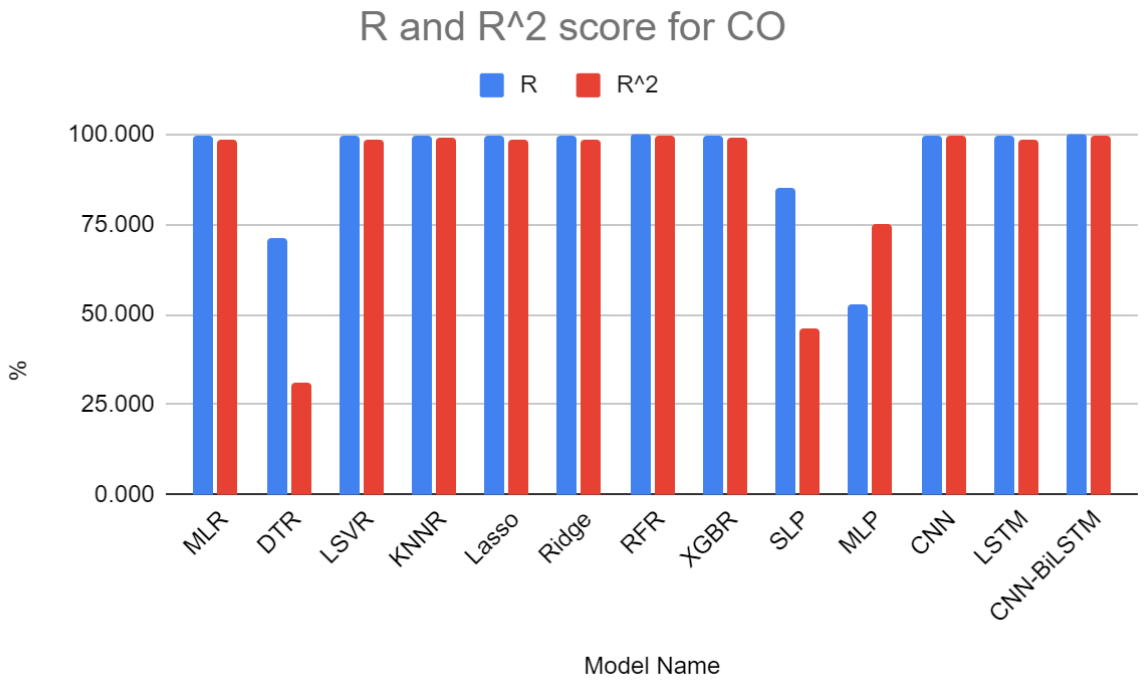


Figure 47: R and R-squared scores for CO

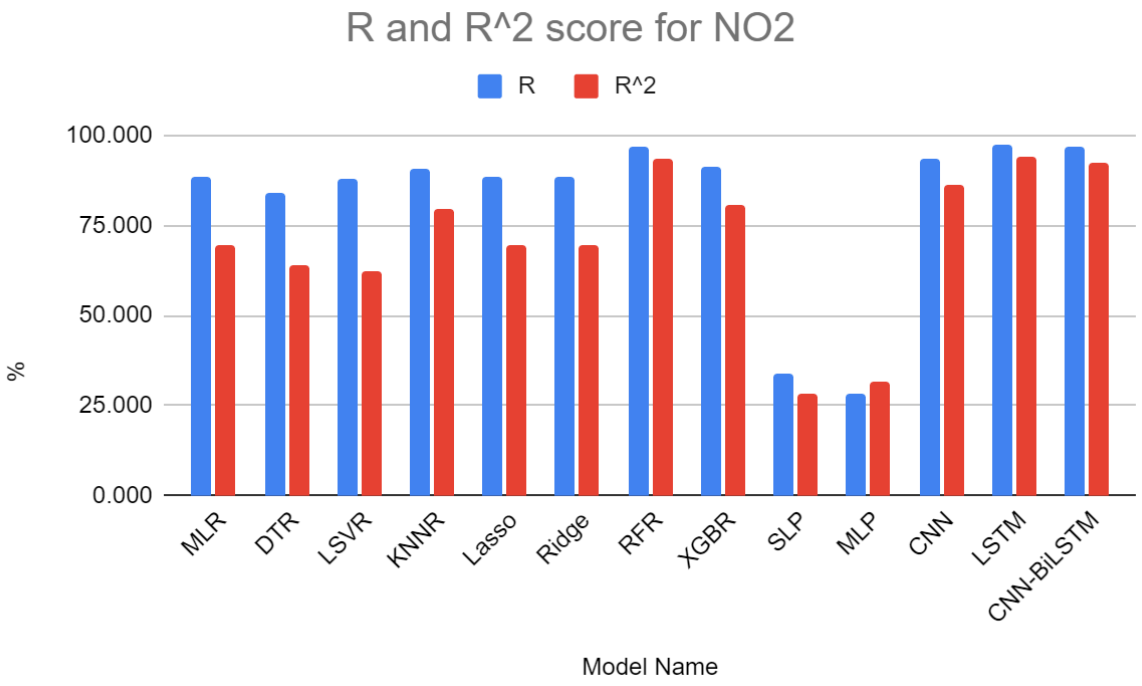


Figure 48: R and R-squared scores for NO2

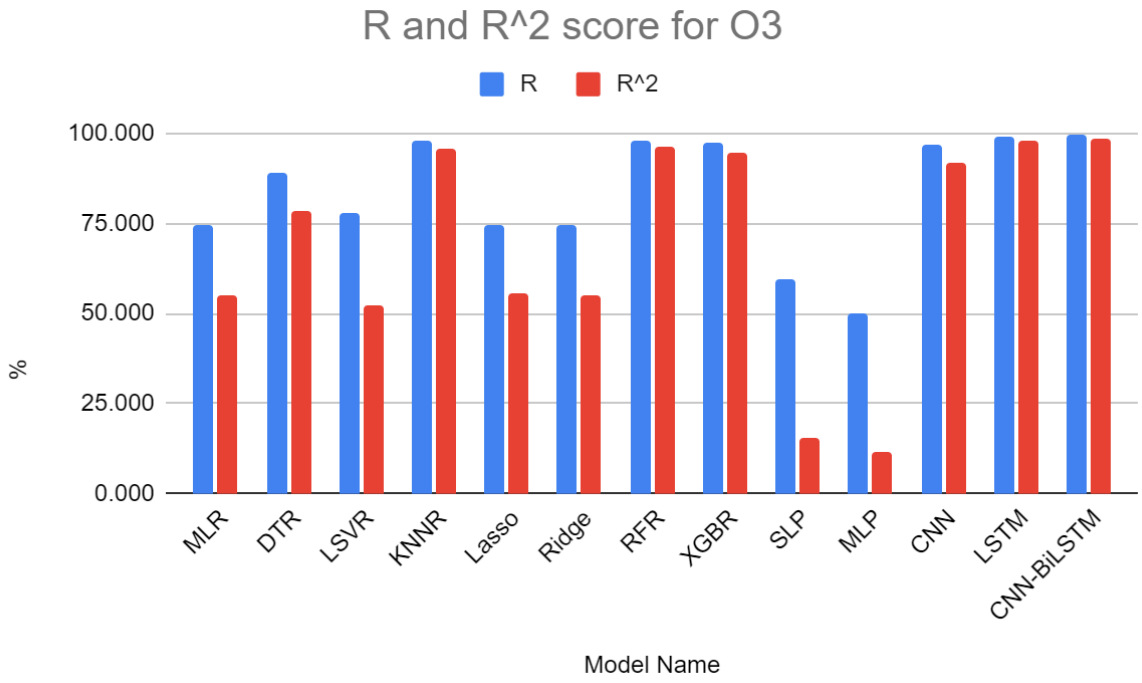


Figure 49: R and R-squared scores for O3

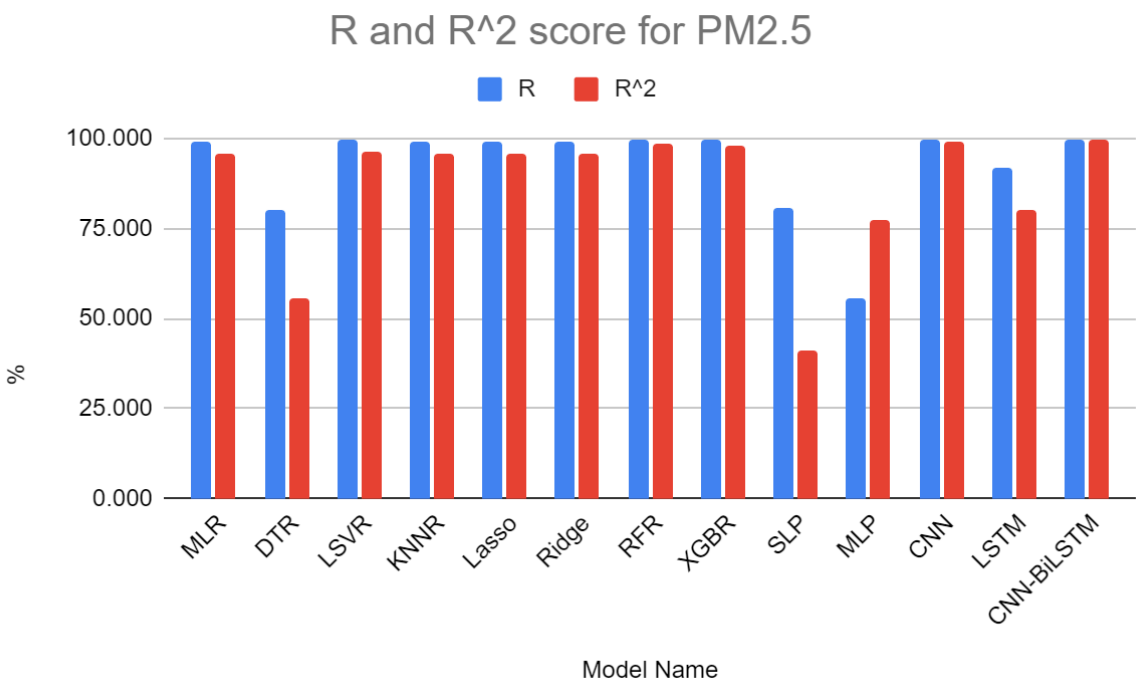


Figure 50: R and R-squared scores for PM2.5

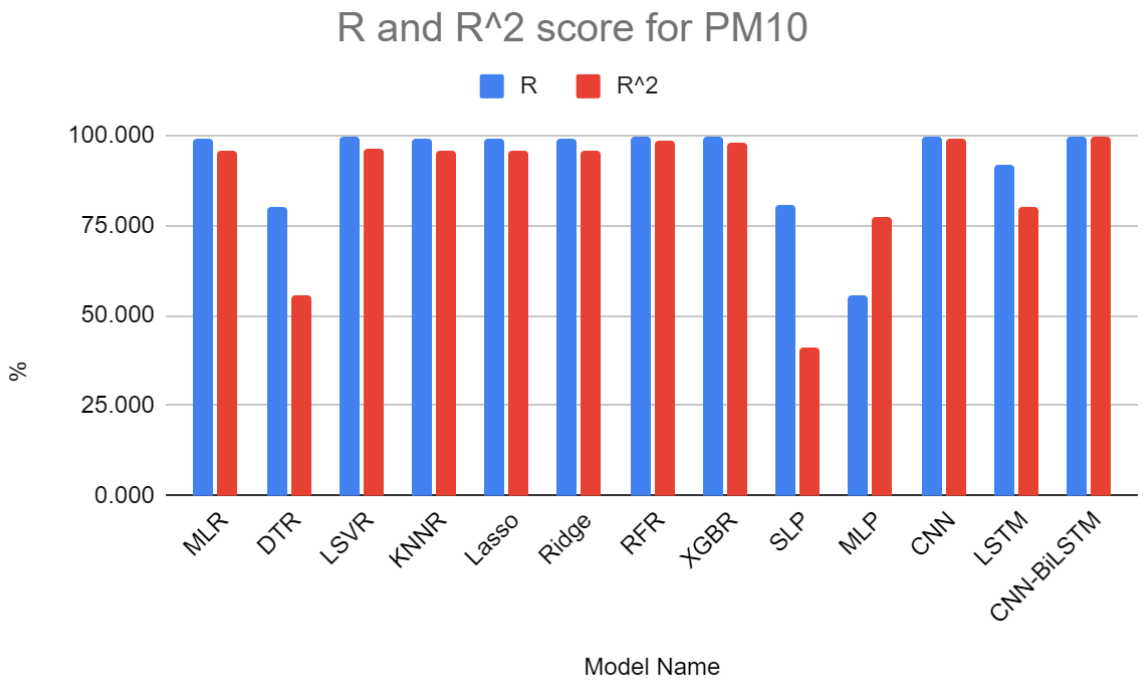


Figure 51: R and R-squared scores for PM10

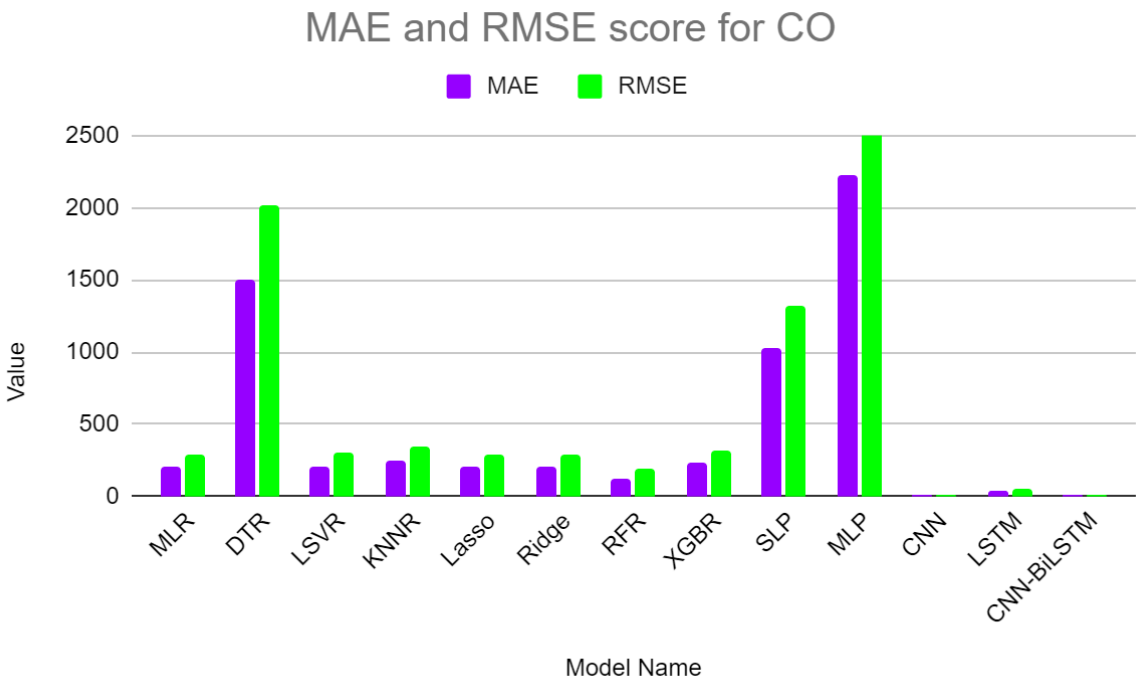


Figure 52: MAE and RMSE scores for CO

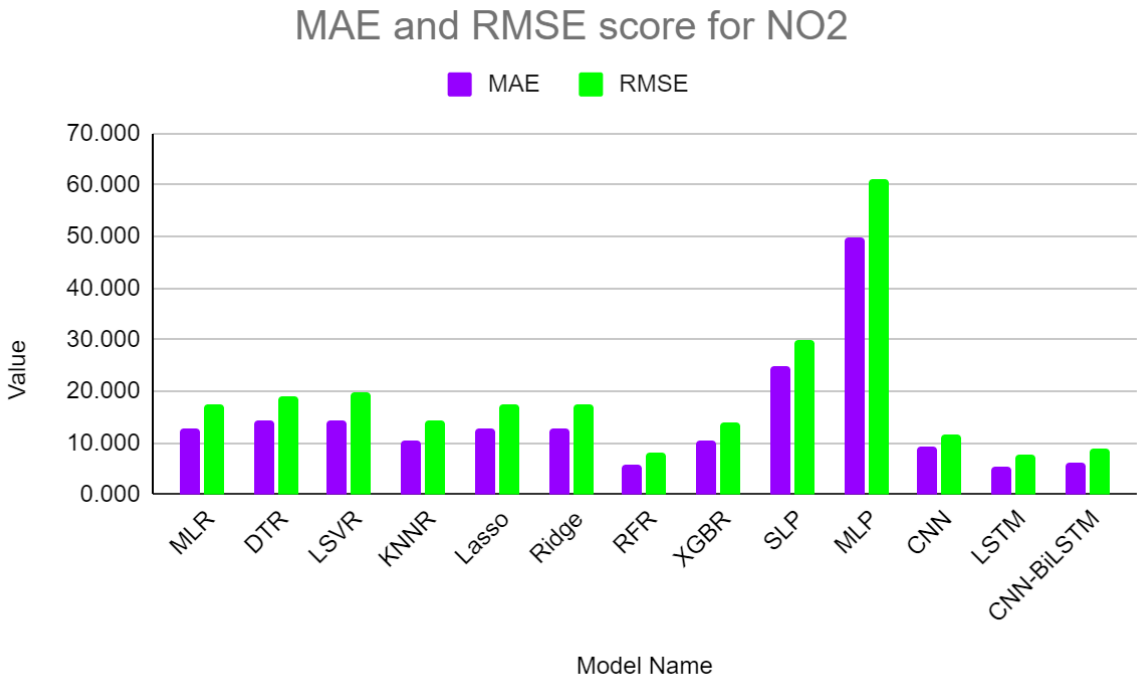


Figure 53: MAE and RMSE scores for NO2

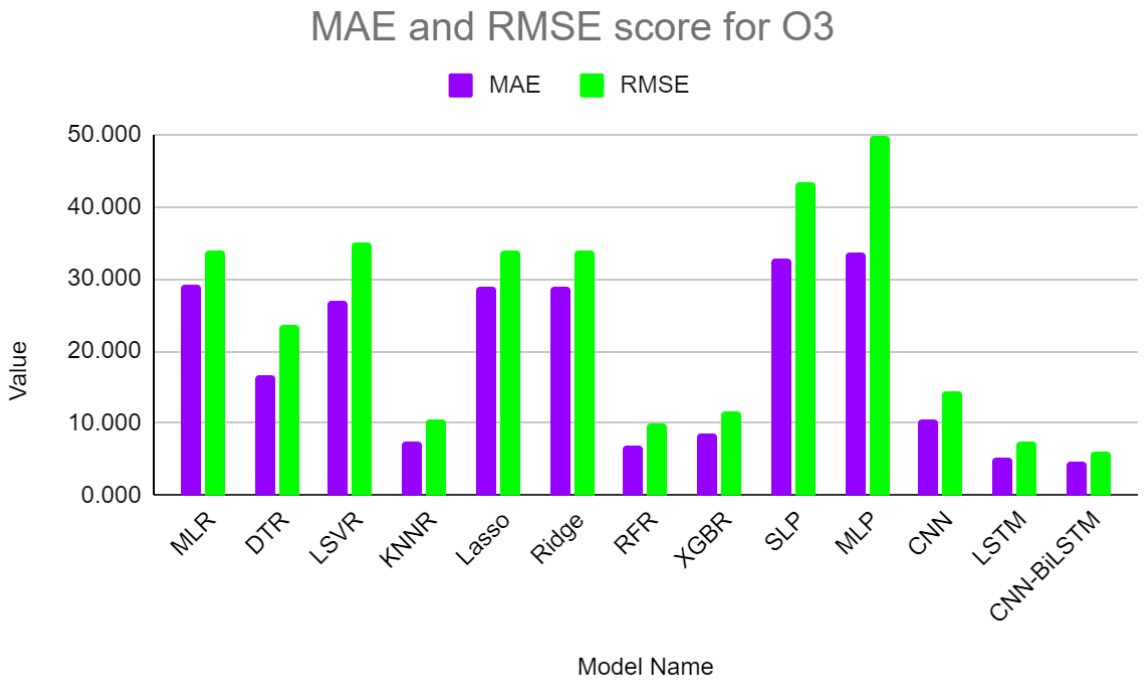


Figure 54: MAE and RMSE scores for O3

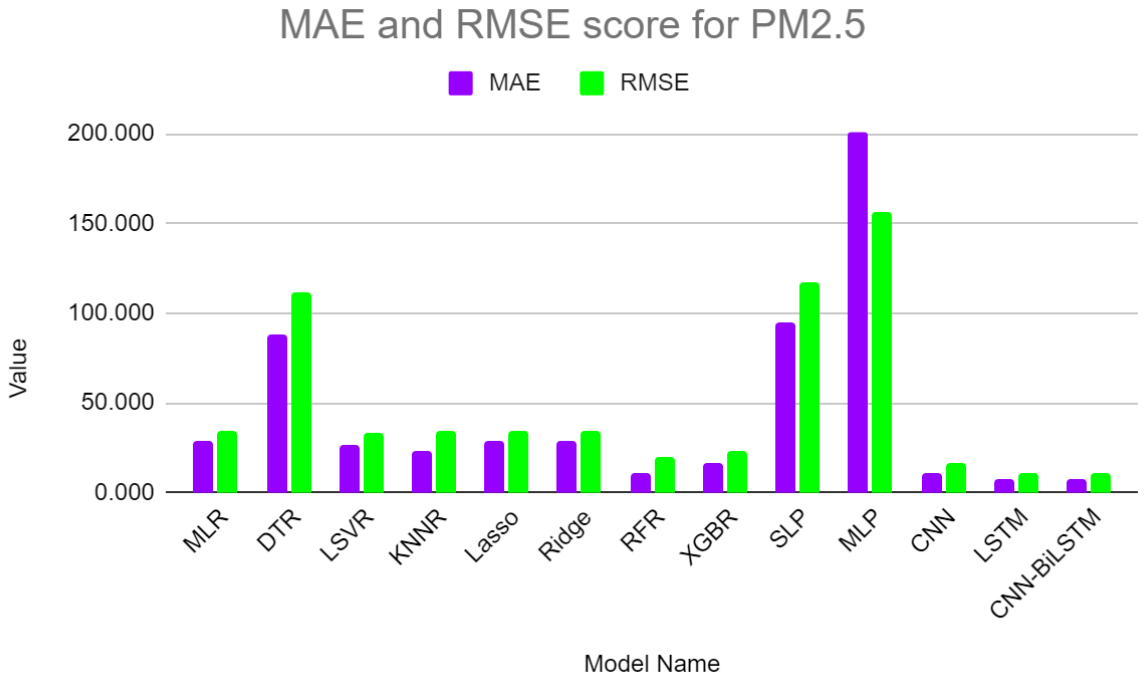


Figure 55: MAE and RMSE scores for PM2.5

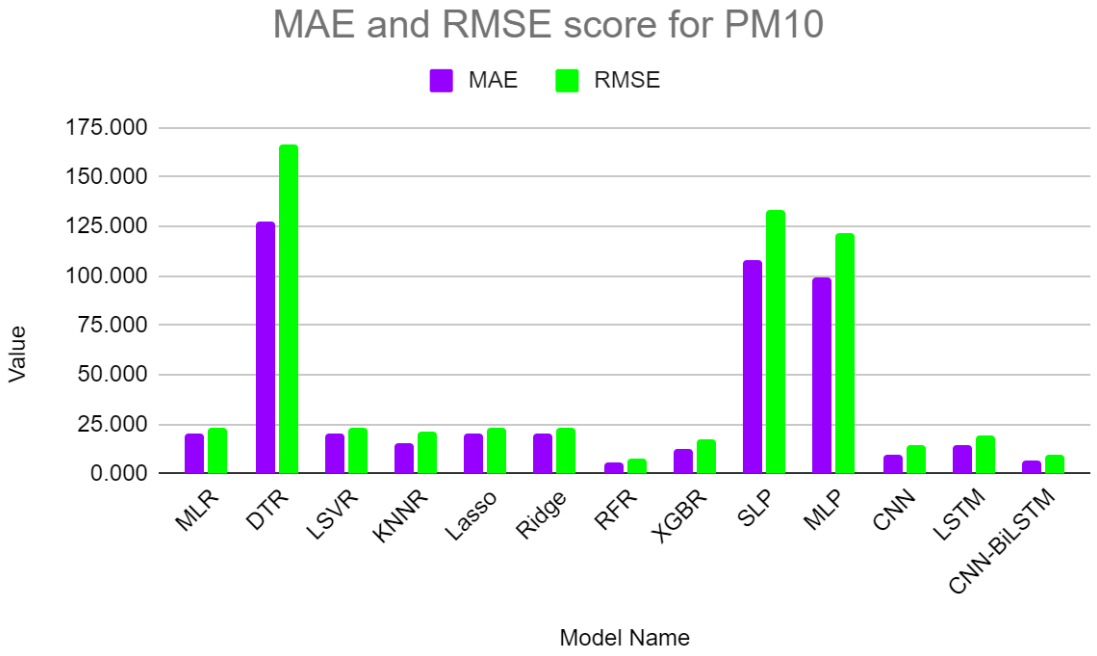


Figure 56: MAE and RMSE scores for PM10

9.3 CNN-BiLSTM Training and Hyperparameter Tuning Results

The CNN-BiLSTM model was tested and evaluated with varying parameters to find the best and most optimals for all features and for all metrics

9.3.1 Epoch and Iteration Testing

During training for the selected CNN-BiLSTM model, the model was run through multiple epochs, where epoch represents a complete pass through the training dataset. The validation data here as shown in figure 35 refers to the testing data that is used for validation purposes during the model's training. Each epoch is tested against model loss per epoch to determine how much loss decreases over a range of epochs. Typically, as can be seen in the graph results for NO2 below, the model losses take their steepest descent within the first three epochs with a steady but consistent decline in later epochs until epoch 8 where its clear both curves level off at around the same. To save computational resources and to speed up execution time, the epochs for the continuously training CNN-BiLSTM was kept at a fixed 8.

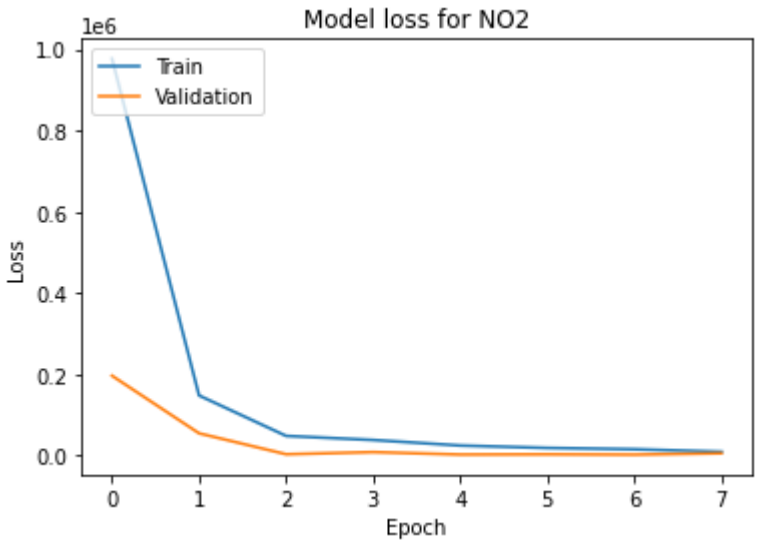


Figure 57: Loss vs Epochs

9.3.1 Current Model Architecture

The CNN-BiLSTM model was put through several tests to determine which parameters of the model including their range of CNN filters, LSTM layers, among others. The following represents the best model architecture post-testing.

Layer (type)	Output Shape	Param #
time_distributed_744 (TimeDistributed)	(None, 1, 1, 128)	768
time_distributed_745 (TimeDistributed)	(None, 1, 1, 128)	0
time_distributed_746 (TimeDistributed)	(None, 1, 128)	0
bidirectional_8 (Bidirectional)	(None, 2000)	9,032,000
dropout_6 (Dropout)	(None, 2000)	0
dense_2901 (Dense)	(None, 1)	2,001

Figure 58: Current architecture of Final CNN-BiLSTM

10. Project Global, Economic, Societal Impact

10.1 Global Impact

- **Environmental Sustainability:** The project's focus on low-power, cost-effective IoT air quality monitoring systems contributes to global environmental sustainability efforts. By reducing power consumption and providing real-time air quality data, it aids in mitigating environmental pollution and its impact on climate change.
- **Health and Well-Being:** The project's accurate and accessible air quality data has global implications for public health. By promoting informed decision-making, it can help reduce the health risks associated with air pollution, benefiting populations worldwide.

10.2 Economic Impact

- **Cost-Efficiency:** The project's emphasis on cost-effective components and low-power operation results in economic advantages. It reduces the financial burden associated with air quality monitoring for governments, organizations, and communities.
- **Industrial Advancement:** The development of affordable and energy-efficient air quality monitoring systems can stimulate economic growth in industries related to environmental technology and IoT. It opens opportunities for innovation and job creation in these sectors.

10.3 Societal Impact

- **Public Health:** The project's real-time air quality data empowers individuals and communities to make informed decisions regarding outdoor activities. This, in turn, enhances public health by reducing exposure to harmful air pollutants and minimizing the risks of respiratory and cardiovascular diseases.
- **Environmental Awareness:** The project fosters environmental awareness among the general public. By providing accessible air quality information, it encourages individuals to take steps toward a more sustainable and eco-conscious lifestyle, potentially reducing their carbon footprint.

11. Conclusion

11.1 Summary

In conclusion, our project emerges as a pioneering response to the critical challenge of inadequate air quality monitoring and prediction capabilities. With a dedicated focus on leveraging an innovative AIoT-based framework for air quality surveillance, our endeavor aligns seamlessly with the overarching goals of the UAE's commitment to environmental stewardship, specifically addressing the imperative outlined in the "Monitoring Pillar" of the UAE Air Quality Agenda 2031. The successful deployment and operation of our system within the confines of our university campus not only serve as a compelling proof of concept but also lay the groundwork for the realization of a scalable and encompassing solution poised to benefit the entirety of the UAE.

By strategically selecting and integrating suitable hardware components, such as the ESP-WROOM-32 MCU and a suite of specialized sensors including MQ131, MICS-6814, PMS5003,DHT-22, we have engineered a robust and energy-efficient system. The utilization of low-power microcontrollers and sensors is a testament to our commitment to sustainable technology implementation. Our choice of the CC3000 Wi-Fi module facilitates seamless communication, ensuring the efficient transmission of real-time air quality data to a cloud system. The adoption of Firebase as our cloud platform introduces a dynamic layer to our project, allowing for streamlined data storage, real-time updates, and the integration of machine learning models for predictive analysis. In addition to the hardware and software components, the incorporation of a 6W 5V Mini Solar Panel not only underscores our commitment to energy efficiency but also introduces a sustainable power source, aligning with global efforts to reduce the carbon footprint of technological implementations. From a software perspective, the project's intricate functionalities are orchestrated through a meticulously designed system of entities, functions, and interfaces. The integration of Flutter in our dashboard and website development, coupled with the incorporation of data visualization widgets enriches the user experience with interactive and insightful graphical representations of air quality metrics. As we move forward, the project sets the stage for potential extensions and future work, including the deployment of sensor nodes on a broader scale, incorporation of advanced machine learning models, the development of a mobile application for enhanced user interaction, and the exploration of additional environmental

factors such as weather data. In essence, our project is not merely a localized solution but a strategic step towards a comprehensive and scalable approach to air quality monitoring, in harmony with the vision of a cleaner and healthier environment outlined in the UAE Air Quality Agenda 2031. Through innovative technology, sustainability considerations, and strategic partnerships, our project endeavors to contribute meaningfully to the realization of a vision shared by environmental advocates and policymakers alike.

11.2 Future work

Our project lays the foundation for several potential extensions and future work areas. One avenue for expansion involves a more extensive deployment of sensor nodes across a broader geographical area. This can be achieved by implementing a multi-node deployment strategy, enhancing the coverage of air quality monitoring and providing more comprehensive datasets for analysis. Furthermore, exploring communication protocols for interconnected nodes can lead to a networked approach, allowing nodes to collaborate and share data, thereby contributing to a more detailed understanding of air quality patterns. In the realm of machine learning, the project can be extended by integrating more advanced models, particularly delving into deep learning techniques. The inclusion of deep neural networks can capture intricate patterns in historical data, potentially leading to more accurate air quality predictions. Real-time model training mechanisms could also be implemented, allowing the system to adapt to changing environmental conditions and continuously improve prediction accuracy.

User interaction and experience can be enhanced through the development of a mobile application. This application could provide real-time air quality information, personalized alerts, and location-based recommendations, particularly beneficial for individuals with health conditions affected by air quality. Additionally, incorporating a feedback system within the user interface allows users to contribute observations, which can be valuable for refining machine learning models and improving the accuracy of air quality predictions.The integration of real-time weather data presents another avenue for future work. Correlating air quality data with weather patterns can provide a more holistic understanding of the factors influencing air quality fluctuations. Furthermore, expanding the project to assess the impact of climate change on air quality by analyzing long-term trends contributes to a broader understanding of environmental conditions. To address sustainability, exploring advanced power management techniques is essential for optimizing the energy efficiency of the system.

This could involve the integration of more energy-efficient components and the exploration of alternative power sources. Investigating the feasibility of energy harvesting technologies, such as solar, wind, or kinetic energy harvesting, adds a dimension of sustainability to the system, reducing reliance on external power sources.

Finally, collaboration with environmental agencies is a key future work area. Initiatives such as data sharing and collaboration with relevant authorities can contribute to larger-scale environmental monitoring efforts. Moreover, actively engaging in policy recommendations based on the collected data can influence regional policies aimed at improving air quality. These potential extensions collectively offer opportunities for further development, scalability, and societal impact of the project. Depending on the project's goals and available resources, prioritizing one or more of these future work areas can guide subsequent phases of development.

References

[1] "A healthy lifestyle - WHO recommendations." Jun. 23, 2023. https://www.who.int/europe/news-room/fact-sheets/item/a-healthy-lifestyle---who-recommendations (accessed: Sep. 27, 2023).

[2] J. MacDonald Gibson, J. Thomsen, F. Launay, E. Harder, and N. DeFelice, "Deaths and Medical Visits Attributable to Environmental Pollution in the United Arab Emirates." PLoS ONE, vol. 8, no. 3, 2013, doi: 10.1371/journal.pone.0057536.

[3] H. Akasha, O. Ghaffarpasand, and F. D. Pope, "Climate Change, Air Pollution and the Associated Burden of Disease in the Arabian Peninsula and Neighbouring Regions: A Critical Review of the Literature." Sustainability, vol. 15, no. 4, p. 3766, 2023, doi: 10.3390/su15043766.

[4] R. AL-Dabbagh, "Dubai, the sustainable, smart city." Renewable Energy and Environmental Sustainability, vol. 7, p. 3, 2022, doi: 10.1051/rees/2021049.

[5] "UAE AIR QUALITY INDEX MANUAL." Sep. 20, 2018. [Online]. Available: https://www.moccae.gov.ae/assets/91c95f18/uae-air-quality-index-manual.aspx.

[6] "UAE National Air Quality Agenda 2031." Jun. 13, 2023. [Online]. Available: https://www.moccae.gov.ae/assets/download/b989a286/UAE%20National%20Air%20Quality%20Agenda%202031.pdf.aspx.

[7] J. Shah and B. Mishra, "IOT-enabled low power environment monitoring system for prediction of PM2.5," Pervasive and Mobile Computing, vol. 67, p. 101175, 2020. doi:10.1016/j.pmcj.2020.101175

[8] T. Villa, F. Gonzalez, B. Miljievic, Z. Ristovski, and L. Morawska, "An Overview of Small Unmanned Aerial Vehicles for Air Quality Measurements: Present Applications and Future Prospectives," *Sensors*, vol. 16, no. 7, p. 1072, Jul. 2016, doi: https://doi.org/10.3390/s16071072.

[9] National Center of Meteorology (NCM), "NCM.AE," NCM.ae - National Center of Meteorology. https://www.ncm.ae/maps-radars/uae-radars-network?lang=ar

[10] J. -Y. Kim, C. -H. Chu and S. -M. Shin, "ISSAQ: An Integrated Sensing Systems for Real-Time Indoor Air Quality Monitoring," in IEEE Sensors Journal, vol. 14, no. 12, pp. 4230-4244, Dec. 2014, doi: 10.1109/JSEN.2014.2359832.

[11] A. Kumar and G. P. Hancke, "Energy Efficient Environment Monitoring System Based on the IEEE 802.15.4 Standard for Low Cost Requirements," in IEEE Sensors Journal, vol. 14, no. 8, pp. 2557-2566, Aug. 2014, doi: 10.1109/JSEN.2014.2313348.

[12] S. C. Folea and G. Mois, "A Low-Power Wireless Sensor for Online Ambient Monitoring," *IEEE Sensors Journal*, vol. 15, no. 2, pp. 742–749, Feb. 2015, doi: https://doi.org/10.1109/jsen.2014.2351420.

[13] J. C. Hall *et al.*, "A Portable Wireless Particulate Sensor System for Continuous Real-Time Environmental Monitoring," *42nd International Conference on Environmental Systems*, Jul. 2012, doi: https://doi.org/10.2514/6.2012-3441.

[14] H. P. L. de Medeiros and G. Girão, "An IoT-based Air Quality Monitoring Platform," *IEEE Xplore*, Sep. 01, 2020. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9239070 (accessed May 02, 2022).

[15] J. Shah and B. Mishra, "Customized IoT Enabled Wireless Sensing and Monitoring Platform for Smart Buildings," *Procedia Technology*, vol. 23, pp. 256–263, 2016, doi: https://doi.org/10.1016/j.protcy.2016.03.025.

[16] S. Ali, T. Glass, B. Parr, J. Potgieter, and F. Alam, "Low Cost Sensor with IoT LoRaWAN Connectivity and Machine Learning Based Calibration for Air Pollution Monitoring," *IEEE Transactions on Instrumentation and Measurement*, pp. 1–1, 2020, doi: https://doi.org/10.1109/tim.2020.3034109.

[17] P. Das, S. Ghosh, S. Chatterjee, and S. De, "A Low Cost Outdoor Air Pollution Monitoring Device With Power Controlled Built-In PM Sensor," *IEEE Sensors Journal*, vol. 22, no. 13, pp. 13682–13695, Jul. 2022, doi: https://doi.org/10.1109/JSEN.2022.3175821.

[18] K. Laubhan, Khaled Talaat, S. Riehl, S. Aman, A. Abdelgawad, and Kumar Yelamarthi, "A low-power IoT framework: From sensors to the cloud," May 2016, doi: https://doi.org/10.1109/eit.2016.7535315.

[19] Y. Wang, Y. Huang, and C. Song, "A New Smart Sensing System Using LoRaWAN for Environmental Monitoring," Oct. 2019, doi: https://doi.org/10.1109/comcomap46287.2019.9018829.

[20] Person, V. Mikhail, and T. Kanevski, “Machine learning for spatial environmental data: Theory, applications,” Taylor & Francis, https://www.taylorfrancis.com/books/mono/10.1201/9781439808085/machine-learning-spatial-environmental-data-mikhail-kanevski-vadim-timonin-alexi-pozdnukhov (accessed Nov. 13, 2023).

[21] K. P. Singh, S. Gupta, A. Kumar, and S. P. Shukla, “Linear and nonlinear modeling approaches for urban air quality prediction,” Science of the Total Environment, vol. 426, pp. 244–255, Jun. 2012, doi: 10.1016/j.scitotenv.2012.03.076.

[22] S. B. Sonu and A. Suyampulingam, "Linear Regression Based Air Quality Data Analysis and Prediction using Python," 2021 IEEE Madras Section Conference (MASCON), Chennai, India, 2021, pp. 1-7, doi: 10.1109/MASCON51689.2021.9563432.

[23] N. Redell, “Shapley decomposition of R-squared in machine learning models,” arXiv.org, https://arxiv.org/abs/1908.09718 (accessed Nov. 17, 2023).

[24] N. Dahiya, S. Gupta, and S. Singh, “A review paper on Machine learning Applications, Advantages, and techniques,” ECS Transactions, vol. 107, no. 1, pp. 6137–6150, Apr. 2022, doi: 10.1149/10701.6137ecst.

[25] D. Varghese, “Comparative Study on Classic Machine learning Algorithms,” Medium, Dec. 06, 2021. [Online]. Available: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222#:~:text=SVM%20can%20handle%20non%2Dlinear,outperforms%20log%20loss%20in%20LR.

[26]“Support Vector Machine (SVM) algorithm,” GeeksforGeeks, https://www.geeksforgeeks.org/support-vector-machine-algorithm/ .

[27] K. M. A. Bola Pavan Kumar, “Linear regression vs. support vector machine: A formal enhancement in online purchase utilizing novel customer feedback segmentation,” Journal of SurveyinFisheriesSciences,https://sifisheriessciences.com/journal/index.php/journal/article/view/420.

[28] W. C. Leong, R. O. Kelani, and Z. A. Ahmad, “Prediction of air pollution index (API) using support vector machine (SVM),” Journal of Environmental Chemical Engineering, vol. 8, no. 3, p. 103208, Jun. 2020, doi: 10.1016/j.jece.2019.103208.

[29] R. Nainggolan, R. Perangin-angin, E. Simarmata, and A. F. Tarigan, “IOPscience,” Journal of Physics: Conference Series, https://iopscience.iop.org/article/10.1088/1742-6596/1361/1/012015.

[30] T. Atwan, “Methods for testing linear separability in Python,” TechTalks, https://www.tarekatwan.com/index.php/2017/12/methods-for-testing-linear-separability-in-python/

[31] I. Iwok and A. Okpe, "A Comparative Study between Univariate and Multivariate Linear Stationary Time Series Models," American Journal of Mathematics and Statistics, vol. 2016, pp. 203-212, Jul. 2016. doi: https://doi.org/10.5923/j.ajms.20160605.02.

[32] P. K. Gabriel, "On the application of Multivariate Times Series Models," Journal of Physical Science and Technology, vol. 8, no. 10, pp. 51-62, 2015. ISBN: 3-4428-321-9.

[33] Baolei Lv, W. Geoffrey Cobourn, and Y. Bai, “Development of linear empirical models to forecast daily PM2.5 and ozone levels in three large Chinese cities,” vol. 147, pp. 209–223, Dec. 2016, doi: https://doi.org/10.1016/j.atmosenv.2016.10.003.

[34] M. Babyak, “ A Brief, Nontechnical Introduction to Overfitting in Regression-Type Models,” Duke University, Apr. 2004. Available: https://people.duke.edu/~mababyak/papers/babyakregression.pdf

[35] J. H. Kim, “Multicollinearity and misleading statistical results,” *Korean Journal of Anesthesiology*, vol. 72, no. 6, pp. 558–569, Jul. 2019, doi: https://doi.org/10.4097/kja.19087.

[36] W. C. Leong, R. O. Kelani, and Z. Ahmad, “Prediction of air pollution index (API) using support vector machine (SVM),” *Journal of Environmental Chemical Engineering*, vol. 8, no. 3, p. 103208, Jun. 2020, doi: https://doi.org/10.1016/j.jece.2019.103208.

[37] T. H. Abraham, "(Physio)logical circuits: The intellectual origins of the McCulloch-Pitts neural networks," Journal of the History of the Behavioral Sciences, vol. 38, no. 1, pp. 3–25, 2002, doi: https://doi.org/10.1002/jhbs.1094.

[38] Steffanie Van Roode, Juan Jesús Ruiz-Aguilar, J. González-Enrique, and I. J. Turias, "An artificial neural network ensemble approach to generate air pollution maps," *Environmental Monitoring and Assessment*, vol. 191, no. 12, Nov. 2019, doi: https://doi.org/10.1007/s10661-019-7901-6.

[39] H. Maleki, A. Sorooshian, G. Goudarzi, Z. Baboli, Y. Tahmasebi Birgani, and M. Rahmati, "Air pollution prediction by using an artificial neural network model," Clean Technologies and Environmental Policy, vol. 21, no. 6, pp. 1341–1352, May 2019, doi: https://doi.org/10.1007/s10098-019-01709-w

[40] C. H. Cordova, M. N. L. Portocarrero, R. Salas, R. Torres, P. C. Rodrigues, and J. L. López-Gonzales, "Air quality assessment and pollution forecasting using artificial neural networks in Metropolitan Lima-Peru," Scientific Reports, vol. 11, no. 1, Dec. 2021, doi: https://doi.org/10.1038/s41598-021-03650-9.

[41] S. Cakir and M. Sita, "Evaluating the performance of ANN in predicting the concentrations of ambient air pollutants in Nicosia," Atmospheric Pollution Research, vol. 11, no. 12, pp. 2327–2334, Dec. 2020, doi: https://doi.org/10.1016/j.apr.2020.06.011

[42] G. K. Uyanık and N. Güler, "A Study on Multiple Linear Regression Analysis," *Procedia - Social and Behavioral Sciences*, vol. 106, no. 1, pp. 234–240, Dec. 2013, doi: https://doi.org/10.1016/j.sbspro.2013.12.027.

[43] W. W. Hsieh and B. Tang, "Applying Neural Network Models to Prediction and Data Analysis in Meteorology and Oceanography," American Meteorological Society, vol. 79, no. 9, pp. 1855–1870, Sep. 1998, doi: https://doi.org/10.1175/1520-0477(1998)079%3C1855:ANNMTP%3E2.0.CO;2.

[44] J. Chu, X. Liu, Z. Zhang, Y. Zhang, and M. He, "A novel method overcomeing overfitting of artificial neural network for accurate prediction: Application on thermophysical

property of natural gas," Case Studies in Thermal Engineering, vol. 28, p. 101406, Dec. 2021, doi: https://doi.org/10.1016/j.csite.2021.101406.

[45] G. Corani, "Air quality prediction in Milan: feed-forward neural networks, pruned neural networks and lazy learning," Ecological Modelling, vol. 185, no. 2–4, pp. 513–529, Jul. 2005, doi: https://doi.org/10.1016/j.ecolmodel.2005.01.008.

[46] F. Şahin, G. Işik, G. Şahin, and M. K. Kara, "Estimation of PM10 levels using feed forward neural networks in Igdir, Turkey," Urban Climate, vol. 34, p. 100721, Dec. 2020, doi: https://doi.org/10.1016/j.uclim.2020.100721.

[47] S. Abdullah, M. Ismail, and A.-M. Najah, "Multi-Layer Perceptron Model for Air Quality Prediction," Malaysian Journal of Mathematical Sciences, vol. 13, pp. 85–95, Dec. 2019,Available:https://www.researchgate.net/publication/339088619_Multi-Layer_Perceptron_Model_for_A ir_Quality_Prediction

[48] A. Rahimi, "Short-term prediction of NO2 and NO x concentrations using multilayer perceptron neural network: a case study of Tabriz, Iran," Ecological Processes, vol. 6, no. 1, Jan. 2017, doi: https://doi.org/10.1186/s13717-016-0069-x.

[49] D. Voukantsis, H. Niska, K. Karatzas, M. Riga, A. Damialis, and D. Vokou, "Forecasting daily pollen concentrations using data-driven modeling methods in Thessaloniki, Greece," Atmospheric Environment, vol. 44, no. 39, pp. 5101–5111, Dec. 2010, doi: https://doi.org/10.1016/j.atmosenv.2010.09.006.

[50] G. Chattopadhyay and S. Chattopadhyay, "Autoregressive forecast of monthly total ozone concentration: A neurocomputing approach," Computers & Geosciences, vol. 35, no. 9, pp. 1925–1932, Sep. 2009, doi: https://doi.org/10.1016/j.cageo.2008.11.007.

[51] A. Chaloulakou, G. Grivas, and N. Spyrellis, "Neural Network and Multiple Regression Models for PM_{10} Prediction in Athens: A Comparative Assessment," *Journal of the Air & Waste Management Association*, vol. 53, no. 10, pp. 1183–1190, Oct. 2003, doi: https://doi.org/10.1080/10473289.2003.10466276.

[52] A. Chaloulakou, P. Kassomenos, N. Spyrellis, P. Demokritou, and P. Koutrakis, "Measurements of PM10 and PM2.5 particle concentrations in Athens, Greece," *Atmospheric Environment*, vol. 37, no. 5, pp. 649–660, Feb. 2003, doi: https://doi.org/10.1016/s1352-2310(02)00898-1.

[53] Z. Car, S. Baressi Šegota, N. Anđelić, I. Lorencin, and V. Mrzljak, "Modeling the Spread of COVID-19 Infection Using a Multilayer Perceptron," *Computational and Mathematical Methods in Medicine*, vol. 2020, p. 5714714, 2020, doi: https://doi.org/10.1155/2020/5714714.

[54] M. Gardner, "Neural network modelling and prediction of hourly NOx and NO2 concentrations in urban air in London," *Atmospheric Environment*, vol. 33, no. 5, pp. 709–719, Feb. 1999, doi: https://doi.org/10.1016/s1352-2310(98)00230-1.

[55] S. A. Marhon, C. J. F. Cameron, and S. C. Kremer, "Recurrent Neural Networks," *Intelligent Systems Reference Library*, pp. 29–65, 2013, doi: https://doi.org/10.1007/978-3-642-36657-4_2.

[56] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, doi: https://doi.org/10.1109/tnnls.2016.2582924.

[57] S. V. Belavadi, S. Rajagopal, R. R, and R. Mohan, "Air Quality Forecasting using LSTM RNN and Wireless Sensor Networks," *Procedia Computer Science*, vol. 170, pp. 241–248, 2020, doi: https://doi.org/10.1016/j.procs.2020.03.036.

[58] "Open Government Data (OGD) Platform India," *data.gov.in*, Jan. 21, 2022. https://tn.data.gov.in/catalog/real-time-air-quality-index (accessed Nov. 17, 2023).

[59] J. Wang, J. Li, X. Wang, J. Wang, and M. Huang, "Air quality prediction using CT-LSTM," *Neural Computing and Applications*, vol. 33, no. 10, pp. 4779–4792, Nov. 2020, doi: https://doi.org/10.1007/s00521-020-05535-w.

[60] J. Teuwen and N. Moriakov, “Convolutional neural networks,” *Handbook of Medical Image Computing and Computer Assisted Intervention*, pp. 481–501, 2020, doi: https://doi.org/10.1016/b978-0-12-816176-0.00025-9.

[61] H. Yue, L. Duan, M. Lu, H. Huang, X. Zhang, and H. Liu, “Modeling the Determinants of PM2.5 in China Considering the Localized Spatiotemporal Effects: A Multiscale Geographically Weighted Regression Method,” *Atmosphere*, vol. 13, no. 4, p. 627, Apr. 2022, doi: https://doi.org/10.3390/atmos13040627.

[62] R. Chauhan, H. Kaur, and B. Alankar, “Air Quality Forecast using Convolutional Neural Network for Sustainable Development in Urban Environments,” *Sustainable Cities and Society*, vol. 75, p. 103239, Dec. 2021, doi: https://doi.org/10.1016/j.scs.2021.103239.

[63] Qin, D. et al. (no date) A novel combined prediction scheme based on CNN and LSTM for Urban PM2. Available at: https://ieeexplore.ieee.org/document/8632898 (Accessed: 07 November 2023).

[64] Wang, J. et al. (2022) An air quality index prediction model based on CNN-ILSTM, Scientific reports. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9120089/ (Accessed: 08 November 2023).

[65] P. Kapoor and F. A. Barbhuiya, "Cloud Based Weather Station using IoT Devices," TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), Kochi, India, 2019, pp. 2357-2362, doi: 10.1109/TENCON.2019.8929528.

[66] V. N. Hidayati, Iskandar and A. B. Satriobudi, "Web Dashboard Development for Cloud Server-Based Air Quality Monitoring System," 2022 16th International Conference on Telecommunication Systems, Services, and Applications (TSSA), Lombok, Indonesia, 2022, pp. 1-5, doi: 10.1109/TSSA56819.2022.10063897.